Core concepts and methods in load forcasting

Stephen Haben
Marcus Voss
William Holderbaum

Core Concepts and Methods in Load Forecasting

With Applications in Distribution Networks

Springer

Core Concepts and Methods in Load Forecasting

Stephen Haben · Marcus Voss ·
William Holderbaum

Core Concepts and Methods in Load Forecasting

With Applications in Distribution Networks

Springer

Stephen Haben
Mathematical Institute
University of Oxford
Oxford, Oxfordshire, UK

William Holderbaum
Engineering
University of Reading
Reading, UK

Marcus Voss
Faculty IV—Electrical Engineering
and Computer Science
TU Berlin
Berlin, Germany

ISBN 978-3-031-27851-8 ISBN 978-3-031-27852-5 (eBook)
https://doi.org/10.1007/978-3-031-27852-5

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

One of the oldest and most cherished desires for any quantitative scientific and technological field is the ability to develop forecasts of the, as yet, unforeseen. Of course, whatever has been observed already is, in most applications, a moving conveyor belt. Hence, this ambition raises fundamental questions, such as *What actually is a good forecast?*, and *What is the value of older data?*.

Many realistic challenges involve spikey data, perhaps within nonstationary (drifting and step change) settings. In turn, this fact modifies the answers to the questions above. If we aim to predict realistic forward profiles, as opposed to point estimates, we shall need to generate realistic structures (with spikes, gaps, and so on). This is problematical for methods that treat errors as a noise component that is to be minimised (in some way). It presents a meta problem.

This book is timely and very much needed. From my own experience, I see that whenever novel concepts are posited so as to address some of these issues, they are eagerly feasted upon and further developed by many scientists within a plethora of applied fields. The breadth of ideas and methods here is what gives this book its power: readers will return to it again and again. For data science applications, there is a need for options that address "what works" as well as "why it works". In many fields of highly regulated sciences pertaining to the interests of public individuals, it is simply unethical to present opaque methods; *why* is as essential as *what*. This provides transparency and assurance to the subject who feels the consequences of the forecasting outputs.

No serious professional data scientist can be oblivious to the contents presented here. The interests of readers should be both refined and peaked by dipping into this book. Open it and read at random (like a grasshopper), or crawl through it (like an ant): your investment of both interest and effort will be rewarded. Now that is a forecast!

Oxford, UK
January 2023

Peter Grindrod CBE

Preface

Electricity networks around the world are rapidly moving towards digitalisation, producing an ever-increasing amount of data. This data is opening up vast opportunities to decarbonise the energy system, as well as helping us to increase the efficiency of the energy we use. In turn, this improves the chances of addressing some of the most urgent problems causing climate change.

To implement the necessary data analytical and modelling techniques for the future, low carbon economy requires a wide range of skills, knowledge, and data literacy. Without these, there is a genuine threat of a skills gap where there is insufficient personnel who can put into practice these methods and models. One of the main goals of this book is to help support these vital skills by providing an accessible but thorough introduction to the techniques required for household and low voltage load forecasting.

The area of load forecasting at the Low Voltage (LV) level is relatively immature compared to high voltage or national level forecasting, and there are many important and exciting areas still to be explored. The volatile and spiky nature of LV level demand provides many challenges within applications which utilise forecast inputs, but also within the forecast themselves. Hence, a second major aim of this book is to lay a strong foundation for researchers and innovators to develop the future novel methods and advanced algorithms that can produce ever more varied products and services.

The seeds of this book started over 10 years ago when the authors began their research into forecasting smart meters and low voltage demand. At the time there was very little data available in this area, and its unique challenges such as the "double penalty effect" were hidden or ignored. Much research simply applied the techniques which had been successfully applied at the national or system level, with very little thought to their appropriateness to the LV system. The area is now rapidly progressing and starting to embrace the much-needed probabilistic techniques and advanced time series machine learning methods. Additionally, more data is becoming available all the time, further supporting the development of robust benchmarks and new applications.

Although data science techniques are rapidly developing in LV level load forecasting (and will continue to), this book provides the fundamental techniques which serve as the foundation for anyone interested in this area (and load forecasting more generally), as well as the timeless, but necessary, principles and approaches underlying them. Discussed in this book are the core concepts of time series forecasting, the unique features of LV level demand, fundamental data analytics, methods for feature engineering, a plethora of statistical and machine learning forecast models, as well as a demonstration of them in a case study applied to real-world data. At the end of this book, the reader should be well versed on the complete load forecast process, as well, perhaps, ready to develop some novel models of their own!

It was our desire to produce a book "we wished was available when we were first starting out in this field". We feel that we have achieved this, and we hope that it supports you on your journey into load forecasting.

Oxford, UK — Stephen Haben
Berlin, Germany — Marcus Voss
Reading, UK — William Holderbaum
January 2023

Acknowledgments

We would like to thank the University of Reading, who funded the Open Access publication of this book. The authors would also like to thank all the colleagues who have contributed to the content of this book either directly or indirectly. Much of the knowledge and work presented here have come from the authors's research, collaborations, and discussions. In particular a special thanks to Peter Grindrod, Danica Vukadinovic Greetham, Colin Singleton, Billiejoe Charlton, Florian Ziel, Ben Potter, Timur Yunusov, JonathanWard, Laura Hattam, Lauren Barrett, Matthew Rowe, Bruce Stephen, David O'Sullivan, Gideon Evans, Maciej Fila, Wayne Travis, Rayner Mayer, Asmaa Haja, Marcel Arpogaus, Alexander Elvers, Brijnesh Jain, and Daniel Freund. Furthermore, the authors would like to give special thanks to Siddharth Arora, Charlotte Avery, Jethro Browell, Alison Halford, Sam Young, and especially Georgios Giasemidis for their time to provide invaluable feedback on drafts of the book.

The Case Study in the book will be based on real data from low voltage residential feeders from the Thames Valley Vision project. We would like to thank Scottish and Southern Electricity Networks and our partners who helped support this work throughout the project.

The author would also like to thank the following journals that have allowed the use of figures and material published by the authors in this book:

1. International Journal of Forecasting [35] which provides the basis for the LV residential feeder forecasting case study in Chap. 14.
2. Springer International Publishing for permitting the use of material from Chapter [85] which is used to demonstrate the storage control example in Chap. 15.
3. Applied Energy for use of the Network diagram in Chap. 2 which was originally plotted in [34].

Contents

Chapter 1
Introduction

This chapter introduces the context and motivation for this book as well as some of the best ways to use it. It focuses on:

- Why is forecasting needed to help support the future energy system, especially at the low voltage level.
- Why is this book needed.
- Description of the contents and how to read it.
- What to include in a semester long course.

1.1 Motivation

Mitigation of the climate crisis by meeting global carbon reduction targets is going to require dramatic changes to how we generate and use energy. Global leaders and the energy sector must act with urgency if we are to avoid the most catastrophic outcomes of climate change.

Carbon reduction will mean a continuous shift towards the decarbonisation of major sectors and infrastructures, especially in heating, industry and transport. Many of these applications will require electrification with, for example, heating moving from gas boilers to electric heat pumps, and transport moving from petrol combustion engines to electric vehicles.[1]

This electrification is increasingly generated from renewable energy sources, such as wind, solar, wave, hydro and geothermal (See Fig. 1.1). The change in how energy is used and generated has two main consequences. First, larger and more volatile demand consumption behaviour due to new low carbon appliances (electric vehicle charging at home), and second, an electricity supply which has much more uncertainty due to the dependence on less predictable and intermittent weather behaviour.

[1] Other applications, especially industrial applications, may move to utilising hydrogen.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_1

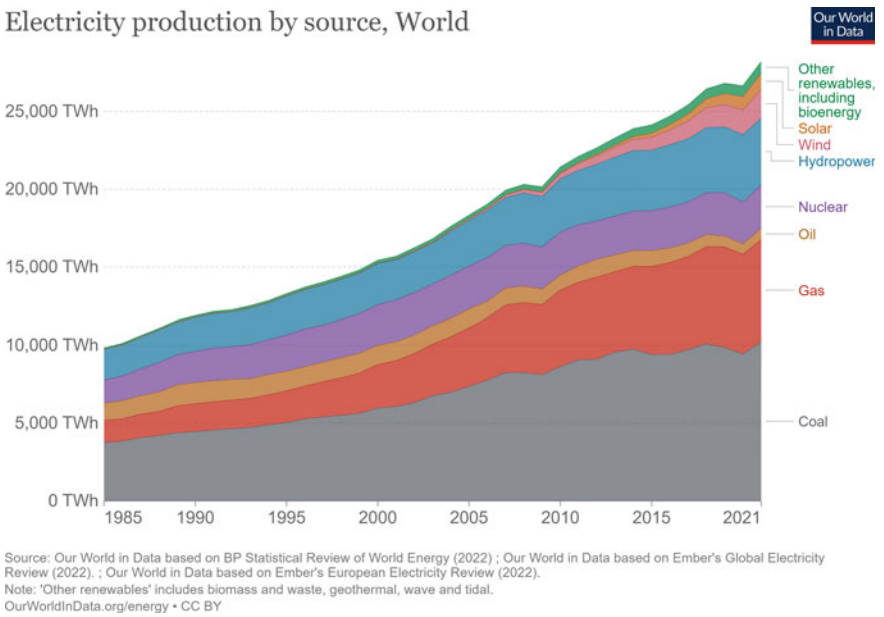


Fig. 1.1 Proportion of electricity produced per year broken down by source. Plot by Our World in Data licensed under CC BY 4.0

These changes will effect the security of the electricity supply since it increases the chances the energy generated may not match the demand of the end consumers, which in turn could result in blackouts or damage to the electricity networks. These are going to present significant challenges for governments to "ensure access to affordable, reliable, sustainable and modern energy for all" which is one of the 17 Sustainable Development Goals established by the United Nations General Assembly in 2015.[2] To mitigate the effects of electrification several solutions are being developed including:

1. **Control of Storage**: storage devices can "shift" and "smooth" demand by charging during times of low demand, and discharging during periods of high demand. This can help reduce the effects of increasing demand but can also ensure optimal utilisation of renewable energy at time periods when they are needed most. Most renewable generation is dependent on weather conditions which means energy from renewable sources may be generated when it is least needed. Storing this generation can ensure that clean energy is used to meet the high demand instead of energy with higher carbon intensity from the grid.
2. **Demand Side Response**: If there is insufficient generation to match the demand, devices can be turned off to balance the network. For example, heat pumps could

[2] See https://sdgs.un.org/goals/goal7.

be turned off for a short period, reducing demand whilst ensuring heating comfort is retained. Similarly if there is too much generation demand can be turned on.
3. **Co-ordinated EV Charging**: If Electric Vehicle (EV) uptake is sufficiently high then there is a likelihood that local networks will be under excessive strain since households will choose to charge at similar periods (for example, plugging them in after arriving home from work, or charging over night so the vehicle is ready for use in the morning). One way to alleviate this effect is to co-ordinate their charging so that fewer vehicles are drawing energy from the grid simultaneously.
4. **Local Electricity Markets**: With more and more distributed generation, and an increase in controllable assets with two-way communications it creates opportunities for smart grids and localised energy markets where demand and generation can be traded. Energy can be utilised close to where it is generated and this also reduces the losses through transmitting energy over large distances.

All of these applications require varying degrees of foresight of the demand or generation in order to operate optimally and ensure minimal disruption and costs to consumers. For example, consider a storage device with the objective to maximally charge using energy generated from renewables, and to use the stored energy to reduce the peak energy usage. This requires accurate estimates of both the future demand and generation to schedule the appropriate charging and discharging of the device. These estimates are produced by so-called **load forecasts**, the topic of this book!

1.2 Demand Forecasting for LV Systems

As described above accurate forecasting is a vital tool for a whole range of applications including optimising energy management systems (such as storage devices), redistributing demand, peak demand reduction and electrical infrastructure development (i.e. determining where and what assets to install). Unfortunately, load forecasting at the LV level is a complex task compared to forecasting at the national level or system level. The small number of consumers connected to the LV substations mean that the demand is relatively irregular and volatile. Feeders on low voltage substations often consist of between 1 and 100 consumers and they may be residential, commercial, or a mix of both. Further to this there is street furniture such as lighting which can have a non-trivial effect on the overall demand at the LV level.

This volatility is apparent when comparing the half hourly demand over a week for different aggregations of households. This is shown in Fig. 1.2 for a single consumer, an aggregation of 40 consumers (the size of a typical LV secondary feeder) and finally, 540 consumers. At the highest aggregation the profiles are relatively smooth with the demand very similar from one day to the next. In contrast the single household is very volatile and irregular from day to day. LV feeders will typically be connected to around 40 consumers, and it is clear that although they are more regular than a single household, they still have significant levels of volatility and uncertainty. This book

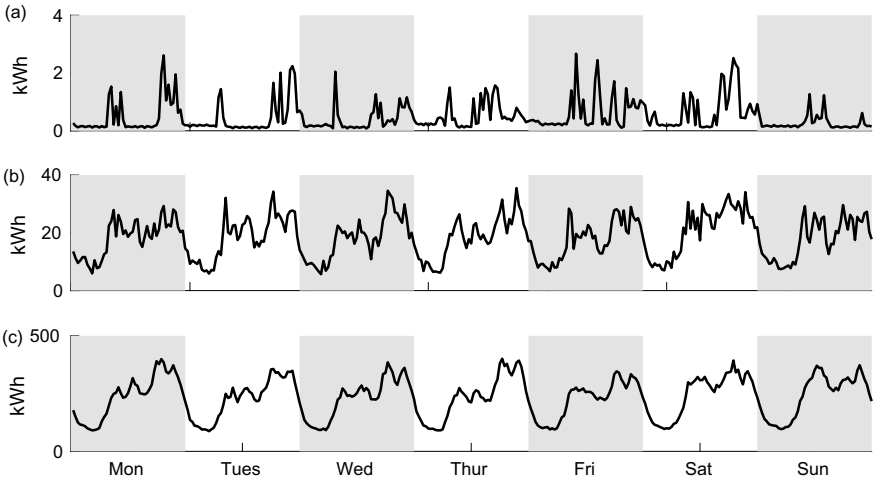


Fig. 1.2 Examples of half hourly demand profiles for a week for **a** a single household, **b** the aggregation of 40 households, **c** the aggregation of 540 households. Constructed using data from the CER Smart Metering Project—Electricity Customer Behaviour Trial, 2009–2010 [1]

will present a case study in Chap. 14 comparing forecasts on feeders of a range of sizes, showing how aggregation links to accuracy. This variety in demand behaviour also means there is no one-size-fits-all forecast model which will forecast accurately across all LV substations, unlike systems-level or national-level forecasting.

In this book, forecast models will be used to predict the demand as both point estimates and probabilistic estimates. The latter is essential to deal with the uncertainty inherent in LV level demand and ensure the optimal performance is obtained for the applications mentioned above and in this book.

1.3 Why Do We Need This Book

As outlined above, forecasting is an essential requirement to support the changing dynamics of the electricity network, and the emerging applications and opportunities. However, the unique challenges associated to low voltage demand, in particular, the increased volatility and irregularity, will require knowledge of advanced techniques and models which are not necessarily required for the more regular demand at the system or national level.

Unfortunately knowledge and experience in these areas are relatively sparse due to the lack of available data, and the recency of the area itself. Monitoring of demand is generally reserved for higher voltage levels of the network and smart meters have only been rolled out from the 2010s onwards. There is therefore a urgent requirement for these indispensable skills to enable the growing needs of a low carbon economy.

Forecasts are the fundamental component to a vast array of applications in the smart grid and low voltage level. This book will therefore enable researchers in a vast array of fields from optimisation, control theory, power systems engineering, and uncertainty quantification. Due to the array of techniques presented, it will also be useful as a reference to more experienced forecasters who may wish to utilise further approaches and models.

Several applications are illustrated in both the case study (Chap. 14) and the further examples (Chap. 15). These are based upon the authors' research experience using real world data to demonstrated the techniques presented. The case study in particular shows all the difficulties that can be found with processing, analysing and interpreting of real measured data, and utilising them within state-of-the-art forecast models. This is in contrast to other books which often use toy examples, which although illustrative, perhaps do not demonstrate the sometimes subtle complications which are prevalent in real systems and data.

1.4 Aims and Overview of This Book

This book aims to present a comprehensive guide to all the components necessary to develop accurate and effective load forecasts for low voltage electricity networks. The behaviour of electricity demand has changed with new energy sources such as energy storage and renewable energies sources which aim to supply the increase in energy demand. New energy control technologies have been shown to reduce energy costs, emissions, and peak demand, and load forecasting characteristics provide the opportunity to investigate the benefits of optimal control strategies. While not providing exhaustive details on all approaches, this book's aim is to provide a unique overview of the most commonly applied techniques and methods for load forecasting at the LV level. It covers the basic time series forecasting concepts and introduces both, common statistical and machine learning methods. The reader is referred to more detailed literature on each of the methods in the Appendix.

Before developing precise forecast techniques and models, a practitioner will require understanding fundamental concepts in energy systems, statistics and machine learning. For these reasons, Chaps. 2, 3 and 4 serve as knowledge foundation for the rest of the book and any topic which requires more information will be elaborated on in later chapters. Chapter 2 briefly describes the important features of a electricity distribution network, including those attributes which make LV level forecasting much more challenging than system level. Chapter 3 introduces core concepts in probability and statistics. Many of the models and theory for time series forecasting are framed within a statistical framework, in particular when the variables' underlying uncertainty needs to be utilised, this is especially true when implementing probabilistic forecasts for probabilistic forecasting (Chap. 11). The final primer is Chap. 4 and describes the main definitions and concepts in machine learning, of which predictive analytics is a major subset. Many chapters in this book are on different areas of machine learning and build on this primer.

Chapter 5 focuses on the core definitions, descriptions and concepts of time series, and time series forecasting. From this chapter the reader should be able to understand the required components in order to frame a well-defined time series forecast problem. Note that while we motivate this book for load forecasting, the concepts presented are not necessarily limited to be applied in forecasting electric load. Most of the concepts introduced are applicable also in other similar time series forecasting tasks.

The next three chapters develop the tools and techniques required to develop these forecast models. This begins with Chap. 6 which shows how to analyse and understand the relationships within and related to the load data time series. This includes preparation techniques, such as identifying anomalous values, and feature engineering techniques which can help you understand what variables are the main drivers of the load. These relationships will determine the types of models the forecaster will eventually choose as well as the main input variables.

An accurate forecast model is not possible without having a proper evaluation method. This is the topic of Chap. 7 which introduces a whole host of error measures and skill scores for both point and probabilistic forecast models. It also introduces some simple checks which can be performed to help improve the accuracy of the model. The next chapter (Chap. 8) describes how to appropriately train and select forecast models. Here one of the most important concepts in machine learning is introduced, namely "bias-variance trade-off". This is vital to ensure models can generalise to new, unseen data and hence capture the true behavioural patterns in the load time series. The chapter considers a number of techniques, in particular regularisation, which helps enable this generalisation.

With the main tools and techniques introduced, the next three chapters describe a plethora of forecasting methodologies which have been developed for producing successful point and probabilistic load forecasts. Most of these models utilise the patterns found in the historical demand data since most LV demand has regular features in past behaviour. Chapters 9 and 10 focus on point forecasts using statistical and machine learning based models respectively. These separate categories have different advantages and disadvantages but are both useful to enable methods which have the full potential of descriptive power and high performance. Chapter 11 introduces a range of probabilistic forecasts. These are essential for many distribution level applications because of the relatively high uncertainty in the demand behaviour. In LV systems in particular, the demand is typically quite volatile, hence it is likely that probabilistic methods are going to be increasingly required for future smart grid applications. Since probabilistic forecasts can be defined in several ways, this chapter gives many different methods for doing this. By the end of the three main model chapters, the reader should have a solid understanding of the best methods in time series forecasting, and when to apply them. An overview of the models explored in this book are given in Fig. 1.3. The whole forecasting process, from data analysis to evaluation, as well as tips on how to choose the most appropriate model, are described in Chap. 12.

Time series forecasting is a vibrant area of development and research. Many of these techniques are likely to be applied more and more in real world applications. For

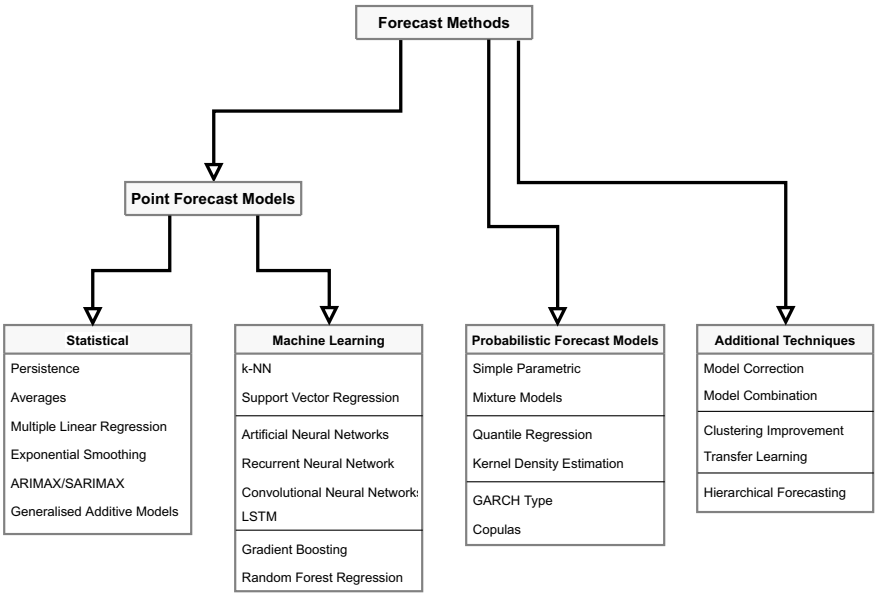


Fig. 1.3 Some of the main models explored in this book for both point and probabilistic forecasts. Also represented are additional techniques which are applied in LV load forecasting

this reason, some of the more advanced topics are described (briefly) in Chap. 13. This includes improving accuracy through combining models, and how to test whether two forecasts are statistically different in accuracy from one another.

Given the model, techniques and tools, the final two chapters of this book are devoted to applying these techniques in forecast experiments and real LV level applications. Chapter 14 gives a full forecasting case study applied to the demand data of 100 real-world low voltage level feeders. In this chapter, it is demonstrated how to analyse the demand data, develop several forecast models and test them. These forecasts illustrate some of the challenges and difficulties that come with real demand data. Chapter 15 then considers a number of applications where these forecasts can be applied, most prominently featured is a battery storage scheduling problem.

It is worth noting, that this book also serves as a relatively thorough introduction to data science in general, albeit focusing on a specific domain. It includes all the features and good practice in machine learning, or more generally artificial intelligence, such as cross-validation, model evaluation, benchmarking etc. as well as a whole host of different techniques including decision trees, neural networks, linear regression and generalised additive models. Hence by studying this book you can become highly knowledgeable in generating machine learning models, in particular for time-series forecasting based applications.

The primary use case of this book is as an introduction to the short term load forecasting for undergraduate or graduate students. However, this book can also

serve as an introduction for interested readers from industry, such as data scientists, statisticians or analysts working in utilities, system operators, or other companies in the energy industry. The main case study presented in this book comes from peer-reviewed papers published by the authors, and highlight the heavily applied focus of this topic, hence we hope this book can be a useful reference to those interested in applying the methods in real-world applications.

1.5 How to Read This Book

The best way to read this book mainly depends on the experience of the reader. If you are a statistician, data scientist or probabilist with experience in modelling, then the reader may want to focus on the applications and the case study and only refer back to the technical chapters when needed. In contrast, if the reader is an engineer or an expert in energy systems, but with minimal knowledge of data science then the focus should be on the earlier chapters, specifically the Chaps. 3–7. In either case, the reader should familiarise themselves with the notation and can skim through parts of these chapters based on their background. Figure 1.4 shows the links and dependencies between the chapters.

The models described in the three main Chaps. 9–11 focus on point and probabilistic forecast models. Those with a statistics background will feel most comfortable with Chap. 9 which looks at many traditional time series forecast models such as linear regression and ARIMA, but also has more recent popular techniques such as generalised additive models. These are highly interpretable models and can be useful when trying to evaluate the performance of the forecast.

Those with a computer science or more traditional machine learning background will better understand the techniques in Chap. 10 which considers many methods such as deep learning, random forest and support vector regression. These models are not so interpretable but are often high performing. The more complicated and computational intensive probabilistic forecasts are introduced in Chap. 11 and include a mix of both more statistical and machine learning models. The statistical primer (Chap. 3) and the probabilistic error measures section of Chap. 7 are essential for this topic.

There are a lot of methods in Chaps. 9–11. The reader of course could read the entire chapter and get a solid overview of the wide variety of methods available for load forecasting. However, the reader may also wish to focus on point forecasts (traditional or more modern machine learning) or probabilistic, or may wish to mix-and-match a few methods from each. For a semester course then the choice of methods would ideally be based on those which are utilised in the case study from Chap. 14. This ensures a full forecast experiment can be demonstrated. The next section below outlines one possible delivery of a single semester course.

Machine learning can only be grasped by doing. For this reason each chapter finishes with a few questions on the contents and in the appendix there is also a full walkthrough which can guide the reader through the entire process from

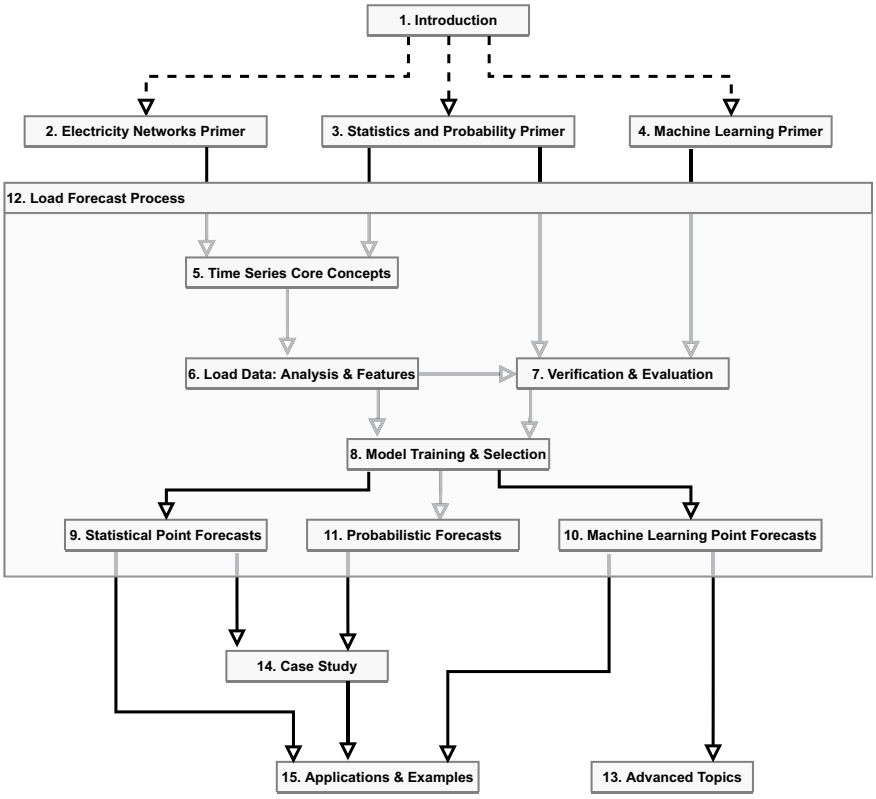


Fig. 1.4 Figure illustrates direct dependencies between the various chapters in the book. Note that Chap. 12 summarises the load forecasting process and therefore integrates most of the method and technical sections

data collection, data cleaning, model selection and evaluation. Also linked is a notebook (https://github.com/low-voltage-loadforecasting/book-case-study) which demonstrates some of the techniques applied within a python environment. This will hopefully illustrate how to apply the analysis and modelling described in this book and is outlined in the Case Study, Sect. 14.3.

1.6 Note for a Semester Delivery Course

There are a few prerequisites for learning the material presented in this book. Students should be familiar with basic calculus, and properties of basic statistic and probabilities. It is not recommended to cover the whole book in one semester course unless students already have knowledge of probability theory and the main concepts in machine learning.

This book introduces one full case study in Chap. 14 on point and probabilistic forecasts for low voltage residential feeders. For this reason, the most suitable focus for a semester course will be to introduce the main concepts and models which can demonstrate the case study. To gain hands on experience, students should develop and test their own models and for these reason lists of open data are given in the appendix. To assist with this, the authors have developed a python notebook which steps through some of the concepts and is available at https://github.com/low-voltage-loadforecasting/book-case-study. The context is described in Sect. 14.3 but the reader should try and develop their own models in addition to those illustrated in the notebook.

To cover the case study, a semester course will require covering the main concepts in data analysis, model selection and evaluation, but will only require a selection of models from Chaps. 9 and 11. In other words the semester will only focus on some statistical and probabilistic forecast models. The following sections would therefore make for a coherent introductory course:

Chapter 2: The main LV network context and motivation for the case study is presented in this chapter. If these concepts are not taught in other courses then a brief overview of this chapter should be included. At the very least the section on "features of distribution networks".

Chapter 3: This is a primer on essential statistical and probabilistic principles and it is recommended that it is presented in full if not covered in other modules.

Chapter 4: This has useful core concepts for machine learning and the sections on supervised learning and optimisation are useful if there is available time.

Chapter 5: This section introduces the main definitions and descriptions of time series forecast and hence is necessary to present in full. It also introduces the notation throughout the rest of the book.

Chapter 6: This chapter should be taught in full as it introduces the main data analysis techniques used in the case study. A reduced version of this chapter can be taught if there is other modules which teach time series data analysis.

Chapter 7: The point error measures should be taught, as should the CRPS from the probabilistic error measures. The residual checks section should also be included if time.

Chapter 8: Section 8.1 on general principles should be taught in full but only the sections on least squares and information criteria are required from Sect. 8.2.

Chapter 9: Only particular methods are required to be covered in this section, this includes the benchmarks (Sect. 9.1), exponential smoothing (Sect. 9.2), and multiple linear regression (Sect. 9.3).

Chapter 11: From this chapter teach Sect. 11.4 on quantile regression and the section on ensemble methods (Sect. 11.6).

Chapter 14: The case study should be worked through in full, but can be split into the different components and introduced with the relevant parts from each of the previous chapters.

Assessment There are some questions at the end of each chapter. These can be worked through to support the teaching of the material. There is also a guided walkthrough in the Appendix C, Instead of presenting the case study this could be worked through or set as a coursework challenge. A list of open data is also given in Appendix D.4 which students could use to develop and test their own methods.

Reference

1. Irish Social Science Data Archive. Commission for energy regulation (cer) smart metering project - electricity customer behaviour trial (2012)

Chapter 2
Primer on Distribution Electricity Networks

This chapter gives a brief overview of the electricity distribution network. This knowledge is important to understand some of the core features of the network and the corresponding data, what are the main of applications, and how to create an appropriate forecast model.

2.1 The Electricity Distribution Network and Core Concepts

In the traditional electricity network, electricity is generated at the transmission level via conventional fossil fuel generators such as coal, and gas, and also nuclear fission. This is then supplied to consumers by first transporting the electricity over long distances at high voltage via the **transmission network**, and then stepping the voltage down and transferring more locally via the **distribution network**.

The distribution network typically starts at the so-called **grid supply point** where power is transferred from the transmission to the distribution network, and then is stepped down through various substations until it reaches the consumer. Larger consumers will be connected at higher voltages whereas residential consumers will be connected at the lowest voltage. The objective for the transmission network operator (TSO) is to match supply and demand by either increasing or decreasing the generation supplied, or the demand consumed. The focus of the traditional network is very much on the generation side.

With the transition towards a low carbon economy, the electricity networks have become much less centralised and much more diverse. There are two main developments. Firstly there has been an increase in renewable energy generation including wind, solar and even tidal. Rather than generating energy at the highest level of the transmission network renewable sources often operate at lower voltage levels and more locally to where electricity is used. For example, for rooftop solar photovoltaics, the energy generated may be directly used by the occupants of the building.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_2

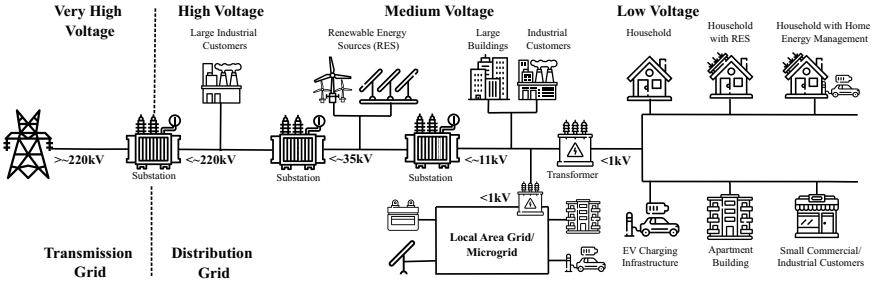


Fig. 2.1 Illustration of the electricity grid as well as the low voltage area, the focus of this book. Reprinted from [1] with permission from Elsevier

The second major development is the increase in **low carbon technologies** (LCTs) such as heat pumps and electric vehicles (EVs). These are promoted in order to decarbonise both heating and transport which has traditionally been fuelled by high carbon technologies such as gas, and petrol respectively. An illustration of a more modern electricity network is shown in Fig. 2.1.

The effect of the transition to a Net Zero energy system is to create a electricity grid which has

1. Increasing demand (due to the increased number of high demand LCTs).
2. Much more localised, weather-dependent energy generation.

These make the network much more complicated to operate and maintain security of energy supply. For one, more LCTs increase the stress on the networks which have not been designed to cope with many heat pumps or electric vehicles being connected. Secondly, the generation in a local area of a network may be much higher (or lower) than the local demand. Further to this, wind turbines do not generate energy when the wind isn't blowing, and solar panels do not generate when the sun isn't shining. This weather dependence makes renewable generation less reliable, being much more volatile and intermittent. Some ways to solve for this are to utilise storage devices (e.g. see Sect. 15.1 for an example with batteries) so that energy can be "moved" to times it is needed by storing the energy when generation is high but demand is low and then releasing the energy when generation is low and demand is high.

The lowest voltage level of the network is the step down from the **secondary substations** to the final consumers. From the secondary substation, the demand will be split into individual **feeders** (up to around six) which usually follow the roads and streets and then *feed* electricity to the individual consumers. These feeders can connect to a range of different consumers, from a single larger consumer (say a supermarket) or to about a hundred smaller consumers (usually residential), and every combination inbetween. In addition, feeders also supply electricity to other street furniture such as lighting, traffic cameras, elevators etc.

Although this book will focus on the distribution networks in general, the most challenging area is the low voltage network. This can be roughly defined as the

area from the **Primary substation** (about 11 kV) down to the final household (see Fig. 2.1). This has many more unique challenges as will be described in the next section.

2.2 Low Voltage Networks

The low number and variety of connections to the low voltage (LV) network means there are two main challenges which are intensified compared to the higher voltage or national levels of the network.

The first problem is that demand is relatively volatile. Since on the lowest level of the network there is only on average around 40 or 50 consumers (but can be as few as a single larger consumer), the demand is much spikier and less regular than the aggregation of 100s or 1000s of consumers as is true at the next step up in the distribution network. Thus it is much more difficult to predict or model the demand. This makes things complicated for **distribution network operators** (DNOs) who are in charge of the cables and are required to maintain a supply of electricity to consumers. It is much harder to optimally plan and manage the network when the demand is less predictable with varying degrees of uncertainty. Further to this the demand is likely to change much more as increasing numbers of households and businesses install EV charging, PV solar and heat pumps.

The second problem is that low voltage networks are much more sensitive to individual demands. For example, a few LCTs (say heat pumps) will have a much bigger relative impact on the LV network than at the higher voltage levels. It only takes a few large devices to completely change the demand patterns. Many current networks have been designed without LCTs in mind which means they do not have the head room necessary to allow excessive numbers of high demand appliances or renewable generation sources. For the LV network to operate properly and protect against damage, the network should operate within particular constraints. The cables are built to be able to take a certain size of demand, and if this is exceeded it can break the network and cause outages. There are several ways that the network can be broken and we briefly discuss them below.

The first is **thermal constraints**. Usually the demand can exceed the specified headroom for a short period (an hour or two) but if the demand is higher than the thermal capacity of the cables for too long then overheating will occur and the network may be damaged. The demand should be lowered by either reducing the demand, e.g. through demand side response, or by utilising battery storage devices. The chance of exceeding the thermal constraints has been increasing due to the increase in low carbon technologies like electric vehicles which often have high power ratings (7 kW even for the slower chargers) and will also be utilised at similar times (when people come home from work they may all plug in their EVs).

The second potential problem is **voltage constraints**. The voltage must operate within particular limits. In the UK, the last mile of the network is a nominal 230 V and should be no less than 216.2 V (i.e. -10%) and no more than 253.0 ($+6\%$). This can

be different depending on the country of course. Voltage decreases when there is more demand on the network, and increases if there is generation on the network. As power flows down a cable to consumers and more demand is applied the voltage drops. If the demand is too high, then the voltage may drop outside the lower constraint. If there is excess generation on the network (from solar photovoltaic generation from numerous consumers) then the voltage may increase beyond the upper constraint. In both cases this can cause network failure and damage to electrical components.

The final main problem is **phase imbalance**. The electricity in a feeder is often split into three **phases** through individual wires and the current and voltage are 120 degrees out of phase with each other. The details here are not required for this book, but the importance is that demand should be roughly equal across the three phases. If not then this can generate power losses, reduce the lifetime of appliances, and increase the heat within the cables, causing damage and possible failures. Connections to a feeder in the last mile of the LV network will split across consumers but it is unlikely to be evenly distributed. One phase may have many more consumers than another, hence LV networks are likely to be particularly unbalanced. The uptake of EVs and heat pumps will likely lead to further imbalances.

2.3 Some Features of Distribution Networks

Compared to system or national level demand, there is much less known about low voltage demand since it has not been monitored or analysed as extensively. One example, which will be demonstrated in the case study in Chap. 14, is the effect of temperature. This is generally considered a strong driver of national demand in the UK because a lower temperatures should mean more electrical heating appliances and hence greater demand. In hotter countries with air conditioning there may also be an increased demand when the temperature is higher. However, this may not necessarily be the case at the low voltage. Much of the heating in the UK at the time of writing (although this should change as the country moves towards lower carbon alternatives) is fuelled by gas. Thus if a network has only a few consumers which use electricity, there may be a small, or zero relationships with the temperature values. In any case, weather is a potential driver of demand on an LV network, and should be considered as a potential input for any forecast model.

Since electricity networks are radial it means that they are arranged in trees. Electricity is stepped down from higher voltages down to lower voltages and therefore energy typically flows in one direction. This suggests the following question: Is the demand at a substation simply an aggregation of the connected loads down stream? The answer is no, and for the following main reasons:

1. **Losses**: There are losses of energy as it travels through the cables. In other words, not all electricity makes it to its final destination. The total energy of all loads on a substation will therefore be lower than the energy recorded at the substation

itself which needs to supply more energy to account for that which is lost. These are typically small, around 5% in some instances.
2. **Switching**: Often electricity has to be rerouted to entirely different nearby networks. This could, for example, be because there was a shortage in the other network and therefore another substation has to temporarily supply the electricity (through some linking box).
3. **Unknown connections/demands**: In practice it is unlikely that all the downstream connections on a substation are known, and even when the roll-out of smart meters is complete, it is unlikely that all households will have half hourly electricity readings.[1] So in many cases the substation load cannot be fully estimated using the known downstream loads.

To add to the above complications, it cannot be assumed that the substation electricity flows in only one direction. With the increasing numbers of distributed generation, electricity flows are now reversing direction in some networks, something they were not originally designed to do. These complications must be taken into account when developing forecasts at low voltage level. If the effect of each of the above is significant then they must be integrated into the modelling. Switching is one of the more difficult to deal with as it requires taking into account a temporary shift in the demand behaviour which will then shift back at a later date. For this reason adaptive methods which quickly learn the new demand behaviour may be preferable and regime switching models may also be useful (Sect. 13.6.3). To deal with the misalignment in load between the substation and aggregated downstream loads (either through unknown connections or losses) the difference itself could also be included in the modelling, since this will either be a scaling (losses) and/or be itself an aggregate of the few unmonitored consumers.

So what does distribution, or even individual consumer load look like? For simplicity, complications such as losses and street furniture (they will be relatively small) are ignored, and it is assumed there is no switching behaviour. Examples of residential demand are shown in Fig. 2.2. These are very diverse and no two are the same, although there are some similar features. For example, since occupants are often at work during the day and more active in the morning and evening, the corresponding demand usually has peaks at similar times. Further, most households have weekly and daily seasonality, with Saturdays and Sundays having different patterns than typical weekdays. This is not obviously true for households which may be occupied by shift workers etc. There are also some technologies which produce particularly strong demand features, such as electric vehicles and overnight storage heaters which can create large overnight demand.

The difference in household demand is worth highlighting further. Even when the homes are very similar (e.g. 1920s Semi-detached), and the occupants have similar socio-demographics, their residential demand may be very heterogeneous, with different regularities and distributions of daily demand. Figure 2.3 shows the half-hourly demand of four randomly selected households on a Monday. The top

[1] For example, some households will not have a suitable location for installing a smart meter, or there may not be sufficiently available signal to transmit the information.

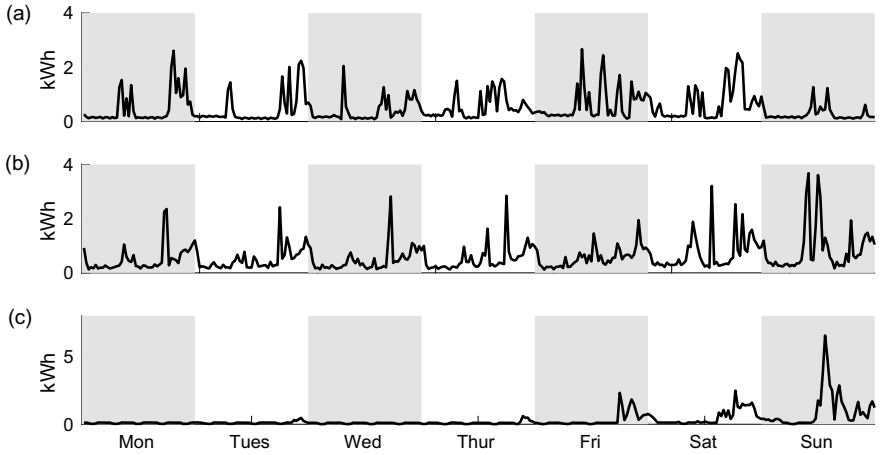


Fig. 2.2 Three examples of residential smart meter demand over a week at half hourly resolution. Constructed using data from the CER Smart Metering Project—Electricity Customer Behaviour Trial, 2009–2010 [2]

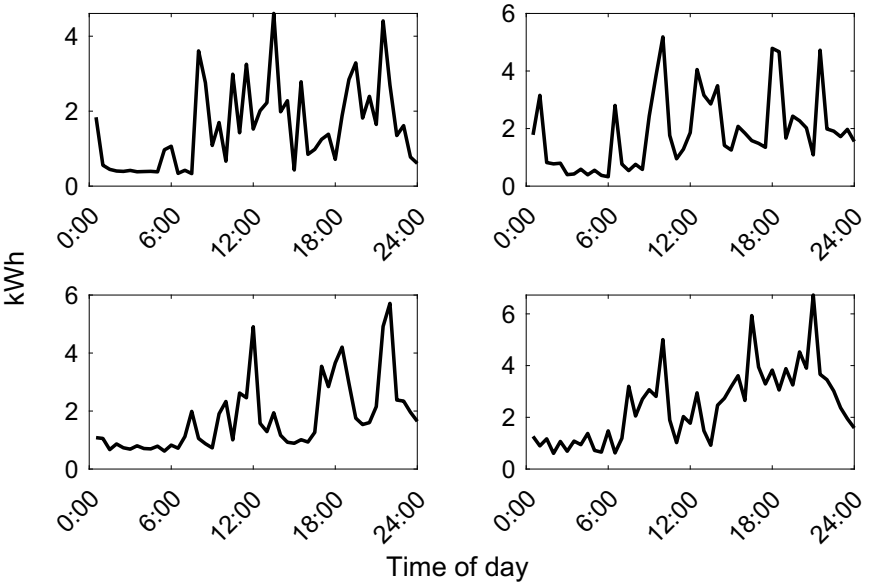


Fig. 2.3 Examples of half hourly demand profiles over a single Monday for aggregations of five households. Constructed using data from the CER Smart Metering Project—Electricity Customer Behaviour Trial, 2009–2010 [2]

two in this case have higher demands around midnight than the bottom two, whereas the bottom two have much more distinct morning and evening peaks. This suggests the bottom two have more regular "9 to 5" jobs outside the home, with peaks in the morning due to switching on say kettles or electric showers, and then evening peaks due to returning home, and perhaps cooking or switching on the TV etc. In contrast, the top two profiles seem to suggest a least some of the occupants within the house most hours of the day as there is peaks throughout the day. This could be from someone performing their job from home, or could be other household chores etc. Identifying what appliances may be in operation by analysing the household profile is another entirely separate branch of energy analytics (outside of the scope of this book!) called **Non-Intrusive Load Monitoring** or NILM.

Small to medium enterprises (SMEs), such as hairdressers, churches, schools, shops etc. are also very different even within the same categories (pubs for example) although they will be more similar than across categories. The demand magnitude and its distribution is often based on the operational hours and the type of appliances that are used within the SME. For example, offices will be determined by computing equipment, heating and lighting, usually during the day. In contrast a pub will be based on heating and lighting, but also hand dryers, pumps, cooking appliances, and refrigeration. The demand will also be mainly within the evening. Some real profiles of SME demand are shown in Fig. 2.4. The clear daily regularity of the first two SMEs is very apparent as is the fact that they are both closed on Sunday (in fact the first SME is also closed on Saturday). The other consumer has much more volatile behaviour and there is some demand on all days of the week.

Since distribution network demand is mostly made up of the aggregation of different residential and commercial consumers their demand distribution will be as diverse as the possible combinations of consumers. However, distribution, and especially

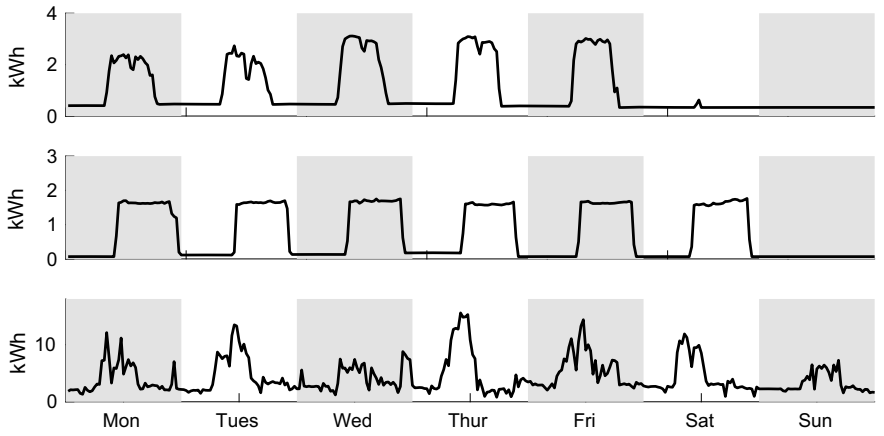


Fig. 2.4 Three examples of smart meter demand for SMEs over a week at half hourly resolution. Constructed using data from the CER Smart Metering Project—Electricity Customer Behaviour Trial, 2009–2010 [2]

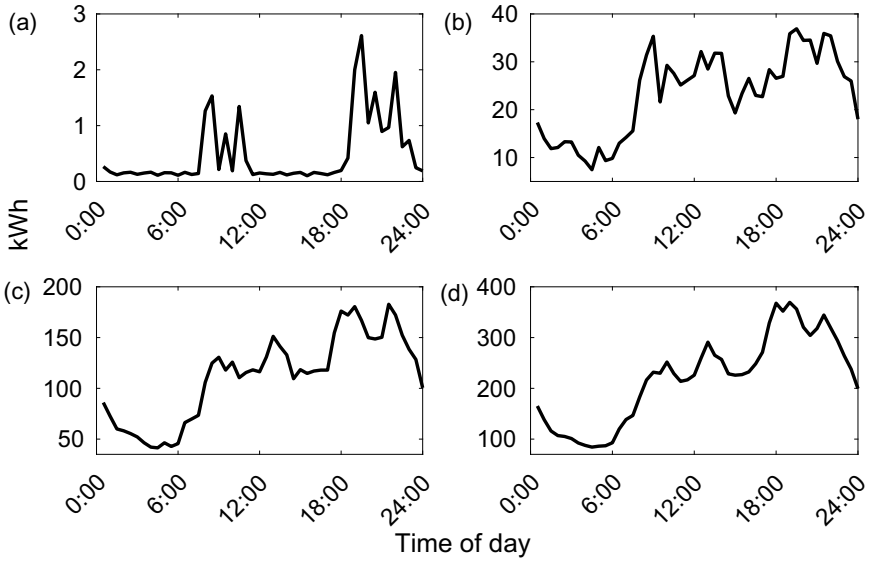


Fig. 2.5 Examples of half hourly demand profiles over a single Monday for **a** a single household, **b** the aggregation of 50 households, **c** the aggregation of 250 households, and **d** the aggregation of 500 households. Constructed using data from the CER Smart Metering Project—Electricity Customer Behaviour Trial, 2009–2010 [2]

LV, networks are very diverse in terms of the numbers and mixture of consumers. Figure 2.5 shows the aggregation of different numbers of individual (residential) smart meters for a single Monday (Another example for a full week is shown in Sect. 1.2, Fig. 1.2). LV feeders with larger numbers of consumers (all residential in this case) have much less relative volatility and are more regular. Further to this, it must be remembered that real LV feeders are not connected to purely residential consumers, but also supply energy to different street furniture and can also include commercial consumers. A single commercial consumer can change the dynamics of a network significantly since they often have larger demand and they may use energy at very different times: e.g. a hairdresser will have higher demand during the day, but households typically have evening peaks. The data analysis for the case study in Sect. 14.2.2 includes some examples of real feeders (Fig. 14.1) and shows that even when they have similar numbers of residential connections they can produce very different demand over the year and on special days such as Christmas. If the feeders are purely residential then their demand profiles look like smoothed versions of individual residential demands (see Fig. 2.5). The effect of a single commercial connection is demonstrated in Sect. 14.2.2, Fig. 14.3.

2.4 Managing the Distribution Network

So what tools and solutions are required for the distribution networks to ensure they work properly and consumers can be supplied with the energy they need?

As discussed in Sect. 2.2 some of the major issues to be managed are phase imbalances, voltage constraint violations and thermal constraint violations. These were relatively well managed in the last few decades and the worse case scenario usually required digging up the road to install larger cables to handle the increased demand. With increased uptake of LCTs and more distributed renewables, new solutions may be required since the traditional network reinforcement upgrades will be quite expensive and disruptive.

Instead, what is envisioned is the move towards a **distribution system operator** (DSO) who will control the supply and demand on the distribution network much like the electricity system operator matches the supply and demand at the national level. The DSO would do this by procuring various flexibility services from across the network, for example this could be demand management where demand is controlled via a storage device (Sect. 15.1), or consumers can be required to change the demand of particular appliances by responding to signals from system operators via so-called *demand side response* (Sect. 15.2).

There are also ways to persuade consumers to change demand by offering new incentives such as smarter tariffs to consumers. Instead of the current practice of a simple flat unit-rate for use of electricity, the charges can change over the day to identify periods of high or low use, or can be dynamic to respond to the current conditions of the network. They can also be tied to renewable generation to try and utilise cleaner energy.

With these consumer focused initiatives, local communities may also be more involved in the managing of the network. The aggregation of hundreds or thousands of controllable assets such as EVs, heat pumps and battery devices can produce significant aggregated effects on the network and save consumers money. For example, EV charging could be co-ordinated to ensure there isn't a large charging peak when all consumers plug in their devices. Alternatively, the collective battery power of hundreds of EVs could be used to reduce network peaks and utilise more local solar PV generation. This integration of thousands to millions of controllable devices with the ability for two-way communication is often referred to as the **smart grid** and could lead to a much more decentralised energy system where power is generated and utilised locally via the smart control of the assets.

There are also emerging energy markets which are offering participants ways to make savings through their own devices to offer additional capacity, balancing services, or other ancillary services, such as frequency control. The interesting aspect of many of these applications is that for them to optimally operate their behaviours must be accurately anticipated. This book therefore, and many of the models presented, can be used to help, and indeed are necessary, to support the future energy network. A selection of applications will be presented in Chap. 15.

2.5 Questions

1. Think about your electricity usage throughout the day. What would your usage profile look like? How would it change on a weekend versus the weekday?
2. Which of your electricity usage behaviours are the most flexible? I.e. which ones could be easily moved to a different time of day? Which ones could move the most? Which ones would it be difficult to move?
3. Download a set of smart meter datasets (some are listed in Appendix D.4). Plot (using any of your favourite plotting software) a few weekly profiles from some households. Try and think about what appliances may contribute to the major demands you see. Plot the demand over a year. What is the shape? Is there any annual seasonalities or does it stay consistent over the year? What is the largest demands you see? Can you guess what they may be? Is there any unusual or anomalous values you see? Very large ones, or periods of missing data?
4. If you have an EV what would the typical charging profile look like? If you don't have an EV think about where you would charge it, would it be at home or at the workplace? Think about what some other typical charging profiles would look like. Do they have a lot of diverse load or are they going to generate a large peak from charging at the same time?
5. Reflecting on how you use heating within your home. What would the energy profile look like over a period in winter? What about the average daily demand over a year?

References

1. S. Haben, S. Arora, G. Giasemidis, M. Voss, D.V. Greetham, Review of low voltage load forecasting: methods, applications, and recommendations. Appl. Energy **304**, 117798 (2021)
2. Irish Social Science Data Archive. Commission for energy regulation (CER) smart metering project - electricity customer behaviour trial (2012)

Chapter 3
Primer on Statistics and Probability

Throughout this book, various concepts from statistics and probability will be used which are essential for understanding and constructing forecasts. In this section some of the fundamental tools and definitions will be presented. It is assumed that the reader knows some basic probability and statistical concepts, and this chapter is only intended as a refresher of the main ideas which will be used in other parts of the book. This chapter will go over basic definitions of distributions, methods for estimating them as well as introduce some important concepts such as autocorrelation and cross-correlation. For a more detailed description of basic statistics the authors recommend an introductory text such as [1] (In addition see the further reading material listed in Appendix D).

3.1 Univariate Distributions

Real world data typically has some degree of uncertainty with the values it takes **distributed** over some (potentially infinite) range of points. A major part of probabilistic forecasts is trying to accurately describe, or model, the distribution of the values of interest. For the purposes of this book, distributions will be used to describe probabilistic forecasts and hence understand the uncertainty associated to the estimates they produce. Note that the focus will be on demand data and hence the methods will typically apply to real continuous variables (as opposed to discrete/categorical variables). In this section only univariate distributions will be considered, i.e., those which describe a single real variable. We define a continuous variable, whose values depend on a random process and has a continuous distribution, as a **random variable**. Notice in the following, the random variable will be denoted with a capital letter, e.g. X, whereas lower case variables, e.g. x, will refer to particular realisations/observation of that random variable.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_3

One of the most common ways to describe the distribution of a random variable X is through its **probability density function** or PDF, $f_X(x)$, over some (possibly infinite) interval $x \in (a, b) \subseteq \mathbb{R}$. The PDF is a non-negative function which describes the relative likelihood that any value $x \in (a, b)$ will be observed and can be used to calculate the probability of the variable taking some value within an interval $(a_1, b_1) \subseteq (a, b)$ as follows

$$P\{a_1 \leq X \leq b_1\} = \int_{a_1}^{b_1} f_X(x)dx. \tag{3.1}$$

Note that the integral of the PDF is bounded above by one by definition.

An alternative but equally important representation of the distribution is the **cumulative distribution function** or CDF. Again assume the CDF is defined over some (possibly infinite) interval $x \in (a, b) \subseteq \mathbb{R}$. The CDF represents the probability that the random variable X will take a value less than or equal to some specified value x, and is often written as a function $F_X(x) = P\{X \leq x\}$. It is related to the PDF via

$$F_X(x) = \int_a^x f_X(t)dt. \tag{3.2}$$

I.e. an integration of $f_X(t)$ for t over the interval (a, x). Notice that the CDF is a monotonically increasing function, which means that if $x_1 \leq x_2$ then $F_X(x_1) \leq F_X(x_2)$. Also it satisfes the limits, $\lim_{x \longrightarrow a} F_X(x) = 0$ and $\lim_{x \longrightarrow b} F_X(x) = 1$.

The expected value and variance are two important derived values associated to a PDF/CDF. The expected value is defined as

$$\mathbb{E}(X) = \int_a^b t f_X(t)dt, \tag{3.3}$$

and the variance as

$$\mathbb{V}ar(X) = \int_a^b (t - \mu)^2 f_X(t)dt, \tag{3.4}$$

where $\mu = \mathbb{E}(X)$. The expectation (or mean) essentially represents a weighted average of the values of the random variable X with values weighted by the probability density f_X. It acts as a typical, or expected, value of the random variable.

The variance is the expected value for the squared deviation from the mean. It is often used to represent the spread of the data from the mean and hence is a simple measure for the uncertainty of the random variable. The square root of the variance is known as the **standard deviation** (often denoted $\sigma = \sqrt{\mathbb{V}ar(X)}$).

One of the most commonly studied, and important, distributions is the one dimensional **Gaussian** (also known as the **Normal**) distribution which has a PDF defined by

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu)^2}{\sigma^2}\right). \tag{3.5}$$

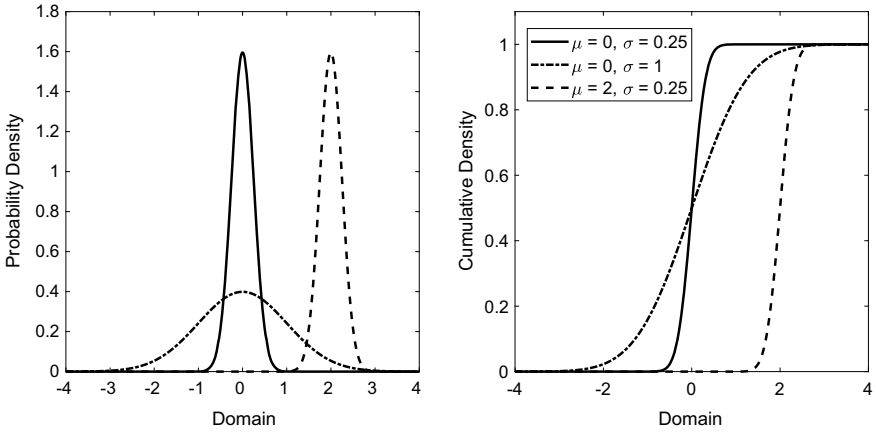


Fig. 3.1 Examples of the PDF (left) and the CDF (right) for the Gaussian distributions with different means and standard deviations

Thus the Gaussian is defined entirely by two parameters, namely the mean μ and the variance σ^2. When $\mu = 0$ and $\sigma = 1$ then (3.5) is known as the **standard normal distribution**.

An example of the Gaussian distribution for various means and standard deviations is shown in Fig. 3.1. Notice that the Gaussian distribution is always bell-shaped and is symmetric around the mean value. Also the bigger the variance the wider the distribution, as expected. In the plot for the corresponding cumulative distribution, smaller variances translate to steeper gradients.

Not all variables are Gaussian distributed, or even symmetric. There are a whole host of other parametric families of distributions. The **lognormal distribution** is suitable for variables which are positively skewed distributions with long tails to the right and has PDF defined by

$$f(x) = \frac{1}{x\sqrt{2\pi}\sigma} \exp\left(\frac{-(\ln(x) - \mu)^2}{\sigma^2}\right), \tag{3.6}$$

for parameters μ and σ. Notice that this is simply a Gaussian distribution (3.5) but for the logarithm of the variable. There are also more general distributions such as the **gamma distributions** which can represent a whole range of different distribution shapes. The gamma CDF has the relatively complicated form

$$Gamma(X, \alpha, \beta) = \frac{1}{\beta^\alpha \Gamma(\alpha)} \int_0^X t^{\alpha-1} e^{-t/\beta} dt, \tag{3.7}$$

for parameters α, β often called the shape and scale parameters respectively and $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$ is the so-called gamma function. The PDFs for the lognormal and gamma distribution for various values of their parameters are shown in Fig. 3.2.

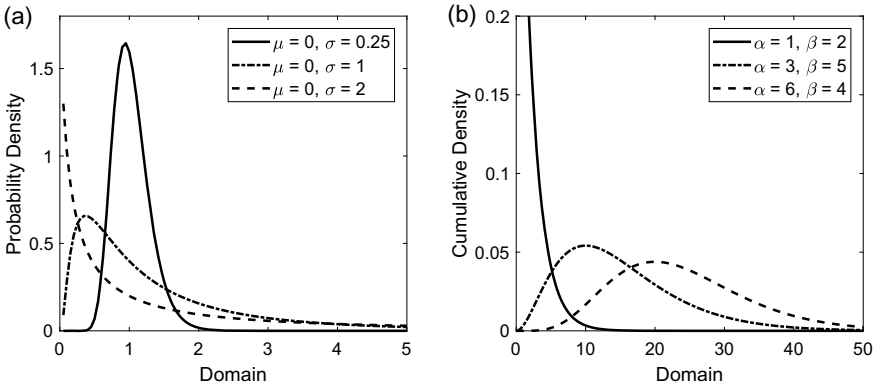


Fig. 3.2 Examples of PDFs for **a** the lognormal and **b** the gamma distirbution for various values of their parameters

The Gaussian, gamma and lognormal distribution are examples of **parametric** distribution functions because they are defined completely in terms of their input parameters. There are also **nonparametric** distributions which do not assume that the data come from any specific parametric family of functions. Kernel density estimation is a popular method for non-parametrically estimating a distribution and will be described in Sect. 3.4. In this book, the main focus will be on nonparametric models, but the Gaussian distribution will also be commonly used especially when modelling the distribution of errors.

How is this translated to short term load forecasts?

Point forecasts are most often estimates of the expected values of a distribution and hence are a central concept within the context of this book. The standard deviation of the forecast errors can also produce an indication of the uncertainty in the prediction and the spread in the potential values. Methods for producing point forecasts will be described in detail in Chaps. 9 and 10.

3.2 Quantiles and Percentiles

The continuous CDF admits an inverse function F_X^{-1} which can take any $q \in [0, 1]$ and give a unique value $x_q = F_X^{-1}(q)$. This value is the qth **quantile**, or q-quantile, also known as the $100q$ percentile [2]. The most well known values are the median which is the 0.5-quantile or 50 percentile and the lower and upper quartile which are the 25th and 75th percentiles, or the 0.25- and 0.75-quantiles respectively. Essentially the q-quantile defines the value in the domain, less than which a q proportion of the

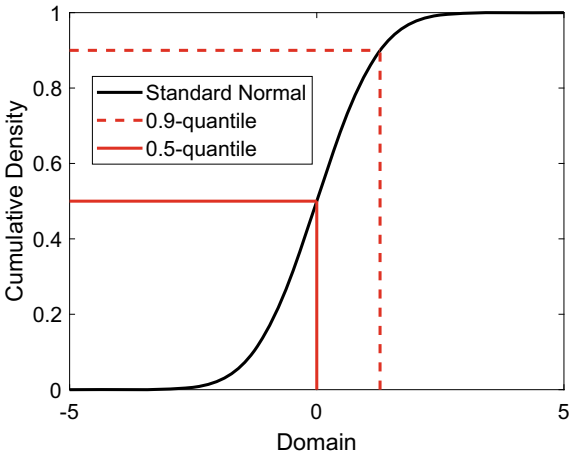


Fig. 3.3 Illustration of the 0.5- and 0.9 quantiles on the CDF for the standard normal distribution.

data lies. In other words, the proportion of the random variables X which are less than x_q is q.[1]

Example of the 50 and 90 percentiles for the standard Normal distribution are shown in Fig. 3.3. Notice that q-quantile is simply the domain value corresponding to where the horizontal line at $y = q$ intersects the CDF, as illustrated in Fig. 3.3. Often the complete CDF is unknown but a finite number of quantiles can be estimated. When enough quantiles are calculated an accurate estimate can be formed of the overall distribution. A technique for estimating the quantiles from observations is given in Sect. 3.4.

Quantiles are important tools for estimating distributions when only samples of the overall population are available and can be used to create probabilistic forecasts as will be demonstrated in Sect. 11.4.

3.3 Multivariate Distributions

Multivariate distributions are an extension of the univariate distributions introduced in Sect. 3.1 to distributions of more than one variable. This time consider N random variables $X_1, X_2, \ldots, X_N$. Analogous to the PDF for the univariate distribution, is the **joint** probability distribution

$$f_{X_1, X_2, \ldots, X_N}(x_1, x_2, \ldots, x_N) \tag{3.8}$$

[1] Note that some authors refer to the n-quantiles where the quantiles are defined in terms of dividing the domain into $n \in \mathbb{N}$ sets. Hence the 2-quantile would be the median, the 4-quantile would consist of the median, the lower quartile and the upper quartile etc.

which, like the PDF describes the relative probability that the set of values $(X_1, X_2, \ldots, X_N)$ will be observed. The CDF for a multivariate distribution is defined as

$$F_{X_1,\ldots,X_N}(x_1, \ldots, x_N) = P\{X_1 \leq x_1, \ldots, X_N \leq x_N\}, \tag{3.9}$$

and can be written in terms of the joint probability distribution

$$F_{X_1,\ldots,X_N}(x_1, \ldots, x_N) = \int_{-\infty}^{x_N} \ldots \int_{-\infty}^{x_1} f_{X_1,X_2,\ldots,X_N}(t_1, t_2, \ldots, t_N) dt_1 dt_2 \ldots dt_N. \tag{3.10}$$

If $N = 2$ then the multivariate distribution is known as a bivariate distribution. One of the simplest multivariate distributions is the multivariate Gaussian joint distribution defined by

$$f_{X_1,\ldots,X_N}(x_1, \ldots, x_N) = \frac{1}{(2\pi)^{N/2} det(\boldsymbol{\Sigma})^{1/2}} exp\left((\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right), \tag{3.11}$$

were $\mathbf{x} = (x_1, \ldots, x_N)^T \in \mathbb{R}^N$, $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_N)^T \in \mathbb{R}^N$ where μ_k is the expected value for the random variable X_k and $\boldsymbol{\Sigma} \in \mathbb{R}^{N \times N}$ is the covariance matrix and describes the covariance between the variables. The diagonal elements of this matrix describe the variance of the random variables, i.e. $(\boldsymbol{\Sigma})_{k,k} = \mathbb{V}ar(X_k)$, and the off diagonal elements describe the variation of one random variable in relation to another random variable. Consider two random variables X_k and X_m, then the covariance between these two variables can be written in terms of the expectation as

$$Cov(X_k, X_m) = \mathbb{E}[(X_k - \mathbb{E}[X_k])(X_m - \mathbb{E}[X_m])]. \tag{3.12}$$

Notice that the covariance matrix is symmetric and positive semi-definite.[2] The **(Pearson) correlation** is defined to be

$$Corr(X_k, X_m) = Cov(X_k, X_m)/\sigma_k \sigma_m, \tag{3.13}$$

and is bounded by $[-1, 1]$ and is a measure of the linear dependence between two variables. In other words, the correlation between two variables is the covariance scaled by the standard deviation of the variables. If two variables are independent (i.e. the change in one variable doesn't effect the change in the other) then they are uncorrelated and their covariance is equal to zero. For the special case of two variables, the bivariate covariance matrix can be written as

$$Cov = \begin{bmatrix} \sigma_1^2 & \sigma_1 \sigma_2 \rho \\ \sigma_1 \sigma_2 \rho & \sigma_2^2 \end{bmatrix}$$

[2] A symmetric real-valued matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ is said to be positive-definite if $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$ for all non-zero $\mathbf{x} \in \mathbb{R}^N$.

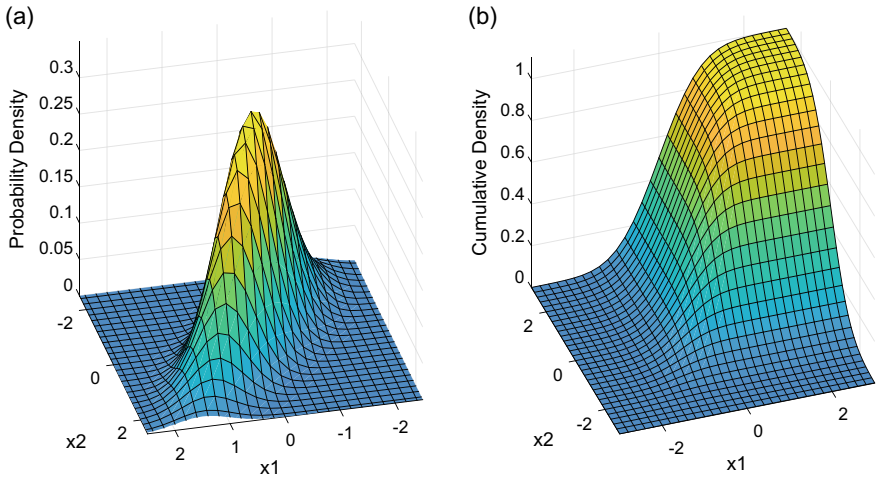


Fig. 3.4 Examples of a bivariate normal PDF (**a**) and CDF (**b**)

where $\rho \in [-1, 1]$ is the correlation $Corr(X_1, X_2)$ and σ_1, σ_2 are the standard deviation of the random variables X_1 and X_2. An example of a bivariate Gaussian distribution is shown in Fig. 3.4 for $\rho = 0.6$, $\sigma_1^2 = 0.6$ and $\sigma_2^2 = 1$. Here (a) is the joint density and (b) is the joint CDF. The correlation ρ is relatively large and hence the variables are somewhat correlated with each other.

To simplify the discussion on multivariate distributions, the focus of the rest of this chapter will be on bivariate distributions, but the results extrapolate to more general multivariate distributions.

One of the most important sub-structures of a multivariate distribution is the **marginal distribution**. Given a joint bivariate distribution, $f_{X_1,X_2}(x_1, x_2)$, the marginal distribution of X_1 describes the distribution of X_1 given no knowledge of X_2 and is found by integrating over X_2

$$f_{X_1}(x_1) = \int_{-\infty}^{\infty} f_{X_1,X_2}(x_1, x_2) dx_2. \tag{3.14}$$

Similarly the marginal distribution of X_2 can be defined

$$f_{X_2}(x_2) = \int_{-\infty}^{\infty} f_{X_1,X_2}(x_1, x_2) dx_1. \tag{3.15}$$

The joint and marginal distributions for a bivariate Gaussian (the same joint as given in Fig. 3.4) are illustrated in Fig. 3.5. Notice that if the random variables X_1 and X_2 are independent then the joint density is simply a product of the marginals for

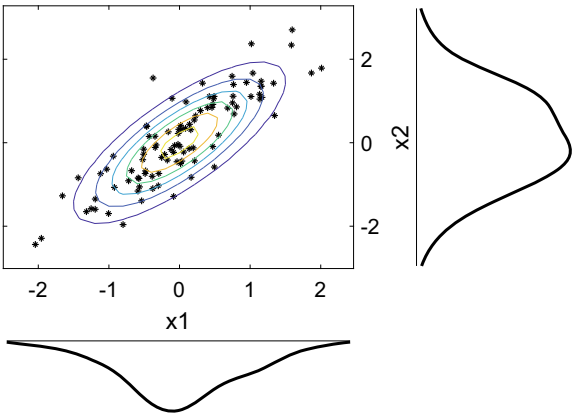


Fig. 3.5 Illustration of the joint and marginal of a bivariate Gaussian distribution. The contours show equal values from the joint distribution. Samples from the joint distribution are shown as the scatter plot whereas estimates of the marginal distributions for each variable are drawn on the corresponding axes

each variable $f_{X_1,X_2}(x_1, x_2) = f_{X_1}(x_1) f_{X_2}(x_2)$. The marginals can often be easier to estimate since they only require estimating each individual variable rather than needing to model any interdependencies between them.

If one of the values, say X_2 is observed so its value x_2 is known for certain, then the distribution of X_1 given this particular value is known as the **Conditional density** and is written $f_{X_1|X_2}(x_1|x_2)$.

The joint, marginal and conditional distributions are related by the following formula

$$f_{X_1|X_2}(x_1|x_2) = \frac{f_{X_1,X_2}(x_1, x_2)}{f_{X_2}(x_2)}. \tag{3.16}$$

As a simple illustrative example, consider randomly sampling from a bag containing ten identical-looking balls, each with a unique number, one to ten, written on them. Since each ball has an equal chance of being sampled then the probability of drawing any of the balls is the same, $1/10$. However the *conditional* probability of drawing a $three$ given that we know that the ball has an odd value written on it is $1/5$.

How is this translated to short term load forecasts?

The joint, marginal and conditional densities are all useful when considering probabilistic forecasts. In this book, the focus will be on day ahead, often hourly, demand forecasts. This means that the aim is to estimate a 24-dimensional joint distribution $f_{X_1,X_2,\ldots,X_{24}}(x_1, x_2, \ldots, x_{24})$, where X_k is the demand for the kth hour of the following day. It is often much easier to estimate the univariate distribution of each X_k. These are *marginal distributions* of the full *joint distribution*. Now consider a day ahead forecast with hourly updates. I.e. the probabilistic forecast is updated as new observations are made. In other words, after the kth hour of the day the aim is to estimate the *conditional density* $f_{X_{k+1},\ldots X_{24}|X_1,\ldots,X_k}(x_{k+1}, \ldots x_{24}|x_1, \ldots, x_k)$, where $0 \leq k < 23$. Methods for producing probabilistic forecasts will be described in detail in Chap. 11. In particular this chapter introduces Copulas, a very popular method for producing joint density forecasts, and heavily depends on combining marginal density estimates with a copula function that models the interdependencies between the random variables (Sect. 11.7).

3.4 Nonparametric Distribution Estimates

In this section basic nonparametric methods for estimating and understanding the distributions from available observations are introduced. Some of these, such as kernel density estimation methods, will be used as building blocks for some of the probabilistic forecasts in Chap. 11. These type of methods are useful when the data is known not to come from a specific parametric family of distributions, e.g. Gaussian, or Gamma (see Sect. 3.1).

One of the most common methods for estimating the PDF is through a **histogram**. A histogram is simply a count of the number of observations within some predefined discrete partitions (called *bins*) of a variable. Bins are defined by dividing the space into discrete groups. For a univariate random variables these are just intervals $[a, b]$, for multivariate data these are regions defined by intervals, e.g. $[a_1, b_1] \times [a_2, b_2], \times \cdots \times [a_N, b_N]$. Often the histograms are restricted to univariate and bivariate data due to the difficulty of visualising higher dimensions. Each bin is usually of equal size but this is not necessarily required. An example of a histogram using 20 equally spaced bins is shown in Fig. 3.6a.

A limitation of the histogram approach is the dependence on the position and size of the bins and small adjustments to them can change the shape of the plot significantly. Further, the histogram is a discrete representation of a continuous variable which means some information is lost when binning. A preferable, but slightly more complicated estimator is **kernel density estimation** (KDE). KDEs are summations of small smooth functions K, called *kernels*, which are defined around each obser-

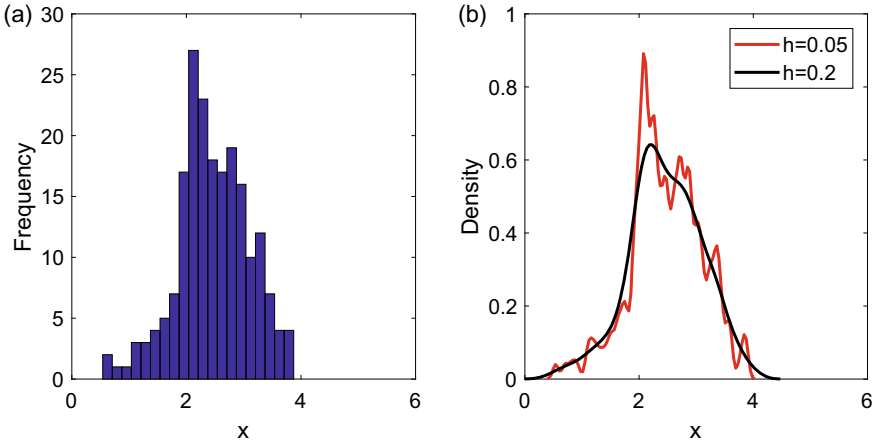


Fig. 3.6 Examples of estimating a distribution from 200 observations using **a** a histogram with 20 equally spaced bins and **b** a kernel density estimate with two different bandwidths

vation x_k of the random variable X to contribute to the overall PDF and are can be written as:

$$g(X) = \frac{1}{Nh} \sum_{k=1}^{N} K\left(\frac{X - x_k}{h}\right), \tag{3.17}$$

where h is the bandwidth hyperparameter. There are a variety of kernels but one of the most common is the Gaussian kernel defined as

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x\right). \tag{3.18}$$

The most important parameter is the bandwidth h which determines the smoothness of the final distribution. The larger the h the smoother the final estimate. The optimal value of this parameter is often found through cross-validation methods (see Sect. 8.2) although there are sometimes rules of thumb used when there is assumptions about the underlying shape of the distribution. A representation of a KDE for two different bandwidths is shown in Fig. 3.6b (for the same data as in Fig. 3.6a). Notice that if the bandwidth is too small then the KDE will overfit to individual observations (see Sect. 8.1.3). In contrast, a bandwidth which is too large will mean features are lost due to underfitting to the observations. Extensions to KDEs to generate probabilistic forecasts will be explored in Sect. 11.5.

Now consider estimation of the CDF for a univariate distribution with samples $x_1, x_2, \ldots, x_N$. The CDF can be easily estimated using the **empirical cumulative distribution function** (ECDF) defined by

$$\hat{G}_X(x) = \frac{\text{number observations less than } x}{N} = \frac{1}{N}\sum_{k=1}^{N} \mathbf{1}_{x_k < x}, \tag{3.19}$$

where $\mathbf{1}_S$ is the indicator function which takes the value 1 if the statement S is true and 0 otherwise. In this case the statement is whether the observation is less than x. In other words, the empirical CDF simply counts the proportion of observations less than a particular value. An example of the CDF for the standard Normal and the corresponding empirical CDF (for 20 randomly sampled points) is shown in Fig. 3.7. Notice the Empirical CDF jumps at every point observed and the more observations available the closer the approximation is to the true CDF.

The quantiles (Sect. 3.2) can also be estimated from a finite sample of points. Suppose the observations are ordered, i.e. the samples $x_1, x_2, \ldots, x_N$ are such that $x_k < x_{k+1}$ for $k = 1, \ldots, N-1$ (These are also known as order statistics). Then the q-sample quantile, for $q \in (0, 1)$ is defined as the closest x_k where k rounds to qN.

Of course the PDF estimate created from the KDE can also be easily turned into a CDF estimate using the definition of the CDF itself (Sect. 3.1). However, since often the kernels are distributions themselves (as with the Gaussian) means that the CDF is simply the sum of the CDFs of each kernel.

Sometimes it is not necessary to visualise the entire distribution of points and a good general impression of the distribution can be understood from a few points. A **boxplot** gives a visualisation of a few summary statistics of a distribution. An example of a box plot for two data sets is shown in Fig. 3.8 where the first data set is the same as that shown in Fig. 3.6, whilst data set 2 is a simple Gaussian distribution with mean and standard deviation equal to 0.5. What is included in a box plot can vary but they all typically show the following things:

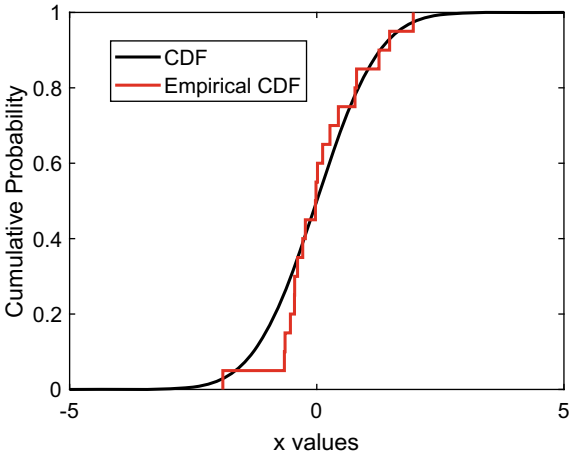


Fig. 3.7 Example of empirical CDF with true CDF for the standard normal distribution. The ECDF was generated using 20 randomly drawn points from the true CDF

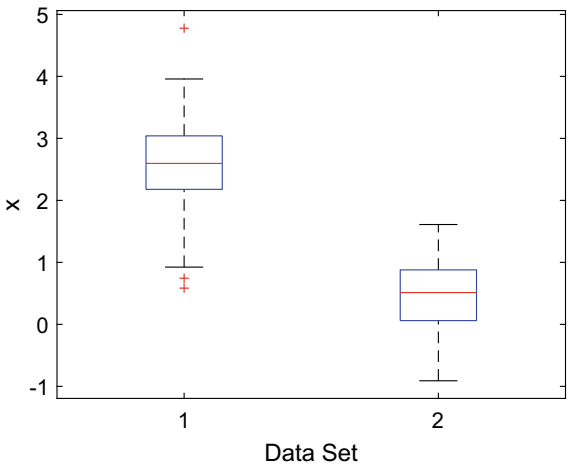


Fig. 3.8 Example of a box plot for two data sets. Data set 1 is the same as that used in Fig. 3.6. Data set two is just a simple Gaussian with mean and standard deviation equal to 0.5

- A centralised value which is given by a line within the main box. In the boxplot in Fig. 3.8 this is the median and is given by the red line.
- The first and third quartile which are given by the bottom and the top of a box.
- Whiskers which show the span of the points to the smallest and largest values (often this does not include points considered outliers). These are the dotted lines in Fig. 3.8.
- Outlier values defined as those which are more than 1.5 times the interquartile range from the top or bottom of the box. These are given by red crosses in the plot.

The box plot, although relatively simple can be used to generate some insight to the data. Firstly it gives a very basic representation of the spread of the data, including where the middle 50% of the data lies. The comparison of the box plot for the two data sets can also indicate whether there is significant overlap between the data. Finally, if the median line is not in the centre of the box then this indicates skewness in the data.

How is this translated to short term load forecasts?

Nonparametric methods provide a data-driven way to develop estimates of uncertainty with few assumptions about the data. Although there are hyper-parameters (such as the bandwidth for the kernel density estimates), unlike parametric models there is minimal input required from the modeller and therefore this allows for more automation and scaling. Nonparameteric forecasts models will be considered in more detail in Chap. 11.

3.5 Sample Statistics and Correlation

The expected value and variance are important values associated to distributions and are often used as estimates of 'typical' values and uncertainty respectively. However, in practice the distribution is not known and important features of a distribution can only be estimated from the available observations. Suppose $x_1, \ldots, x_N$ is a sample of observations of a univariate continuous random variable X, and each of the samples is independent (i.e. none of the samples are dependent on each other). A population estimate for the mean is the **sample mean**, defined as

$$\hat{\mu} = \frac{1}{N} \sum_{k=1}^{N} x_k. \tag{3.20}$$

Similarly there is the sample variance

$$\hat{\sigma^2} = \frac{1}{N-1} \sum_{k=1}^{N} (x_k - \hat{\mu})^2, \tag{3.21}$$

which is divided by $N - 1$ rather than N, this is to ensure the estimator is unbiased.[3] As in Sect. 3.1 the sample standard deviation is the square root of the variance σ. Other important measures of central tendency include the **median** (the 0.5-quantile) and the **inter-quartile range** (the difference between the 0.75 and 0.25 quantiles). These values also tend to be more robust (i.e. are less sensitive) to outliers than the mean and variance.

In Sect. 3.3 the concept of covariance and correlation between random variables was introduced for continuous random variables for when the distributions are known. Consider observations $(x_{k,1}, x_{k,2})$, $k = 1, \ldots, N$ for the bivariate random variables (X_1, X_2), then the sample **Pearson correlation** can be defined as

$$\rho = \frac{\sum_{k=1}^{N} (x_{k,1} - \bar{x}_1)(x_{k,2} - \bar{x}_2)}{\sqrt{\sum_{k=1}^{N} (x_{k,1} - \bar{x}_1)^2} \sqrt{\sum_{k=1}^{N} (x_{k,2} - \bar{x}_2)^2}}, \tag{3.22}$$

where $\bar{x}_1, \bar{x}_2$ are the sample means for random variables X_1 and X_2 respectively.

In addition to the Pearson's correlation, another common measure of correlation is **Spearmans rank correlation** coefficient. This is defined as simply the Pearson correlation of the rank of the values in the two random variables. In other words, take the observations $x_{1,1}, x_{2,1}, \ldots, _{N,1}$ for random variable X_1 and assign them based on their rank, i.e. value 1 for the largest value, 2 for the second largest and so on. Similarly do the same for the second random variable, X_2. Then simply calculate the Pearson correlation of these rankings using Eq. (3.22).

[3] Which essentially means the difference between the expected value of the estimator and the true value is zero.

The concept of correlation can be expanded to a univariate time series (Sect. 5.1), i.e. $(L_1, L_2, \ldots, L_N)^T$ where L_t is a single value at time t and the points are ordered consecutively in time. First consider the **autocovariance** and the **autocorrelation**—i.e. the (Pearson) correlation between values in the time series and its lagged values. The sample autocovariance function at lag k is defined as

$$R(k) = \frac{1}{N} \sum_{t=1}^{N-k} (L_t - \hat{\mu})(L_{t+k} - \hat{\mu}), \tag{3.23}$$

where $\hat{\mu}$ is the sample mean of the time series $(L_1, L_2, \ldots, L_N)^T$. Similarly the autocorrelation function (ACF) at lag k is defined as

$$\rho(k) = \frac{1}{N\hat{\sigma}^2} \sum_{t=1}^{N-k} (L_i - \hat{\mu})(L_{t+k} - \hat{\mu}) = \frac{R(k)}{R(0)} = \frac{R(k)}{\hat{\sigma}^2}, \tag{3.24}$$

where $\hat{\sigma}^2$ is the sample variance.

A plot of the autocorrelation at several consecutive lags $\rho(0), \rho(1), \ldots$ is a common way to better identify interdependencies of a time series to its lagged values as well as to identify important features such as seasonalities. They are commonly used to identify the correct orders for the ARIMA models (see Sect. 9.4). The autocorrelation is bounded $-1 \leq \rho(k) \leq 1$ for $k \in \mathbb{N}$ with $\rho = 1$ indicating a perfect positive correlation and $\rho = -1$ a perfect negative correlation. No correlation whatsoever is indicated by $\rho = 0$.

The **Partial autocorrelation function** (PACF) evaluated at lag k describes the autocorrelation between the series L_t and the lagged series L_{t+k} *conditional* on the in between values $L_{t+1}, \ldots, L_{t+k-1}$. In other words it describes the autocorrelation after removing the effects at shorter lags.

An example of an autocorrelation and partial autocorrelation plot for 30 lags is shown in Fig. 3.9. Typically such plots also include two horizontal lines which indicate the level at which significant correlations exist. Values outside of the area defined by these lines indicate significant correlation values (although of course sometimes values can be outside these lines by chance alone).

The ACF plot shows some strong autocorrelations at lags 6, 12, 18, 24 but also other lags in between. The PACF plot also shows the same strong lags at periods of 6 however, many of the other correlations are much smaller (for example at lags 3, 4 and 5) in the PACF plot compared to the ACF plot, which indicates that the influence of the other lags may have suggested a stronger autocorrelation than truly existed in the time series.

The **cross-correlation** is an extension of autocorrelation except between two different time series. To illustrate this, consider a second time series $M_1, M_2, \ldots, M_N$, defined at the same time steps as $(L_1, L_2, \ldots, L_N)^T$. The cross-correlation between L and M at lag $k \geq 0$ can be defined as

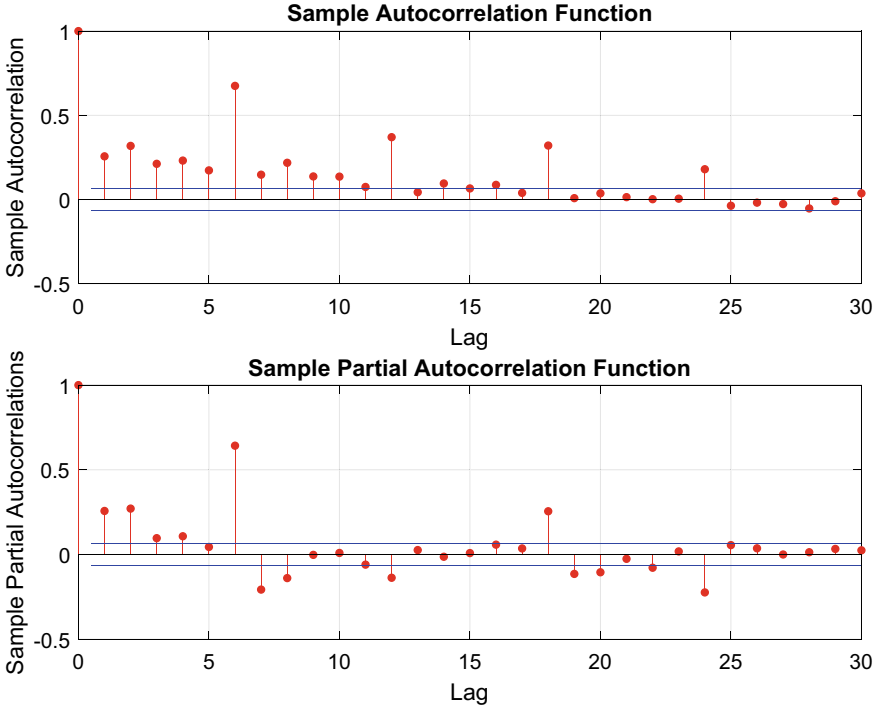


Fig. 3.9 Example of autocorrelation (top) and partial autocorrelation (bottom) plots for the same time series

$$\mathrm{XR}(k) = \frac{1}{N}\sum_{t=1}^{N-k}(L_t - \hat{\mu}_1)(M_{t+k} - \hat{\mu}_2), \tag{3.25}$$

and for $k \leq 0$ as

$$\mathrm{XR}(k) = \frac{1}{N}\sum_{t=1}^{N-k}(M_t - \hat{\mu}_2)(L_{t+k} - \hat{\mu}_1), \tag{3.26}$$

where $\hat{\mu}_1$ and $\hat{\mu}_2$ are the sample means of the time series for L_t and M_t respectively. The value at $k = 0$ represents the correlation between the two series without any lags. The function not only identifies which lags are the most important but also the temporal direction. For example, the temperature could be a strong indicator of the demand used, however, if the nearest weather station is in the next town over then this recorded value may be related to the temperature in the town of interest but with a delay. In that case there will be a strong cross-correlation between the demand and the observed lagged temperature at the next town over.

How is this translated to short term load forecasts?

Sample statistics are essential for load forecasting as the time series is not measured continuously but only at regular intervals, usually half hourly or hourly. Therefore to derive and estimate features of the load forecast will require the use of sample statistics.

Further to this the autocorrelation and cross-correlation are useful tools for identifying the important explanatory variables in the model, as well as the lags of the variables to include as model inputs. It should be noted, even though correlation doesn't mean causation, correlation can still be a very important indicator of what will be useful for a forecast model. These statistical measures will be used throughout the book but in particular for the data analysis (Sect. 6.2) and the statistical point forecast models in Chap. 9.

3.6 Questions

1. Generate different size samples from a normal distribution with a mean and standard deviation of your choice. Plot these values against samples of different sizes. Do the values start to converge to the true values? At how many samples does this convergence begin? This is demonstrating the *central limit theorem*. How much does the mean change for small samples? Now plot the median? How much does this change for small samples? Does one seem more robust to additions of new data than the other?
2. For the standard normal distribution, how much data lies between (a) one standard deviation, σ, (b) two standard deviations, (c) three standard deviations, from the centre?
3. For a standard normal distribution, which quantiles most closely approximate the values one standard deviation from the centre?
4. If a distribution is not symmetric what does the difference between the mean and median tell you about the skewness of the distribution?
5. Sample from a univariate distribution of your choice and plot the histogram (using the default bin size). Change the bin sizes and number of samples and compare them to the original distribution. What appears to be a good bin size for one of your sampling sets (assuming the sample size is big enough)?
6. Sample from the same univariate distribution as in the previous question. Plot a kernel density estimate with different bandwidths and different sample sizes. Observe the effects, what appears to be a good choice of bandwidth for one of your sample sets (make sure the number of samples is big enough)?
7. Plot the box plots for three datasets, with each data set generated from 1000 samples from three different univariate distributions (perhaps use the Gamma, log-normal and Gaussian distributions described in the chapter). Adjust the parameters of each model so the plots can all be seen clearly on the figure. Compare

these plots to the original distributions: How does the central value vary across the box plots? What about the whiskers and the box sizes?

8. Plot a bivariate Gaussian distribution with unit variance on both variables. Change the correlation between the plots and see how they change. How does the marginal distribution change for each variable as you change the correlation?
9. For the same bivariate distribution as the previous question. Consider the conditional distribution for one of the variables by binning samples of the data contained within a small interval of the other variable. Plot the histogram of these points. How does this change as you change the position of the interval?
10. Download some energy time series data (See Sect. D.4). Plot the autocorrelation and partial autocorrelations for a few of these time series (only consider lags for up to a couple of weeks). Where are the major correlation values? What is the difference between the autocorrelation plots and the partial autocorrelation plots? Compare these values to plots of the time series. Are there obvious seasonal patterns, and how do they relate to the autocorrelation plots?
11. Download the GEFCOM 2014 data or alternatively download any other demand time series data which also has some temperature data (See Sect. D.4 for a list of data sets). Plot one of the demand series against the temperature data in a scatter plot. What does the relationship look like? Is there a clear affect of temperature on this data? Calculate the cross-correlation plot for the demand and temperature data. Where are the major correlations. Is there some strong correlations at lagged values?

References

1. F.M. Dekking, C. Kraaikamp, H.P. Lopuhaä, L.E. Meester, *A Modern Introduction to Probability and Statistics: Understanding Why and How* (Springer, London, 2005)
2. D. Ruppert, D.S. Matteson, *Statistics and Data Analysis for Financial Engineering: with R Examples*. Springer Texts in Statistics (2015)

Chapter 4
Primer on Machine Learning

The previous chapter introduced various concepts from statistics and probability relevant to forecasting. As many state-of-the-art approaches rely on machine learning, this chapter will introduce some fundamental definitions and concepts. It does not intend to provide an in-depth understanding but instead plans to overview the main concepts as they are relevant to this book. It provides an overview of practically relevant concepts when using software packages to fit and configure machine learning models. It does not introduce specific algorithms as those machine learning algorithms that are typically used in load forecasting are discussed in detail in Chap. 10. More in-depth overviews of the approaches can be found in the list of further reading in Appendix D.2.

4.1 Definitions and Related Concepts

A common definition of **machine learning** is that it includes algorithms that enable computers to learn from **data** and improve **performance** within a specific **task** without being explicitly programmed. A typical such task is to describe the relationship between a set of input variables that are typically measured or preset and have some influence on one or several outputs (see the next section for other machine learning tasks).

With this definition, machine learning can be distinguished from classic **algorithms** and programs studied in computer science, i.e., a finite sequence of well-defined instructions. A traditional algorithm performs a task deterministically following the steps that have been implemented by the programmer at design time, resulting in a specific performance. In contrast, a machine learning algorithm uses experience/observations, i.e. data, to improve the performance within the task, possibly even when in operation. This allows the algorithm to tackle tasks that are too difficult to solve with classical algorithms and programs written and designed by human beings. A second way of distinguishing machine learning from classical pro-

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_4

gramming is that for classical programming, one provides the input and a function (the algorithm) to compute an output. In machine learning, one provides some example inputs and outputs to find a function which can then be used with new inputs to compute outputs.

The above definition can also be used to distinguish machine learning from **statistics** (see the introduction to some basic concepts in Chap. 3). One notion of distinguishing the two is that statistical models are designed for **inference** about the relationships between variables, and machine learning models are developed to make the most **accurate predictions**, i.e. to maximise performance. A common purpose of statistical models is to make inferences about the relationships between variables, i.e., creating a mathematical model of the process by which data was generated to formalise understanding or test a hypothesis about the system behaviour.

While statistical models can also make predictions, this is often achieved by making parametric assumptions, like assuming that data follows a specific distribution (see Chap. 3). While this improves model understanding and **interpretability**, it introduces **model bias** that for complex data may hinder achieving accurate predictions. Similarly, as too many input variables hinder the interpretability of the process, statisticians may want to avoid situations where $p >> n$, i.e., the number of input variables is much larger than the number of samples. Hence, a statistician may want to attempt to avoid over-parameterisation by performing a variable selection, removing terms that do not contribute significantly to the inference. This generally improves model understanding but may inhibit prediction accuracy, as even small contributions can improve predictions.

In contrast, a machine learning model aims to learn from experience (past data) with the main goal of improving a prediction, in other words it typically sacrifices understanding of the underlying mechanisms for performance. Machine learning models are mostly non-parametric methods (e.g. see Sect. 3.4) and are typically capable of handling many input variables without extensive manual preprocessing or feature selection. Utilising complex models and removing restrictive assumptions, machine learning applies optimisation techniques to find approximate algorithmic solutions (see Sect. 4.3). In contrast statistical models can sometimes find exact closed-form solutions. Alongside optimisation, machine learning methods are often concerned with developing methods to avoid **overfitting**, i.e., finding models and methods that are capable of **generalising** well to new data points that they have not been trained on. This includes different **regularisation** methods (see Sect. 8.2.5). All this is done to improve prediction performance at the cost of interpretability compared to statistical methods.

While the previous paragraphs highlighted some main differences between machine learning to classic computer algorithms and statistics, they are not completely distinct. There are considerable overlaps between the concepts, and machine learning heavily relies on classical computer science and statistics techniques and approaches. For instance, knowing the data-generating process typically provides insights into what makes a good predictor and is often an essential step in the applied modelling process. Machine learning also relies on several areas of computer science and algorithms, for instance, when datasets become too large to fit into the memory

of a single computer (or its GPU). Much of practical machine learning is concerned with developing strategies to manage millions, billions and even trillions of parameters using computer science methods like distributed computing (this aspect is not addressed in this book). Beyond those, machine learning also heavily relies on other mathematical concepts such as optimisation, matrix algebra and calculus.

How is this translated to short term load forecasts?

As in many domains in recent years, machine learning models have increasingly been applied to forecasting problems. In contrast to many other disciplines like image recognition or natural language processing, for time series forecasting they have not yet dominated over other approaches in the field, like simple benchmarks and more sophisticated statistical approaches, as discussed in Chap. 9. This is because many time-series problems have limited data availability, and many machine learning algorithms are often unnecessarily complex. Only in recent years have machine learning models, for instance, recurrent neural networks or gradient boosting, started to outperform other methods consistently (see, for example, the discussion of the M5 time series forecasting competition [1]). With the advent of specialised deep learning approaches like DeepAR [2] and N-BEATS [3], there is a likely trend that more advanced specialised machine learning models may improve in many time series problems. However, as statistical models allow for a better understanding of the relationships between the available data and the forecast, an accurate statistical model should always be used as a benchmark in the load forecasting process (see Chap. 9). The M5 competition showed [1], that *combinations* of statistical and machine learning models can reach state-of-the-art results with the advantage of remaining at least partly interpretable, combining the benefits of both approaches. This makes them particularly interesting for real-world applications.

4.2 Machine Learning Taxonomy and Terms

The former section introduced **machine learning** as algorithms that enable computers to improve the performance within a specific **task** by learning from data. The most common machine learning task is to describe the relationship between a set of input and output variables. This task is called **supervised learning**. Besides supervised learning, there are other sub-types of machine learning, most importantly **unsupervised learning** and **reinforcement learning**.

4.2.1 Supervised Learning

Supervised learning is the task of learning the relationship between a set of input and output variables from data, i.e., to learn some function f to be able to make predictions. These inputs are often referred to as **instances** or **examples**. An instance is a collection of values that can be measured or obtained in some way (e.g. through sensors). It is often denoted as $\mathbf{X} \in \mathbb{R}^n$. In statistical terms, an instance is a realisation vector of the n random variables $X_1, X_2, \ldots, X_n$, so that one can also write $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ (Sect. 3.1). These inputs are often called predictors or independent variables in the statistical literature. In the machine learning literature (and the remainder of this Chapter), the term **features** is more commonly used. Throughout this book, both terms will be used.

A statistician may refer to the outputs as the **response** or the **dependent variables**. Depending on if the task is to predict a numeric variable or a qualitative variable (like the membership to a **class**), in machine learning, the output is referred to as **target** or **label** in the machine learning literature. If the supervised learning task is to predict a numeric variable, the task is referred to as **regression**. To solve this, the machine learning algorithm aims to model a function $f : \mathbb{R}^n \to \mathbb{R}$, or $f : \mathbb{R}^n \to \mathbb{R}^k$ in the case of multiple target variables. In the latter case, the regression problem may be referred to as multi-target or multi-output regression. Note this is not to be confused with multiple regression or multivariate regression, which refers to the dimensionality of the inputs, i.e., $n > 1$. Note that the term regression is sometimes used to denote linear regression (see Sect. 9.3). However, linear regression is only one specific regression method, and the regression task can be performed using many different algorithms, as will be discussed in this book.

A common machine learning task is to predict the assignment of an instance to one of k categories, or **classes**. Here, the machine learning algorithm is equivalent to modelling a function $f : \mathbb{R}^n \to \{1, \ldots, k\}$, with each class assigned an integer. This output can be a numeric code representing a specific class, but more commonly, machine learning algorithms produce a probability distribution over the classes. This task is referred to as **classification**. Depending on whether the output is univariate or multivariate, the output is denoted as $Y \in \mathbb{R}$ or $\mathbf{Y} \in \mathbb{R}^k$ (for the sake of simplicity in the following the univariate case is presented as a special case of the multivariate case for $k = 1$).

The relationship f is learned from a **dataset**. While formally, the order should not matter, a dataset can be seen as a set of N tuples of instances with the respective labels or targets, i.e., $\mathcal{X} = \{(\mathbf{X}_1, \mathbf{Y}_1), (\mathbf{X}_2, \mathbf{Y}_2), \ldots, (\mathbf{X}_j, \mathbf{Y}_j), (\mathbf{X}_N, \mathbf{Y}_N)\}$. However, as many algorithms make use of linear algebra, it is also common to denote the set of inputs of the dataset as a matrix $\mathbf{X}$ and the set of output as a vector or matrix $\mathbf{Y}$. Note that the inputs have been introduced as n-dimensional vectors for simplicity, but for different machine learning tasks, they may be of higher dimensionality, e.g. for images or videos. Hence, they may be considered more generally as **tensors**. However, for convenience and clarity, the focus will be on matrices and vectors, which will be the most common form for load forecasting.

4.2.2 Unsupervised Learning

Unsupervised learning typically refers to algorithms that learn and identify patterns within the data without labels. Denote the dataset as set $\mathcal{X}$ with N members $\mathbf{X}_1, \ldots, \mathbf{X}_i, \ldots, \mathbf{X}_N$ with each $\mathbf{X}_i = (X_1, \ldots, X_j, \ldots, X_n)^T$ consisting of n features. Unsupervised learning tasks can be used to either model a pattern within the set of data $\mathcal{X}$, model within each of the instances $\mathbf{X}_i$, or both.

The type of problem that is most commonly associated with finding patterns within the dataset is the task of finding partitionings or groupings of a dataset, i.e. **clustering** a dataset. The goal is to find k groupings $\mathcal{X}'_1, \ldots, \mathcal{X}'_i, \mathcal{X}'_k$ such that $\cup_{i=1}^{k} \mathcal{X}'_i = \mathcal{X}$. The most common algorithms are the centroid-based **k-means clustering**, distance-based **hierarchical clustering** and the density-based models **DBSCAN** and **OPTICS**. A discussion of those algorithms is not part of this book, but see [4] for k-means and hierarchical clustering and [5] for a discussion on DBSCAN and OPTICS. An example of another approach which can be used for clustering, called **finite mixture models** is given in Sect. 11.3.2, except in this case it is used to model complex distributions.

The most common reason for finding patterns within each of the individual instances $\mathbf{X}_i$ is **dimensionality reduction** and the related problem of finding **embeddings**, i.e., meaningful **latent** representations of the data. Dimensionality reduction methods aim to address the curse of dimensionality[1] by finding lower-dimensional representations of the data. This lower-dimensional representation of the data can be used as features, for instance, in supervised learning or forecasting. A model that uses lower-dimensional data as input to make the predictions may be referred to as a **down-stream model**. It can also be useful for visualising high-dimensional data in 2D or 3D representations. Popular methods are principal component analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE). Figure 4.1 shows the result of using PCA to reduce the time series of three households from 96 values per day (15-min data) into two dimensions. On the left, there are some example samples of three households in 15-min resolution. One can see that the behaviour is generally quite different with load at different times of the day. On the right is a scatter plot of the two-dimensional data after applying PCA with the goal of finding two descriptive features of the data. The colour highlighting of the specific households is added to illustrate how even though the dimensionality has been drastically reduced, the data of the different households generally stay together, indicating that some differences between them are preserved in this low dimensional representation.

A detailed discussion of these methods is not within the scope of this book. Finding a suitable latent representation of raw inputs is essential in working with image or text data. For instance, many machine learning models need fixed-length numerical input sequences and cannot handle sentences of different lengths. It is commonly

[1] In short, the curse of dimensionality is the rapid increase in observations that are required for accurately estimating relationships as the number/dimension of features in the inputs increases. To illustrate this note how points in 2-dimensional space are much less isolated than points in 3-dimensions.

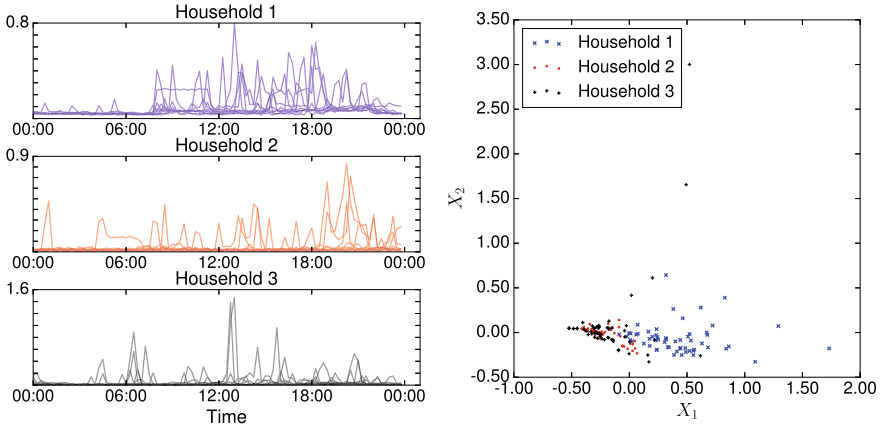


Fig. 4.1 Samples of a dataset of household-data in 15-min resolutions (left) and the resulting two-dimensional representation after applying PCA (right)

associated with neural networks (Sect. 10.4). The goal is to find a lower-dimensional representation of a fixed width, also for different length inputs (e.g., of different sentence lengths) in a latent space where similar instances are closer together.

Density estimation can also be considered an unsupervised learning task and learns patterns within both the dataset and the instances. Section 3.4 introduces some more classical approaches to density estimation from the statistics domain. However, more recently, for complex distributions like text, images and video, **generative machine learning** methods like variational auto-encoders (VAE), normalising flows, generative adversarial networks (GAN), and diffusion models have been introduced. Some of these concepts are sometimes applied to time series, including load data, but are not well established yet in this domain and are hence not further discussed in this book. The goal of the model is to estimate a distribution (see Sect. 3.1), e.g., to make inferences about uncertainties in the forecast estimate, or it can also be used to sample new data (or realisations) from the distribution of the dataset.

More recently, additionally semi-supervised and self-supervised learning methods have emerged, for instance, as part of large language models. Here, either labels partly exist to train embedding models, or they are created by partially obscuring parts of the data. Latent representations are also the core of generative models. However, those are not well established yet in this domain and are hence not further discussed in this book.

4.2.3 Reinforcement Learning

The third common machine learning sub-category is reinforcement learning. In contrast to unsupervised and supervised learning algorithms, it contains algorithms that do not learn from a fixed dataset but interact with the environment, i.e., a feedback

loop between the learning system and data from the environment. The reinforcement learning task is typically modelled as a **Markov decision process (MDP)**, where the learning system is denoted as an **agent** that interacts with an **environment**. First, it observes the current **state** of the environment and chooses an appropriate **action** based on a learned **policy**. The action results in a **reward** as feedback from the environment. Then the environment is in a new state, and the process continues in an iterative manner. The policy is adjusted based on the rewards so that the agent learns. Depending on the algorithms, the agent chooses actions that are known to work well (**exploitation**), or it may explore new actions (**exploration**). As the policy maps state/action pairs to rewards, this mapping can be supported by supervised learning algorithms and, more recently, deep learning algorithms (Sect. 10.5).

However, often real-world applications suffer from different challenges. One problem is the **credit assignment problem**, where often the reward signal is delayed and only available after a sequence of actions, making it difficult to attribute the influence of individual actions to the reward. A second problem is **reward hacking**, where misformulated objective functions can lead to unexpected results. Lastly, the exploitation of new actions may often not be desirable in an application where the cost of ineffective actions is high. Here, simulations (often referred to as **gyms**) can support learning an optimal policy. Reinforcement learning concepts have also been partially applied to the time series forecasting problem, but they are not well established yet and are not further discussed in this book.

How is this translated to short term load forecasts?

Time series forecasting can be formulated as a multivariate regression problem. Therefore supervised machine learning models can generally be applied to time series forecasting problems. While supervised approaches are the most common, unsupervised and reinforcement learning methods can be used in forecasting, either on their own or, more commonly, in combination with supervised approaches. For instance, when forecasting an aggregated time series, clustering techniques can be used to partition the data of the time series that make up the signal. Models are then trained on the resulting clusters before combining the predictions. This can produce improved accuracy compared to directly forecasting the aggregated time series since individual series may have similar features which can be used to design an accurate forecast model of the cluster. Further, dimensionality reduction can improve forecasting models and be applied in exploratory data analysis to identify important patterns and/or develop new inputs.

4.3 Introduction to Optimisation with Gradient Descent

As described above, when using machine learning, the optimal parameters have to be determined based on the prediction errors on a training set. For instance, in neural network models (Sect. 10.4), the weights of the networks must be trained. However, besides the prediction error, other components affected by the model must be optimised at the same time, for instance, the range (or constraints) of the chosen parameters (see Sect. 8.2.4 on regularisation). Overall, this function to be optimised is called the **loss or cost function**.

Whereas optimisation problems in many of the statistical methods can be solved using an analytical **closed-form solution**, most optimisation problems in machine learning (like finding the weights of a neural network) are rather complex, e.g., having non-convex cost functions, i.e., they are having multiple **local optima** or **saddle points**. Figure 4.2 gives an example of such a loss function of a 56-layer pretrained convolutional neural network. For simple cost functions, a **global optimum** can often be found, but for more complex cost functions, the solution will often be only a local optimum. Hence optimisation is often not deterministic, in other words running the same procedure (e.g. the training of a neural network) can result in different solutions, i.e., models that perform better or worse.

Most state-of-the-art machine learning models, especially neural networks, use some form of gradient descent. The **parameters**, i.e., the weights β, are adjusted

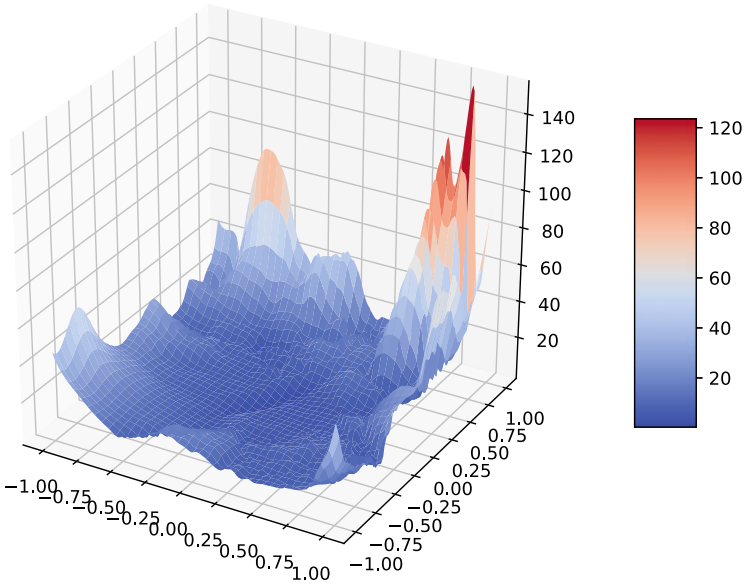


Fig. 4.2 Example loss function along two random normalised directions of pretrained 56-layer convolutional neural network ResNET-56 for image recognition, created with Loss Landscape tool described in [6]

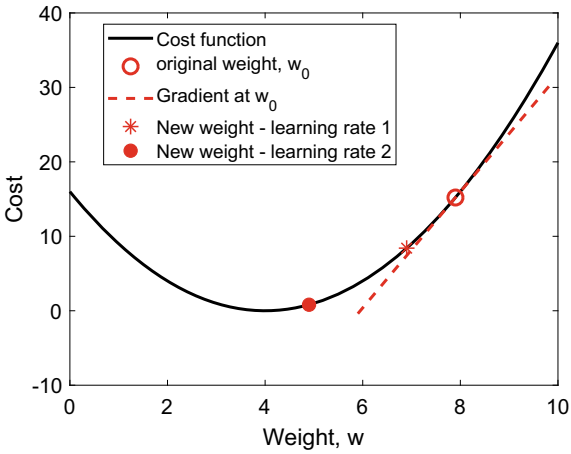


Fig. 4.3 Illustration of gradient descent for a basic cost function. The value w is updated based on the direction in which the gradient descends. The length the update moves is based on the learning rate. Two different learning rates are illustrated here

according to the gradient of the loss function (see the introduction on artificial neural networks in Sect. 10.4). Hence, if the gradient of the cost function is positive with respect to some current weight, it means the cost will decrease if the weight decreases. Similarly, if the gradient is negative then the cost function will decrease if the weight increases. It should be noted that many statistical approaches also use gradient methods, especially for problems with large numbers of parameters and observations where finding the optimal directly in a closed-form is difficult.

To demonstrate this process consider a basic cost function (defined by $(w-4)^2$) as shown in Fig. 4.3. Initially, the weight is $w = 7.9$, and the cost function has a positive gradient at this point. This shows that local to this weight, increasing the weight will increase the cost function. Hence, the weight should be reduced to reduce the value of the cost function and find a value closer to the optimal (zero in this case with $w = 4$). The process is then repeated at the new value. The question remains of how much to reduce the value. This is determined by the so-called **learning rate** or **step-size**. Two different learning rates are shown in Fig. 4.3 where one gets closer to the true minimum (at $w = 4$) for a single step compared to the other rate after one iteration. Clearly, the learning rate is an important hyper-parameter for the algorithm (see Sect. 8.2.3 on tuning hyper-parameters). Too small, and the convergence will take too long (and potentially be stopped too early) to reach the minimum. Too big, and the solution will be too unstable and potentially not converge at all.

In this simple example, the gradient was determined for the full training data. This is called **batch gradient descent**. It calculates the error for each instance using efficient matrix operations and takes an average over the whole dataset to determine the gradient. This averaging provides a stable learning path and hence leads to quick convergence. This is only feasible for simple problems and small datasets. For larger neural networks, for instance, convolutional neural networks and large datasets (e.g. images), this is not feasible as the gradient calculation becomes too computationally complex and the matrices will not fit into your computer's memory (although even

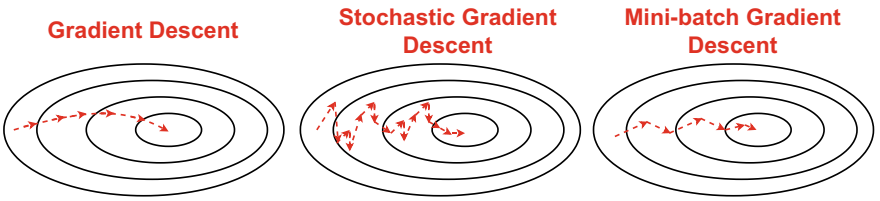


Fig. 4.4 Illustration of the convergence path of gradient descent (smoothly), stochastic gradient descent (unstable) and mini-batch gradient descent (compromise)

if you had large amounts of memory this is perhaps an inefficient use of resources). Instead, on large datasets, using samples from the dataset can often be enough to produce a good approximation of the current iteration's gradient.

When only one instance is considered at each step, this is called **stochastic gradient descent**. The advantage of stochastic gradient descent is that the algorithm is much faster at every iteration. However, the algorithm results in a less regular and stable learning path compared to batch gradient descent. Instead of decreasing smoothly, the cost function will "zig-zag" and may even temporarily move "back up the hill", as individual samples do not accurately approximate the entire dataset's gradient. While this may seem like an undesirable property, this is helpful in training complex neural networks as it introduces a randomness to the optimisation procedure that can help the estimate escape local minima or saddle points. Hence, most current deep learning models use a variant of **mini-batch gradient descent** that combines the concepts of batch and stochastic gradient descent. Figure 4.4 illustrates the paths of these three gradient descent algorithms.

In mini-batch gradient descent, the algorithm computes the gradient based on a subset of the training set at each step instead of the complete dataset or only individual instances. This provides a trade-off, as it takes advantage of efficient matrix operations during the gradient calculation, resulting in a smoother and more stable convergence than stochastic gradient descent. One downside is that this introduces additional **hyperparameters** to the training process, as besides the step-size, a **batch-size** also has to be provided. The mini-batch size is chosen to ensure enough diversity to escape local minima while providing some stability and enough computational efficiency from fast matrix calculations.

Besides the way the training set is split up to calculate the gradient, there are many other ways that the vanilla gradient descent can be improved. One popular modification is to add **momentum**. Regular stochastic gradient descent may have trouble in areas of the loss function where the surface gradient is much steeper in one dimension than in another. Here, a momentum term is added that increases for dimensions where the gradients point in the same directions and decreases for dimensions where gradients change directions. This results in faster convergence and fewer oscillations when moving towards a local optimum (see Fig. 4.5 for an illustration).

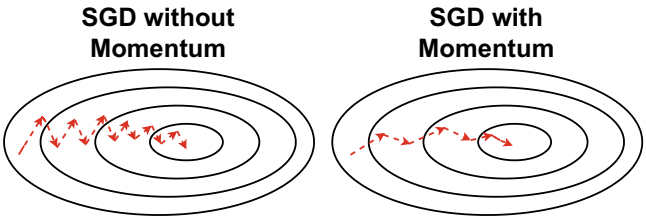


Fig. 4.5 Illustration of the convergence path of stochastic gradient descent without momentum and with momentum

Many optimisation algorithms introduce some form of **adaptive learning rate** that changes the step size value according to some rule. Simple schedules may **decay** (reduce) the step size over time to produce larger changes at the beginning of the training process and then fine-tune towards the end when moving towards a local optimum. A variant is a **cycling learning rate** where the learning rate iteratively becomes larger and smaller according to a rule to improve the chances of escaping local optima. An overview of the many existing optimisation algorithms is out of the scope of this book. An overview of some popular optimisers is given in [7]. One effective and popular optimisation algorithm is Adaptive Moment Estimation (Adam), which uses an adaptive learning rate and the ideas behind momentum. It will usually converge faster to a local minimum than using vanilla stochastic gradient descent without momentum with a simple learning rate decay schedule. Further, it is less prone to get stuck in saddle points and relies less on initialisation parameters like step size and decaying schedule. Hence, Adam with default hyper-parameters is a decent method to use in practice for training deep neural networks.

4.4 Questions

For the questions which require using real demand data, try using some of the data as listed in Appendix D.4.

1. Select the time series of an individual household or building and arrange your dataset in a matrix so that each row represents one day. Resample it to hourly data so that the matrix has 24 columns. Apply principle component analysis (PCA) using a library of your preference, for instance, Scikit-Learn[2] to reduce the dimensions. Then also, apply t-SNE for comparison. Visualise the results for two dimensions and three dimensions using scatter plots. Do you see clusters of data points? Find explanations for what could cause different groupings. Verify your hypothesis by using different colouring in the plot. Next, repeat the application of each of the algorithms. Do you see variation in the results?

[2] https://scikit-learn.org/stable/index.html.

2. Use a load dataset that contains data from several households. Create a matrix $\mathbf{X_1}$ by rearranging the data so that each row of the matrix represents one day of one household. Run k-means clustering with the objective of finding 10 clusters. Visualise members of each of the clusters and compare them. Do you see distinct clusters or are their clusters with very similar patterns that could be merged? Try less target clusters. Does this result in "better" groupings? In the absence of a "ground truth", what could generally be ways to assess and compare the results in clustering?
3. Clustering can be done using the raw load data or by modelling features from the load data. $\mathbf{X_1}$ contains the raw data already. Next, create a matrix $\mathbf{X_2}$ by applying PCA to reduce the dimensionality of the data to 6 components (i.e., the matrix has 6 columns). Create a third matrix $\mathbf{X_3}$ where you derive some manual features, such as the mean and max of certain times of the day. Repeat the k-means clustering in the previous question with the target of 10 clusters. Visualise cluster members of each cluster again and compare the results across the 3 representations (raw, manual feature and automated features from PCA). Do the results vary much? Does one method result in more distinct clusters?

References

1. S. Makridakis, E. Spiliotis, V. Assimakopoulos, The M5 accuracy competition: results, findings and conclusions. Int. J. Forecast. (2020)
2. David Salinas, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Deepar: probabilistic forecasting with autoregressive recurrent networks. Int. J. Forecast. **36**(3), 1181–1191 (2020)
3. B.N. Oreshkin, D. Carpov, N. Chapados, Y. Bengio, N-beats: neural basis expansion analysis for interpretable time series forecasting (2019). arXiv:1905.10437
4. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics (2009)
5. M. Ankerst, M.M. Breunig, H.-P. Kriegel, J. Sander, Optics: ordering points to identify the clustering structure. ACM Sigmod Record **28**(2), 49–60 (1999)
6. H. Li, Z. Xu, G. Taylor, C. Studer, T. Goldstein, Visualizing the loss landscape of neural nets, in *Neural Information Processing Systems* (2018)
7. S. Ruder, An overview of gradient descent optimization algorithms (2017). arXiv:1609.04747v2

Chapter 5
Time Series Forecasting: Core Concepts and Definitions

This chapter will present the main definitions and concepts for time series forecasting. It begins by introducing time series before leading into the general form and definitions of a time series forecast. The following sections will lay the foundations for much of the tools, models and concepts in the later chapters. This chapter will rely on a basic understanding of statistical concepts which will be assumed. Chapter 3 contains a crash course in some of the important elements of statistics and probability and will be referred to throughout.

5.1 Time Series: Basic Definitions and Properties

Time series data will be the core object of study for this book. **Time Series** data are simply a sequence of data points, measured at discrete time points, that are ordered in terms of an increasing time index, i.e. chronologically. Typically, the points are spaced equally in time and the majority of timeseries analysis and methods will assume this is the case. Monitoring equipment used for recording demand, for example smart meters, are designed to collect data at regular intervals, usually half hourly, so this is not an unrealistic assumption and it also simplifies the analysis.[1]

Throughout this book it is assumed that the time series is sampled at uniform (regularly spaced) time steps. A time series will often be denoted by a sequence of letters $X_1, X_2, X_3, \ldots X_N$ where the subscript denotes the time step, with a larger index indicating a chronologically later point. Alternatively, the time series can be written as X_k with time steps $k \in K = \{1, 2, \ldots N\}$. If the series continues forever into the future then $N = \infty$ and $K = \mathbb{N} = \{1, 2 \ldots\}$, the set of natural numbers.

[1] In some cases monitoring equipment may record at less regular intervals. For example, if high resolution monitoring (say second resolution) is required, some equipment may record in a compressed form to save storage space. One way to do this is to only record values when there is a change. However if the data has sufficient resolution it can be accurately interpolated to regular time steps.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_5

The values a time series can take are very diverse and can be discrete values, real numbers, sets of values, or even letters. In the case of this book, since the data are typically energy demand then X_k is a single real-valued random variable (a more detailed explanation of random variables can be found in Sect. 3.1) denoting either power (which will have units watts, W, or kilowatts, kW), or energy (which will have units watt-hours, Wh, or kilowatt-hours, kWh). If the values at each time step consist of only a single variable the series is said to be **univariate**. However, if the series consists of more than one variable per time step, for example say $X_k = (L_k, T_k)$ at each time step, where L_k is the load and T_k is the temperature, then the time series is called **multivariate**. The special case (L_k, T_k), of only two variables, is referred to as **bivariate**.

An important feature of a time series is whether it is **Stationary** or not. A time series of random variables X_t is stationary if the joint distribution over any fixed segment of the data, $X_k, X_{k+1}, \ldots, X_{k+M}$ (for some positive integer M), is the same whatever the temporal shift in the data, i.e. the same as the joint distribution of $X_{k+m}, X_{k+1+m}, \ldots, X_{k+M+m}$ whatever the choice of $m \in \mathbb{Z}$ (See Sect. 3.3 for the definition of joint distribution). In particular it means the expected value and the variance at each time step is fixed.[2] Time series that are not stationary are called, unsurprisingly, non-stationary. Examples of basic stationary and non-stationary time series are shown in Fig. 5.1. Plot (a) is a stationary time series, with each point coming from the same distribution with fixed mean and variance. Plot (b) is non-stationary with values coming from a distribution whose mean and variance increase as time increases. Finally, plot (c) shows a time series with a fixed variance but with a seasonal mean. Stationarity is an important property for many time series forecasting models, for example ARIMA models which will be introduced in Sect. 9.4. They are also easier to model since they have fixed properties in time. It is not trivial to prove that a time series is stationary. Plotting the time series is a typical first check for stationarity, and there are also statistical tests which are briefly discussed in Appendix A.

Two important features that occur often in non-stationary data are **trends** and **seasonality**. Trend in a time series is the general macroscopic (i.e. the low frequency) changes in the data, with the most common being a linear trend, where there is a gradual, linear growth in the time series. Figure 5.1b is as an example with positive linear trend. In energy based applications, an increasing trend in energy demand could be due to, for example, the gradual uptake of less energy efficient technologies, or perhaps simply the uptake of more devices.

Seasonality is defined to be changes in the time series that occur at fixed regular intervals or fixed periods. Often demand behaviour is driven by human behaviour hence there are often strong periodicities at the daily, weekly and annual levels corresponding to typical behavioural patterns. Seasonal time series can often also be called periodic time series. Not all oscillations in behaviour will be of fixed period. For example, shift workers such as doctors and nurses will likely not have standard

[2] In fact, for the purpose of this book often a time series only needs to be *weakly stationary*, which means the mean of X_t and the covariance between X_t and X_{t+l} are constant and only depend on $l \in \mathbb{Z}$.

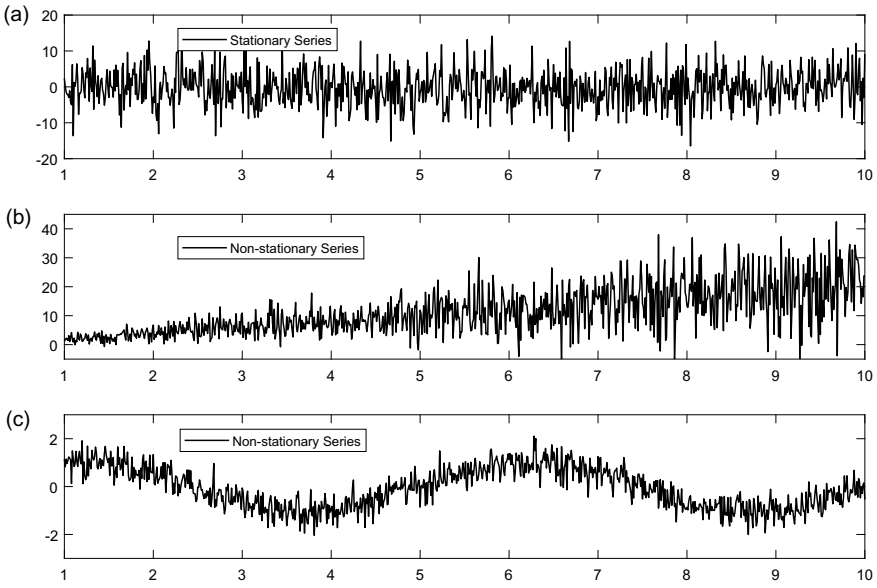


Fig. 5.1 Examples of different time series which are **a** stationary, **b** non-stationary with linear trend, **c** non-stationary with periodic behaviour

daily or weekly patterns (working different days of the week, and perhaps doing a mix of day shifts, long days or night shifts). These are often called *cyclic* patterns. The focus in this book will be on seasonalities with regular periods. An example of a seasonal time series is shown in Fig. 5.1c which has a seasonal period which repeats after every interval of length five.

Finally, another important property of a time series is its **autocorrelation**, which describes how the changes in the time series at one point relate to the time series at **lagged** (or older) points in the time series. A more detailed definition of autocorrelation is given in Sect. 3.5 and is investigated in more detail in Sect. 6.2.4. For now the general principle is described since they are often very important measures for producing accurate forecasts. As a simple example, take a person who gets to work regularly at 8AM every day. One day they may be late to work due to their alarm not going off or their car breaking down, in which case they may decide to work later than usual. In this case their later behaviour is correlated to their earlier behaviour. Notice that a seasonal time series with period P will have relatively high autocorrelation with itself for lags which are multiples of P. Finding correlations in the data is an important part of identifying which historical values may be important for estimating future points (see Sect. 6.2.4 for more details).

So far, only properties of the time series itself have been considered. However, often energy usage is influenced by other external drivers. For example, heating and air-conditioning are obviously related to how cold the occupants feel within a household. Further, the use of lighting will be related to how dark it is outside,

which will in turn also depends on the time of year. Hence the energy demand will strongly depend on external **explanatory variables**. Choosing which external variables to include in a time series model (and its corresponding forecast model) is called **feature selection** and will be described in more detail in Sect. 6.2.

5.2 Time Series Forecasting: Definitions

In its simplest form, a forecast for a time series is an individual, or collection of, estimates for future values using currently available information. For the purposes of this book, the aim will almost always be to accurately forecast the future electricity demand on a low voltage network or application. How we define the accuracy of a forecast will be defined in Chap. 7.

For simplicity, the majority of the following arguments will be in terms of a univariate time series (see Sect. 5.1) but the definitions will easily extend to multivariate time series as well. For the following discussion consider a real-valued, univariate time series $L_1, L_2, \ldots$, defined at uniformly spaced time steps $t_1, t_2, \ldots$, where the current time point is t_n and the aim is to produce a forecast at the next h time steps $h_{n+1}, h_{n+2}, \ldots, h_{n+h}$. Given this scenario a few terms can be defined

- The data $L_1, L_2, \ldots, L_n$, up to the current time t_n, is often referred to as the **historical data** and is a core component of any forecast, especially those with regular seasonal patterns (see Sect. 5.1).
- The current time period t_n is often called the **forecast origin** as it is the starting point for the forecast.
- The value h is referred to as the **forecast horizon** and defines how many time steps beyond the forecast origin are to be estimated by the forecast. These forecasts are referred to as h-step ahead forecasts.

These definitions are illustrated in Fig. 5.2 which demonstrates a time series on a uniform, hourly time step grid, with a forecast origin at time step $t_6 = 6$ and a forecast horizon of $h = 4$ time steps. Note that although lines have been drawn between markers (observations) for clarity, there are no observations between the time steps.

Often forecasts are written using the same lettering as the original time series but with a hat, e.g. $\hat{L}_{n+k}$. To signify the starting (or origin) point of the forecast this can also be written as $\hat{L}_{n+k|n}$ for a forecast which indicates both the forecast origin, t_n, and the time step being estimated, t_{n+k}. In this book both forms will be used and the origin and horizon should be clear from the context.

A special case of forecasts are 1-step ahead forecasts, and these are often used to compare the accuracy of different methods. They can be applied iteratively to produce h-step ahead forecasts by applying the 1-step ahead forecast h times where each new forecast value is fed back into the model for the next time-step forecast. Unsurprisingly, these are referred to as **iterative forecasts**. Alternatively, the entire

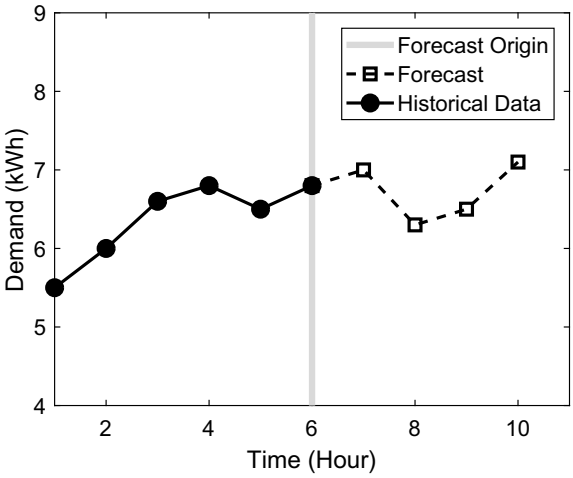


Fig. 5.2 An Illustration of a 4-step ahead forecast with historical data, and forecast origin labelled

forecast horizon can be achieved in one go, in which case such forecasts are called **direct**. Both types of forecasts will be considered in this book.

In many applications, including the core application of storage control presented in this book (Sect. 15.1), forecasts can be updated as new observations become available. This has the advantage of using the most recent data and thus improving the future estimates, especially those at shortest horizons. These are called **rolling** forecasts. Consider a h-step ahead forecast with forecast origin t_n, with estimates $\hat{L}_{n+1|n}, \hat{L}_{n+2|n}, \ldots, \hat{L}_{n+h|n}$. When a new observation becomes available at t_{n+1} the forecast model can be retrained on the updated dataset to produce a new estimate $\hat{L}_{n+2|n+1}, \hat{L}_{n+3|n+1}, \ldots, \hat{L}_{n+h+1|n+1}$. Since more recent information is now incorporated into the model, the forecasts at $t_{n+2}, \ldots, t_{n+h}$ should be more accurately estimated than the previous forecast. The forecast horizon is a **moving window** of width h. An example of a rolling forecast for the same situation presented in Fig. 5.2 in shown in Fig. 5.3. A forecast is originally made at the initial forecast origin at $t = 6$ for the next four time steps ($t = 7, \ldots 10$). When a new observation is made at time $t = 7$ a new forecast can be produced at this new forecast origin for the next four time steps ($t = 8, \ldots 11$). Notice that the new forecast trajectory has now been updated given the new observation.

As suggested in Sect. 5.1, a time series is actually a function of several other factors such as weather variables, time of day, seasonalities and other, perhaps unseen, factors. The aim of the forecast is to try and approximate the function which 'accurately' describes the future behaviour of this time series. Accuracy can be a difficult term to define but is often based on error measures (these will be introduced in Chap. 7) or how much they optimises the application of interest. The forecast can be written in a functional form. The following is a general form for a 1-step ahead forecast

$$L_{n+1} = f(L_1, \ldots, L_n, Z_1, \ldots, Z_k, \boldsymbol{\beta}) + \epsilon_{n+1}, \tag{5.27}$$

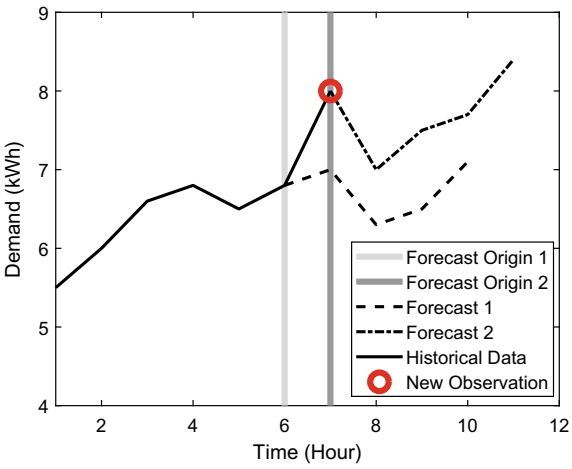


Fig. 5.3 Example of a rolling forecast updated as a new observation becomes available. A new observation is made at time step 7 at which point an updated forecast is produced with a rolling window of size $h = 4$

for some function f which generates the forecast $\hat{L}_{n+1|n}$ and is dependent on the historical data $L_1, \ldots, L_n$ and k explanatory variables $Z_1, \ldots, Z_k$ (Methods for selecting these variables will be considered in Sect. 6.2). For example, in electricity demand forecasting these explanatory variables could be weather or electricity prices. If one of the explanatory variables is a forecast, e.g. a temperature forecast, then estimates for future time steps $t_{N+1}, \ldots, t_{N+k}$ can be included in the model (although note they are still only generated prior to the current time step). It is important to note that the larger the horizon (the bigger the k), the less accurate a forecasted explanatory variable will be and hence may be less effective as a model input. This should be tested as part of the model development. Similarly one can describe a h-step ahead forecast

$$\hat{L}_{n+h|n} = f(L_1, \ldots, L_n, Z_1, \ldots, Z_k, \boldsymbol{\beta}), \tag{5.28}$$

for forecast origin n. Since a h-step ahead forecast can be produced from repeated application of a 1-step ahead forecast the inputs in many of these steps may include forecast values of L as inputs.

Every forecast model has parameters or hyperparameters (Sect. 8.2.3) which determines the response to the inputs. The parameters are represented by $\boldsymbol{\beta}$ in Eq. (5.27), and must be appropriately trained in order to produce an accurate forecast (see Sect. 8.2 for an introduction to how to train these models). As a basic example, consider a simple linear regression $ax + b$ (Sect. 9.3). In this case, the parameters are the coefficients for the model, $\boldsymbol{\beta} = (a, b)$, i.e. the trend and intercept.

There are many terms to describe the inputs, outputs and other elements of a forecast model as represented in Eq. (5.27):

- The variables within the function f, $L_1, \ldots, L_n$ and $Z_1, \ldots, Z_k$ are often known as the **predictor** or **independent** variables.

- The variable to be estimated/predicted is often called the **dependent** or **predicted** variable. For this book this almost always will be electricity demand.
- When the independent inputs are historical versions of the dependent variable, e.g. $L_1, \ldots, L_n$, then these are often referred to as **autoregressive** features.
- $\epsilon_{n+1} = L_{n+1} - f(L_1, \ldots, L_n, Z_1, \ldots, Z_k, \boldsymbol{\beta})$ are the **errors** between the actual observations and the forecast estimate. Since no forecast is ever perfect these will rarely, if ever, be zero. In time series forecasting, errors are also often called **residuals**, although sometimes this term is used to represent what is left over after fitting a model on the training set (see Sect. 8.1.3). This will be the convention typically used throughout this book.

Given any of the models which will be introduced in Chaps. 9–11 (also assume for simplicity that the hyperparameters, Sect. 8.2.3, have already been selected), the role of the forecaster is to find the 'best' version of the model (i.e. the optimal choice of function $f()$) and this will require optimising the parameters, $\boldsymbol{\beta}$, which define that model. As will be seen in Chap. 7, 'best' is often defined in terms of generalisation which is measured by minimising the errors on a test set (Sect. 8.1.3). If the forecast is used for a specific application then an appropriate error measures must be carefully chosen in order to optimise the overall performance.

As will be shown in Chaps. 9–11, there are a wide variety of forecast models each with their own advantages and disadvantages which are suited to different applications. A good forecast model will have zero mean errors because otherwise the forecast can be improved by simply shifting the current forecast model, e.g. $\hat{f}(L_1, \ldots, L_n, Z_1, \ldots, Z_k, \boldsymbol{\beta}) = f(L_1, \ldots, L_n, Z_1, \ldots, Z_k, \boldsymbol{\beta}) - b$ where $\mathbb{E}(\epsilon) = b \neq 0$ is the mean value of the errors (see Sect. 3.1 for definition of mean).

The above mainly describes forecasts in the context of **point forecasts** which only provide a single estimate for each time step $t_{n+1}, t_{n+2}, \ldots, t_{n+h}$ in the forecast horizon. This is usually in terms of some measure of centrality such as the mean or median. A more descriptive alternative is a **probabilistic forecasts** which provides multiple value for each time step and better describes the uncertainty of the future values. Methods for generating such estimates will be given in Chap. 11. Probabilistic forecasts typically take one of the following three forms:

1. **Quantile Forecast**: Here several quantiles (see Sect. 3.2 for more details on quantiles) of the future values are estimated. If two quantiles are used (a high and low) then the area between the two values is often called the **prediction interval** or **forecast interval**. The 10% and 90% quantiles are common choices. An example is shown in the top right of Fig. 5.4).
2. **Density Forecast**: For a density forecast the full continuous distribution (see Sect. 3.1) is estimated for each time step. This is illustrated in the bottom left of Fig. 5.4).

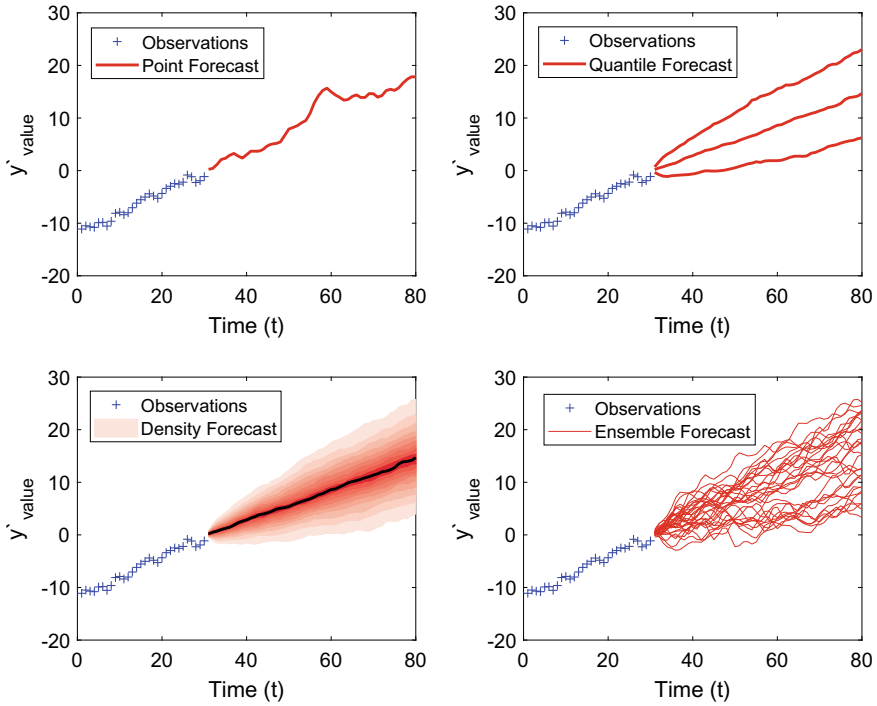


Fig. 5.4 Example of the different types of forecasts, including the three different types of probabilistic forecasts. The blue crosses are historical observations and the forecasts are in red starting at time step $t = 31$. Top left is the point forecast. Top right is a quantile forecast, showing the 0.1, 0.5 (median) and 0.9 quantiles. Bottom left is the density forecast and bottom right is the ensemble forecast

3. **Ensemble Forecast**[3]: The quantile and density forecasts only estimate a distribution at each time step $t_{n+1}, t_{n+2}, \ldots, t_{n+h}$ in the forecast horizon. In reality the time steps are often interdependent with the values at earlier time periods influencing the values at later time periods. Ensemble forecasts estimate realisations from the full joint multivariate distribution for the set of random variables $\hat{L}_{n+1}, \hat{L}_{n+2}, \ldots, \hat{L}_{n+h}$ (See Sect. 3.3 for more details on multivariate distributions). This is illustrated in the bottom right of Fig. 5.4) for 30 ensembles.

A drawback to probabilistic forecasts is the extra computational costs and the requirements for much more training data in order to generate an accurate estimate. If there is sufficient computational resource and data then probabilistic models provide

[3] Note that sometimes these types of forecasts are also called **scenario forecasts** as ensemble forecasts can be confused with ensemble methods such as Random Forest (Sect. 10.3.2). However, scenario forecasts is often used in energy systems to denote different future scenarios, e.g. high electric vehicle uptakes. We will tend to use the term ensemble forecasts to refer to realisations from a multivariate probabilistic forecasts. We will clarify when this is not clear.

a much more descriptive and informative estimation of the uncertainty in the future values.

How is this translated to short term load forecasts?

Load on an electrical circuit is often measured at regular intervals using a meter for billing and other purposes. This includes smart meters in homes and businesses. Hence energy and load is a time series and load forecasting is a specific application of time series forecasting but focused on power or energy values. Due to the seasonal nature of energy usage, applications (as will be illustrated in Chap. 15) typically have specific horizons and forecast origins. Short term forecasts, the primary topic of this book will involve horizons of a day up to a couple of weeks ahead. Similarly forecast origins are usually at the start of each day although there are many exceptions. Another particular feature of short term load forecasts are the independent variables. Energy demand in households and businesses is often driven by weather variables due to their impact on heating and ventilation. However time of day is also an important input and is often included in various ways. Probabilistic forecasts are very useful to estimate the uncertainty of load forecasting, especially at the lower voltage or household level where the demand is relatively volatile compared to the load over a nation.

5.3 Types of Forecasts

As briefly introduced in Sect. 5.2 forecasts can be classified according to whether they are iterative or direct, or as point forecasts or various forms of probabilistic forecasts. Different types and families of forecasts are desirable for different situations, applications and scenarios. Some of the most common groupings of forecasts and their features are listed below.

- **Rolling Forecast Frequency**: Rolling forecasts are updated at regular time steps (it could be every time step) and produce estimates over a horizon of fixed length. So for load forecasts this could be a day-ahead forecast which is updated at every half hour, utilising new observations as they are recorded. Alternatively the updates may only be once a day, say at midnight. The latter is still technically a rolling forecast but much less frequently updated. Those more frequently updated will give much better prediction at very short time horizons as they utilise the most recent information. However, the drawback is that they will require infrastructure in place to collect, transmit and integrate the most up-to-date information.
- **Point or Probabilistic forecasts**: A wide variety of point and probabilistic forecast models will be introduced in the following chapters. As introduced in the previous section, in contrast to a point forecast, a probabilistic forecast provides multiple values per time step to describe an estimate of the spread of the future values. Point

forecasts are quicker to generate as they have fewer parameters to learn, and require less training data. Further, they are often easier to integrate into applications, e.g. storage control models (See Sect. 15.1) since it is easy to utilise a single value per time point rather than a range of values. For volatile data, point forecasts do not describe the uncertainty in demand and hence applications utilising more volatile demand may require probabilistic forecasts. A drawback to probabilistic methods is they are much more computationally expensive to produce and require more storage. In this book, methods for creating both types of forecasts will be considered.

- **Statistical and Machine Learning Methods**: Traditionally time series forecasts have been implemented using statistical models such as ARIMA and exponential smoothing (see Sects. 9.4 and 9.2) and are easy to implement, computationally inexpensive and easy to interpret. More recently, machine learning techniques such as neural networks and random forests have become popular (see Chap. 10). Despite being more computationally expensive, they can engineer unseen features and learn complex nonlinear relationships. Statistical models can be preferable when there are clear, well understood relationships in the data, e.g. daily/weekly seasonality, or clear links to external influences such as weather. They can also be preferable when there is a relatively small amount of data since model assumptions are used to replace learning the relationships directly from the data (although of course if the model assumptions are wrong then the model will be inaccurate). Machine learning methods generally excel for complicated data with nonlinear and possibly unclear relationships (less manual feature engineering is possible). They are also preferable when learning across a large number of time series or for hierarchical time series (see below). The question of which type of model is better is ongoing. The most popular time series forecasting competition, the M-Competitions,[4] have shown in some cases that either type is preferable. More recently combinations of both types of models has shown to have the best accuracy (see Sect. 13.1 for more information on model combination).
- **Hierarchical Forecasts**: Often data are arranged in hierarchies. In power systems, as shown in Chap. 2 the distribution network is a hierarchy with electricity stepped down at substations as it is distributed to consumers. The demand increases from the individual customers up the hierarchy to the substations, all the way up to the transmission and national level. The objective of hierarchical time series forecasting is ensure that forecasts are *coherent* across the hierarchy, i.e. that forecasts at one level of the hierarchy should be coherent with the forecast at the next level of the hierarchy. Another way of saying this is that the aggregate of the forecasts should match the forecast of the aggregate. This will be discussed in more detail in Sect. 13.2.
- **Local versus Global Forecasts**: When forecasting multiple time series there are two main approaches that can be taken. You can take a local approach where you train a model on each time series, or you can take a global approach in which you fit the same model to all time series. The global approach can be preferable when

[4] See https://mofc.unic.ac.cy/the-m6-competition/.

there is a lot of time series and it would be prohibitive to produce a model for each of them. This is particularly relevant when considering smart meter forecasting. If every home in a country is to have a smart meter this is a large number of time series and therefore a global forecasting approach is preferable to a local approach. This is described in more detail in Sect. 13.4.

- **Peak Forecasts**: The above approaches have been written in the context of forecasts for an entire period of a time series (e.g. each half hour of a day or week). In fact, in many cases it is only specific features that are of interest. One of the most important aims of forecast models is to predict the peak of a demand time series over a period (typically a day). The advantages of peak forecasts is that only a single value needs to be estimated for each period although the timing may also be important. However it should be noted that there are less historical examples of peaks and since they are, by definition, extreme values they may be trickier to accurately predict than baseload demand. Furthermore, for volatile demand, such as household smart meter data (see Sect. 13.3), the timing of peaks may be very irregular.

5.4 Notation

Some basic notation of time series and time series forecasts were introduced in Sects. 5.1 and 5.2. Here some of the most important notation used throughout this book are reiterated and expanded on in the context of load forecasting:

- The actual monitored electricity demand will be modelled as a time series, $L_1, L_2, \ldots$, of real numbers with L_n representing the demand at the nth time step t_n. L_1 represents the oldest data point in the data set. Unless otherwise stated the time steps are uniformly spaced, i.e. have the same time difference between one time step and the next, $t_{n+1} - t_n = \Delta t \ \forall n$. For load data, if not stated otherwise, we report the average load over the respective interval in kilowatts, denoted kW.
- Forecasts are denoted as another time series, $\hat{L}_n$, with a hat indicating that this is an estimate of the true demand at time step t_n.
- The notation $\hat{L}_{N+k|N}$ will often be used to indicate that the forecast is for the time step $N + k$ and has been generated starting from the forecast origin at time N, for forecast horizon of length k time steps. However, this notation can be a bit cumbersome and hence is omitted and simply written $\hat{L}_{N+k}$ when the forecast origin is obvious.
- Explanatory time series, for example temperature, will be denoted by another capital letters, e.g. X_t. If there are more than one explanatory variable, for example if using multiple weather variables, then another index will be used to indicate the different variables. For example, given M explanatory variables they can be denoted, $X_{1,t}, X_{2,t} \ldots, X_{M,t}$ for their value at time t. Alternatively different letters may also be used for each individual time series.

5.5 Questions

Some of the following questions will require using some demand data. A list of possible resources is listed in the Appendix D.4.

1. List some other types of time series that you can think of. This can be anything not necessarily energy demand related. What is the range of values that the series can take?
2. Download a demand time series. Is there any trends or seasonality in the data? If you have several time series compare them, do some have different types of seasonality? How many different seasonalities can you see? Is there a difference between the weekday demand and the weekend demand? Are there any other patterns you can see in the data?
3. From the data listed in Appendix D.4. Take some aggregated state level demand (GEFCOM 2014) and household level demand (e.g. the Low Carbon London data set). Plot the data. Compare some of the features: What is the average size of the demand, when are the peaks in the demand? Are there several peaks in a day? When do they typically occur? Do the daily peaks vary much from one day to the next?
4. Generate simple rolling forecasts. Consider a half hourly time series. Create a simple day ahead forecast for the following day by using the previous day as the forecast for the following day (i.e. a 48 half hour shift). For example, to predict Tuesday, use the previous Monday's values. Consider the difference between the actuals and the forecast (see Sect. 7.1). Now create a basic half hour ahead rolling forecast for each half hour of the day by using the previous half hour as a forecast (i.e. a half hour shift of the data). Try this with some of the time series from the GEFCOM 2014 data and some household data (say from the Low Carbon London dataset). Are the errors smaller or bigger than the day ahead forecast? How do the absolute errors compare between the GEFCOM and household data? What about the relative errors?

Chapter 6
Load Data: Preparation, Analysis and Feature Generation

Chapter 5 introduced the general definition of a forecast and many of the concepts and categorisation of the different types and features of time series forecasts. To develop an appropriate model requires identifying genuine patterns and relationships in the time series data. This requires a detailed investigation and analysis of the data, since selecting the correct input features is, arguably, at least as important as selecting the most appropriate forecast model. This data analysis and feature generation is the main focus of this chapter. However, prior to this it is important to understand whether the data is of sufficient quality to allow the training of a good forecast model. The next section begins by considering important features of high quality data and potential preprocessing which may be required. This is followed by methods for analysing the load data and identifying features which may be useful inputs to a forecast model.

6.1 Preparation and Pre-processing

Before training the models, the data must be assessed and cleaned, otherwise the forecasts will be trained on flawed data and the outputs will be inaccurate, misleading, or meaningless. As the machine learning mantra succinctly puts it 'garbage in, garbage out'. Unlike contrived data often used in textbook examples to demonstrate techniques, real data is messy, has little-to-no formatting, and is rarely error free.

Some of the most common data issues are:

1. **Missing values**—Due to errors in communications or faults in the monitoring equipment, recorded data is rarely complete.
2. **Extreme values**—These can be excessively large, or excessively small values. This is particularly tricky to determine, especially for highly volatile data like the low voltage energy demand analysed in this book. Some extreme values can be identified since they are outside the parameters of the system, e.g. exceeding the circuit breaker limits of a house and therefore technically not possible (unless something is wrong with the circuit breaker of course!). However, for the majority

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_6

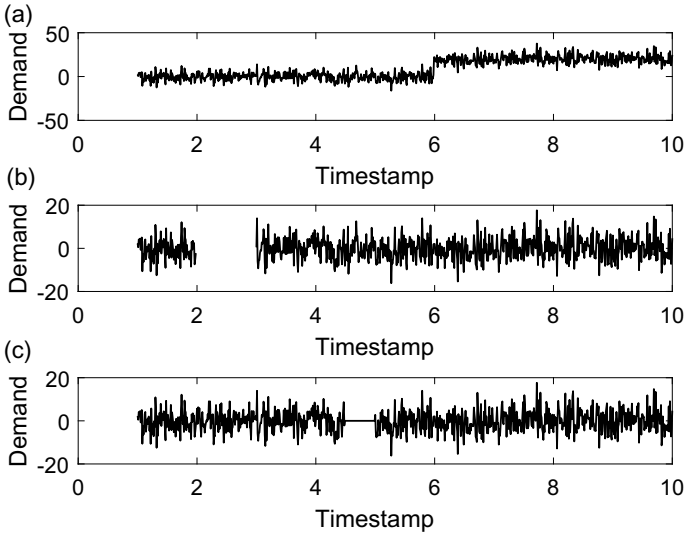


Fig. 6.1 Examples of time series plots with **a** a change in the level of demand, **b** with missing data, and **c** with a section of constant values

of the time, identification is tricky since it is often not possible to confirm whether a value is valid or has been recorded incorrectly.

3. **Anomalous values**—Depending on the application, some values are clearly incorrect. For example on an LV network feeder with no generation all the demand should be positive, with no values smaller than zero. Hence negative demand is clearly impossible and, on feeders with large demand, recorded values of zero should be treated with suspicion.

Examples of some time series with possible anomalous and erroneous data is shown in Fig. 6.1 for difference situations. Plot (a) shows a demand level increase at a particular point in the time series. This could be a fix to the monitoring equipment, or could be a genuine change caused by, say, the occupants installing a new high demand appliance (like a heat pump or electric vehicle) which causes an overall increase in the baseline demand (also referred to as concept shift, Sect. 13.6.3). Plot (b) shows an example of when data is missing from the time series data. Sometimes data is missing for only occasional points, or like in this example it can be over a long period of time. The latter can occur when monitoring equipment experiences a fault. Finally plot (c) shows a section of constant values. Again these can be caused by sensor equipment faults, however they are much harder to detect, especially if they only occur over short periods, since they may not be obvious from basic analysis or simple time series plots.

The impact of these erroneous values can have a detrimental effect on the quality of the models and the accuracy of the corresponding forecasts. For this reason it is important that their effect is mitigated or removed which is typically done by

deleting them from the data set. However, removing values that are not erroneous could reduce the accuracy of the forecasts. This is especially true when the aim is to accurately forecast extreme values such as peaks. Further, when producing probabilistic forecasts it could mean that the tails of the distribution are not properly calibrated.

The topic of data pre-processing is a complex area in its own right and much of it is beyond the scope of this book. Some extra references are included to more advanced techniques in Appendix D.1. For this book, it is sufficient to concentrate on some simple but common methods for identifying and cleaning up time series data.

6.1.1 Outlier Identification

Missing values are obviously easy to identify and anomalous values will also be simple to check for. For that reason the focus is on identifying outliers which in our case will be those values which are unusually large or small. Visual methods are a common way to determine which points are outliers, but this can be considered a little subjective and hence could lead to biases. For stationary time series a more systematic identification is to identify those points which are further from the central value of the data than would be expected given the spread of the sample of the data. One of the most common ways of doing this is to identify those points which are a few sample standard deviations from the sample mean (see Sect. 3.5). Recall for normally distributed data, 95% of values are within 2 standard deviations of the mean, and 99% of values are withing 3 standard deviations. However, often the assumption of data following a normal distribution is not valid and hence the true distribution of the data is unknown. Despite this, the standard deviation approach can often be a useful measure to understanding which points may be outliers but care should be taken if the data is skewed and not symmetrical. For data which is not normally distributed a more robust, but less commonly used way, is to consider how many interquartile ranges the data is from the sample median (Sect. 3.2). More generally estimating which quantiles the data lies in can also identify outliers. The quantiles are often more robust to outliers than the standard deviation approach and therefore can be preferable for defining thresholds in the data.

An illustration of using two standard deviations from the mean to detect large values is shown in Fig. 6.2. In this relatively contrived example, three points are clearly outliers and these have been successfully identified by using the standard deviation criteria.

The process is much more complicated when the data is not stationary. When there are simple and obvious trends in the time series, but the variance is fixed over time, then the standard deviation can be calculated from the detrended series. The detrended series is simply the difference between the observations and a model fit, i.e. the model residuals (Sect. 5.2). Two examples are shown in Fig. 6.3 for one series with linearly increasing trend and one with a simply single periodic behaviour. Also included are the lines of best fit as well as the lines indicating distances of two standard

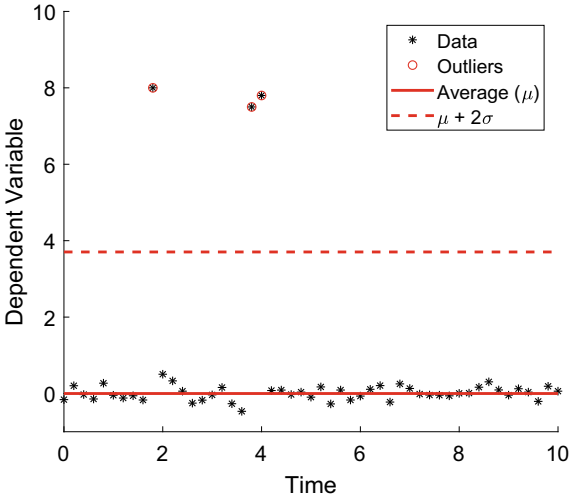


Fig. 6.2 Example of a stationary time series (black markers) with three outliers (red circles). Also included is the average value (solid red line) and the average plus two standard deviations of the points. Any points further than two standard deviations from the mean are labelled as outliers

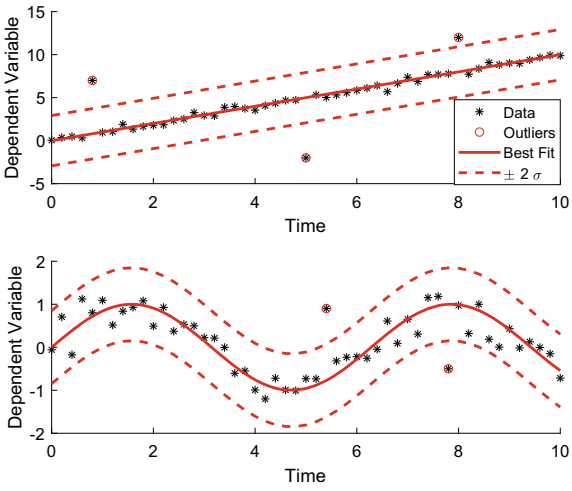


Fig. 6.3 Examples of non-stationary time series (black markers) with outliers (red circles) for a time series with linear trend (top) and one with seasonality (bottom). Also included are the lines of best fit for the data

deviations of the residual series from the line of best fit. Without the detrending it would not be clear that some of these points are outliers since they are within the range of the full data set. A complication with this approach is that the trend fit may be poor because of training on data with outliers/anomalous values, especially if there are a lot of them relative to the full dataset. In that case the extreme values may not register as outliers.

For more general time series it is not possible to easily detrend in order to identify outliers and may not be advisable as it requires making assumption about the underlying process. Hence the forecaster may include some of their biases in the model assumptions and incorrectly label some normal values as outliers. One approach is

to develop a forecast model and compare the performance with, and without, the assumed outliers and analyse their effect on the accuracy. There are other more sophisticated models for identifying outliers beyond the scope of this book, some of which can be found in the further reading in Appendix D.1.

6.1.2 Imputation

After identifying anomalous and outlier data a decision must be made as to whether to remove the values or not. If the values are known to be incorrect then they should definitely be removed from the data set. Otherwise if it cannot be confirmed whether a value is truly an outlier or incorrect value then it is recommended to keep the value in. One possibility is to run the models both with and without the anomalous values to see if the forecast is sensitive to the changes. If there are only a few anomalous values then their inclusion may have little effect on the overall model accuracy anyway. In some cases there may be a large amount of missing data, in which case it may be impossible to produce any good model with decent accuracy. What constitutes "enough" data depends on the type of data, the application and other design elements such as forecast horizon. For short term forecasts (say one day ahead), sometimes a reasonable benchmark forecast can be produced using only four to five weeks of data (See the Case Study in Chap. 14).

When data is missing, or has been removed due to cleaning actions, there will be gaps in the dataset. This causes a number of issues. Firstly, it makes data handling more complicated as techniques must be considered which ignore missing instances, and secondly, it produces potential biases in the data since certain features may be more prominent in the reduced dataset then they otherwise would be. If the missing data is relatively random then there is very little bias introduced and the forecasts can be trained on the reduced data without concern that the models will be skewed by any biases.

An alternative method for dealing with missing values is to insert or *impute* other values. The process is known as **imputation**. This simplifies analysis and model training on the data and there are several different ways to choose the values to insert:

- **Simple Average**—this maintains the sample mean of the data but ignores any trends or seasonalities. A moving average (over a moving window around the data) can be used to better fit any trends.
- **Last value**—this retains the trends in the data and means that the missing values are filled with recent values. However, this ignores any other patterns which may be in the data. It is worth noting that many data acquisition systems used to collect power data often forward fill at regular intervals if they stop receiving data.
- **Seasonal Average**—if the data has seasonality then this retains those features. For example, if the data has daily seasonality then a missing value at 4pm could be filled by using an average of the values at 4pm from the previous days.

- **Regression**—Regression is considered in Chap. 9 but essentially this means using a weighted average of surrounding values to fill the missing data. This allows more complex relationships to be included to impute missing values.
- **Interpolation**—More generally a curve could be fitted (e.g. a polynomial) to the surrounding values of the missing data point and the value on the curve at the missing point can be used to impute the value (See an example of interpolation in Sect. 9.6).

6.1.3 Normalisation and Transformations

Even after cleaning the data, in it's raw form, the data may not be suitable for using directly within a model. For example, for a linear regression model (Sect. 9.3) there are assumptions that the errors follow a Gaussian distribution. This is unlikely to be always true, especially for smart meter data which in some cases has been shown to follow a lognormal distribution (see Sect. 3.1). Further, notice that unless there are reverse power flows at the meter, say due to solar PV generation, then smart meter demand should always be nonnegative (i.e. positive or zero) which is not true for Gaussian distributed data, but will be for a lognormal distribution. Recall (Sect. 3.1) a random variable z has a lognormal distribution if it has PDF of the form

$$f(z) = \frac{1}{z\sqrt{2\pi}\sigma} \exp\left(\frac{-(\ln(z) - \mu)^2}{\sigma^2}\right). \tag{6.29}$$

By definition this just means that the transformed variable e.g.

$$x = (\ln(z) - \mu)/\sigma \tag{6.30}$$

has a Gaussian/Normal distribution. In other words if data is nonnegative, has one long positively skewed tail, applying a lognormal transformation may produce normally distributed data. This in turn may be easier to manipulate and utilise. After training a model on this data an inverse transform can be applied to the forecasts to obtain physically representative data. An example of lognormally distributed data ($\mu = 0, \sigma = 0.5$) is shown in the histogram in Fig. 6.4 (left), together with the same data but log-transformed (right). Notice the lognormal transformed data is now symmetric and bell-shaped as expected and now allows negative values.

Another common transformation applied to raw data is **normalisation**, where the data is scaled prior to visualisation and/or modelling development. This can have two main advantages. Firstly, consider a situation where more than one variables is being modelled, but they have very different ranges of values, say one is bound between 0 and 10 and another is between 0 and 1000. In this case, it can be very difficult to visualise or understand the relationships between them due to the extreme difference in their relative variations. Scaling these values can better highlight these relationships.

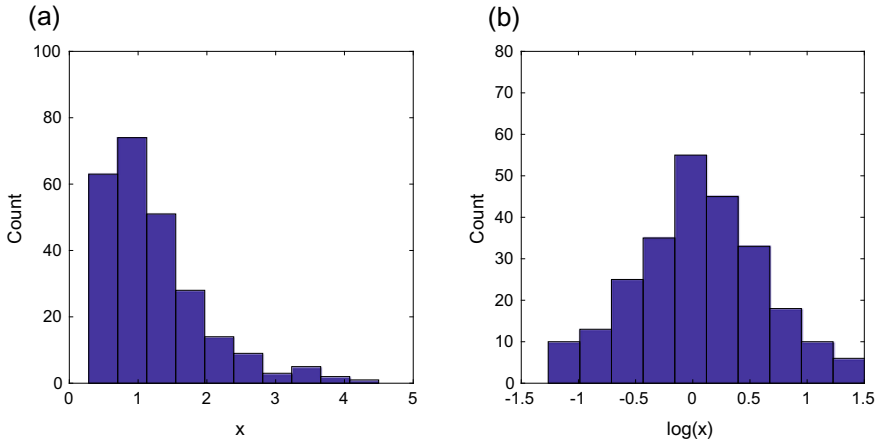


Fig. 6.4 Example of the distribution of a sample of data **a** with a lognormal distribution and then **b** the same data but transformed using the log function

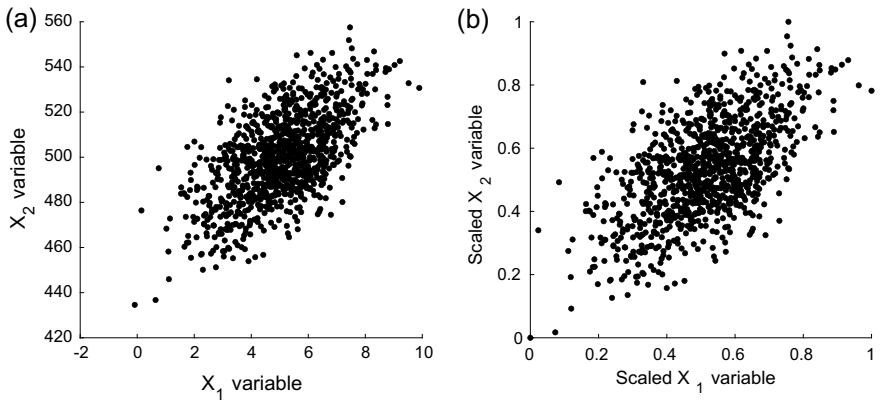


Fig. 6.5 Examples of two variables from a bivariate distribution (**a**) original variables (**b**) the same data but each variable has been scaled to be between 0 and 1

The second major reason for scaling is to help with training the parameters of a model (see Chap. 8 on training). The scale of the data may effect the scale of the parameters. By normalising the data, this restricts the range of the data, and reduces the search space for the optimal parameters. A simple two variable example is shown in Fig. 6.5 where the scatter plot of the original variables is shown in (a) and the rescaled variables are in (b). The scaling has been performed so that each variable is between 0 and 1. In the original data the spread of X_1 is between -2 and 10, in contrast the X_2 variable is between 400 and 500, hence has bigger magnitude and spread. If we were training a linear model, e.g. $y = aX_1 + bX_2$ on this data we would see that b would be relatively small compared to a otherwise the responses to changes

in the dependent variables (within their respective ranges) would produce wildly different responses y. It also means that the coefficients therefore have different search ranges. In contrast the normalised variables would reduce the search space for the linear coefficients a, b. Note that the relationships between the variables will still be preserved by the normalisation albeit scaled.

As will be shown below, there are several ways to normalise the data, but an important requirement is that the data can be rescaled back to its original size. One of the most common forms is the **Min-Max Scaler** which transforms the variables into the interval [0, 1]. Assuming that the time series has a maximum and minimum value given by $x_{\max}$ and $x_{\min}$ respectively, then the scaled version of any data point x is given by

$$\hat{x} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}, \tag{6.31}$$

and thus the series transforms to one which has the maximum value of one, and the minimum of zero. Note if you have an extreme outlier then $x_{\max}$ and $x_{\min}$ may produce an unsuitable normalisation. In this case it may be worth cleaning/preprocessing the data before performing the scaling (see Sect. 6.1.1). To recover the original values you simply rearrange the calculation:

$$x = x_{\min} + \hat{x}(x_{\max} - x_{\min}). \tag{6.32}$$

Another common methods is the **Standard Scaler** which subtracts the mean of the series, μ, and divides by the standard deviation, σ,

$$\hat{x} = \frac{x - \mu}{\sigma}. \tag{6.33}$$

The new series has mean zero and a unit variance and the values are no longer constrained within [0, 1]. Note that the mean may be strongly effected by outliers so as with the Min-max scaler it may be worth removing them (Sect. 6.1.1) before proceeding with the normalisation.

To avoid the effect of outliers there are other normalisations such as the **Robust Scaler** which uses the median x_{50}, and the interquartile range $x_{75} - x_{25}$ which is the difference between the 25th and 75th percentile (See Sect. 3.2 for further details on percentiles/quantiles). The scaled variables are given by

$$\hat{x} = \frac{x - x_{50}}{x_{75} - x_{25}}. \tag{6.34}$$

This has median zero but note that there may still be outliers in the transformed series.

Now consider how to fit a model to scaled data in a simple example. Consider the linear model

$$y = ax + b, \tag{6.35}$$

for variables x, y and coefficients $a, b \in \mathbb{R}$. Suppose we scale the variables x, y to $\hat{x}$, and $\hat{y}$ respectively. The parameters $\hat{a}, \hat{b}$ are found for the scaled model

$$\hat{y} = \hat{a}\hat{x} + \hat{b}, \tag{6.36}$$

and by substituting the rescaling (e.g. Eq. (6.32)) you can estimate the original simple model. This case is much simpler than others as the relationships are all linear. The transformation may not be possible for other more complicated cases.

6.1.4 Other Pre-processing

As well as dealing with missing and anomalous data there is also some standard pre-processing procedures which also should be considered.

Firstly, many time series forecasting methods depend on the values being spaced equally in time. This is often the case as much monitoring equipment is calibrated to record at uniform intervals. However, if the data is not at uniform intervals the data can be estimated at these time steps by interpolating to the time steps of interest. This obviously creates extra errors (in this case errors from the estimation) but they will be smaller the higher the resolution of the original data, and the smaller the volatility of the data. Unfortunately, there is few techniques and packages for dealing directly with time series data which is not recorded at uniformly spaced intervals. However, since most energy monitoring is either high resolution or is designed for regular intervals the rest of the book will assume the data is equally spaced in time with negligible interpolation errors.

Another common issue is the fact that different input data is defined at different temporal resolutions. For example the load data may be recorded at half hourly intervals but the corresponding temperature data may be at hourly resolution. If the temperature variables are important explanatory input variables for a load forecast then it may be worth resolving both data sets to the common resolution of hourly data.[1] For energy data (kWh) this essentially means summing the data over the two half hours, whereas for average Power (kW) data this would require averaging over the two half hour points. The latter (Power) is the more common representation of load data.

6.2 Feature Selection and Engineering

One of the most important factors for creating an accurate forecast is choosing the most appropriate features to include in the model. In many cases the features are more important than the forecast model used. One option is to include all the

[1] Alternatively the temperature data could be interpolated to half hourly.

available features and fit a model which penalises the number of parameters used in the model, such as information criteria and regularisation techniques (Sects. 8.2.2 and 8.2.4). These methods can be used to select variables and create a parsimonious model. In this section several methods will be considered for identifying potentially important relationships between the dependent and independent variables.

As will be discussed in Sect. 8.1 the aim for choosing features is to achieve a bias-variance trade-off. In other words, to try and include all the important features that describe the data, but not too many that the model will end up overfitting. If the number of potential features to include in the model is large then it may be worth considering the information criteria and regularisation techniques mentioned above to reduce the features to the most important ones.

6.2.1 Domain Knowledge

Some variables can be automatically selected based on the domain of interest. For example, if trying to forecast ice-cream sales it would be reasonable to suspect that the outside temperature is a strong determining factor. Similarly when considering residential households energy usage it would be sensible to assume that the demand would be related to time of day and day of the week due to typical behavioural patterns (However, note it is also easy to find households, such as shift workers, who probably won't neatly fit this assumption). In each application there are some strong candidates which, if available, could be included as features in the forecast model. At the very least, further investigation should be applied, e.g. using the visualisation techniques presented in Sect. 6.2.2.

If the obvious candidates are not readily available or cannot be measured, then **proxy values** could also be considered. These are values which are closely correlated to the value considered. For example, the weather data may not be available in the exact location of interest but may be available from an adjacent town. Another real life example of proxy values is where scientists use ice core and tree ring data as a proxy for the past climate.

6.2.2 Visual Analysis

Visualisation is essential for better understanding the data and determining potential features to include in your models. They can also be used to confirm (or deny) relationships that the forecaster anticipates would be useful, or discover entirely new relationships.

The most obvious visualisation for time series data is to simply plot the data against time, unsurprisingly this is called a **time series plot**. Many examples have already been shown including Fig. 5.1. These examples show some simple features

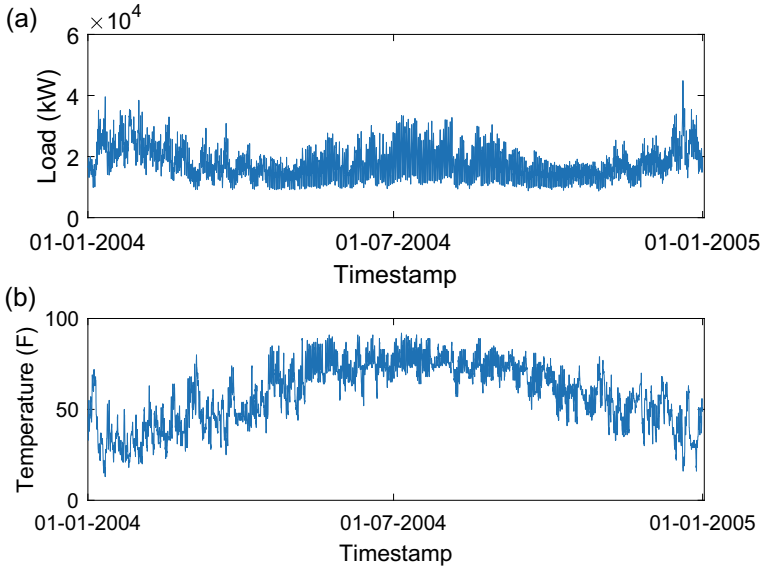


Fig. 6.6 Example of hourly time series of **a** electricity demand in kW, and **b** temperature in Fahrenheit for the whole year of 2004 in the GEFCOM 2012 data set [1]

which are readily identified from a plot including whether the data is stationary (does the distribution of data change in time), and any seasonal or linear trends.

Figure 6.6 shows an example of real, hourly, electricity demand (for one zone for an American Utility) using one year of the Global Energy Forecasting Competition 2012 data (GEFCOM 2012)[2] and the temperature from a nearby weather station (for the data and more details on GEFCOM 2012 see [1]). Instantly evident is the annual seasonalities in both the demand and the temperature values. The demand has high values at the start, middle and end of the year, likely due to the use of increased heating in the winter periods and increased use of air-conditioning in the summer months. It is clear then that the temperature and the demand time series are correlated with each other.

The relationship between temperature and demand is more easily visualised through a **scatter plot** which plots one variable against the other, with each point corresponding to each hourly period. This is shown in Fig. 6.7. In this form the relationship between the variables is much clearer and other characteristics are evident. For example, it can now be seen that, for temperatures less than 50 °F, the demand increases as the temperature decreases but at a much slower rate than the increase in demands with increases in temperatures above 50 °F. Using the extra information which the scatter plot has revealed, the shape of the relationship can be used to develop more accurate forecast models.

[2] Available from http://blog.drhongtao.com/2016/07/gefcom2012-load-forecasting-data.html.

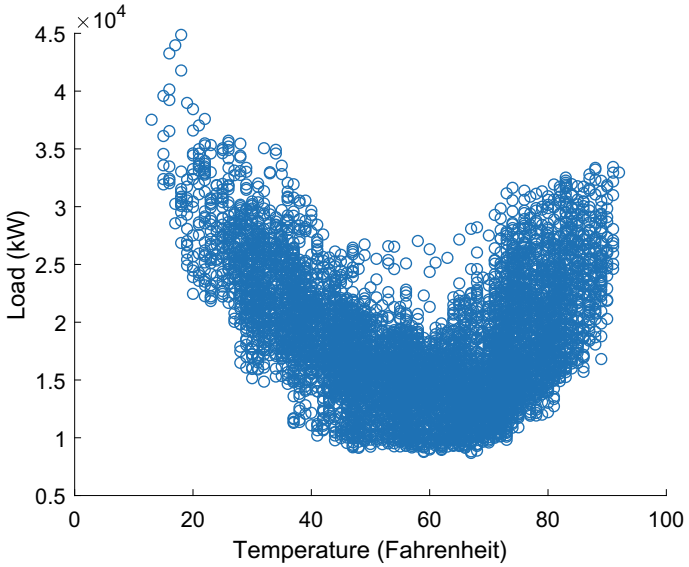


Fig. 6.7 Scatter plot of the hourly load versus the hourly temperature for 2004 in the GEFCOM 2012 data [1]

If there are several variables then it can be cumbersome to produce scatter plots for all the different relationships between the different variables. A more concise representation is a **pair plot** (also known as a scatter plot matrix). This is just a matrix of scatter plots which compares the relationship between each pair of variables. An example of a scatter plot is shown for simulated data in Fig. 6.8 for three variables. Row $k \in \{1, 2, 3\}$, column $j \in \{1, 2, 3\}$ represents the scatter plot for kth variable against jth variable. Notice that row j and column k displays the same information, just reflected, since the plots are reflected across the diagonal of the scatter matrix. Often, as in this example, the kth diagonal contains a histogram of the kth variable. In other words it shows an estimate of the marginal distribution of this variable (see Sect. 3.3 for the definition of a marginal distribution).

It is worth bearing in mind throughout this section and Sect. 6.2.4 that although variables are correlated this does not mean there is a causal link between them (the adage "correlation doesn't imply causation") but this also doesn't mean that features are not useful for the purposes of forecasting, this is known as *Granger causality*.

6.2.3 Assessing Linear Relationships

The scatter plots in Sect. 6.2.2 can suggest different relationships between variables. The Pearson correlation $Corr(X, Y)$ for two random variables X and Y is defined as

$$Corr(X, Y) = \frac{\mathbb{E}(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))}{\sigma_X \sigma_Y}, \tag{6.37}$$

and is a useful measure of the linear correlation between them. Here σ_X and σ_Y are the standard deviations of X and Y respectively, and $\mathbb{E}(X)$, $\mathbb{E}(Y)$ are the corresponding expected values (see Sect. 3.3) for more details). The value ranges from -1 to 1. A value of 1 means the values are perfectly linearly align and positively correlated, i.e. the increase in X will correspond to a linear increase in Y. For a value of -1 the values are again perfectly linearly aligned but this time negatively correlated, so the increase in one variable will simultaneously correspond to a linear decrease in the other variable, and vice versa. Values inbetween indicate less correlation, with $Corr(X, Y) = 0$ indicating no correlation at all.

The lines of best fit and the correlation coefficients are shown for each pair of variables in Fig. 6.8. In this case it is clear that variables $X1$ and $X2$ have very little correlation (0.05), as do variables $X1$ and $X3$ (0.01). In contrast variables $X2$ and $X3$ are strongly positively correlated (0.85).

If several predictors are highly correlated with each other then there can be difficulty in understanding their individual effects on the dependent variable. It may mean that the model is splitting the importance of each variable (e.g. via its trained coefficient) in a way which would be very different if only one of the variables was included in the forecast model. The importance of a variable could be underestimated when included in a model with a variable for which it is highly correlated as its influence may appear minimal. In theses cases it may be worthwhile testing

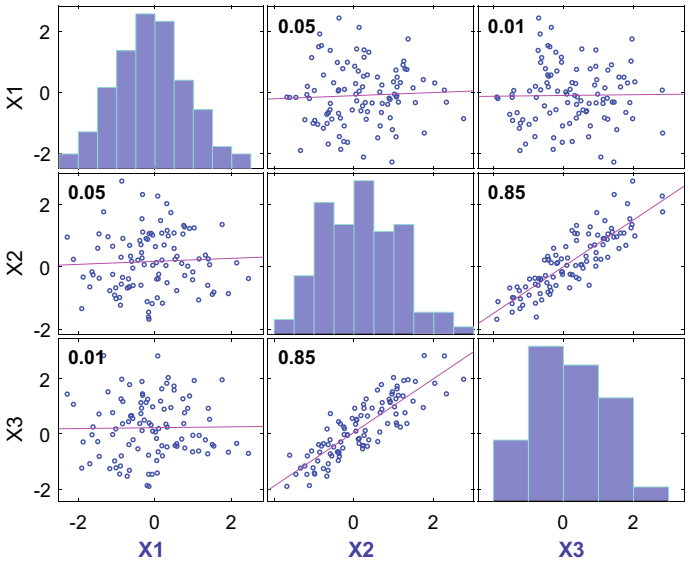


Fig. 6.8 Scatter plot matrix example for three variables. Also included is the line of best fit and their pearson correlation coefficient (see Sect. 6.2.3)

models with different combinations of the correlated inputs to see their effect on the forecast accuracy. The effect of collinearity is discussed further in Sect. 13.6.1.

The Pearson correlation is limited to linear relationships and therefore is not useful for measuring the potential of nonlinear models for the relationships between two variables. For example, it is clear that the relationship between load and temperature is not linear in Fig. 6.7. A common way to test the descriptive quality of a model between variables is the so-called **coefficient of determination** or R^2 value (R squared), which describes how much a model (for example a line) describes the data, and is defined by:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{k=1}^{N} r_k^2}{\sum_{k=1}^{N}(Y_k - \bar{Y})^2}, \tag{6.38}$$

where $\bar{Y} = (1/N)(\sum_{k=1}^{N} Y_k)$ is the mean of the observations, r_k are the residuals between the model and the observations, $SS_{tot} = \sum_{k=1}^{N}(Y_k - \bar{Y})^2$ is the total sum of squares difference between the observations and the mean, and $SS_{res} = \sum_{k=1}^{N} r_k^2$ is the sum of square residuals. R^2 typically takes values between zero and one (but can take negative values when the model is worse than the mean estimate), with the best case $R^2 = 1$ since the model would perfectly fit the observations and hence $SS_{res} = 0$. Explanatory variables with larger R^2 value can be considered more important for describing the dependent variables than those with smaller values. The value of the coefficient of determination can be interpreted as how much of the variation in the dependent variable is captured by the model, so for example, an $R^2 = 0.75$ indicates 75% of the variation is explained by the model. Care must be taken when comparing different models. Better fits (and thus larger R^2 values can usually be achieved by increasing the number of parameters, hence models with different numbers of inputs cannot be compared with the traditional R^2 values. Instead an adjusted R-squared is often used which still compares how much of the variation is captured by the models but also controls for their complexity (the different numbers of parameters). The adjusted coefficient of determination is defined as

$$Adj\,R^2 = 1 - (1 - R^2)\frac{N - 1}{N - p - 1}, \tag{6.39}$$

where p is the number of independent variables in the model (excluding any constant term in the model). The adjusted R^2 value is always less than or equal to the R^2 value, Suppose a new parameter is added to the model, then the adjusted coefficient of determination increases if the improvement in R^2 is more than would be expected by chance. Note, that a good R-squared may be achieved by simply overfitting the data but this doesn't mean the forecasts will be accurate (Recall Sect. 8.1.2), and the adjusted R^2 helps to mitigate against this.

6.2.4 Temporal Correlation Analysis

The previous sections have compared at least one variable against another variable. However, in time series, the future values frequently depend on their historical values. The **autocorrelation** function (ACF) can be used to assess these temporal interdependencies. The autocorrelation simply calculates the correlation between the time series and a shifted (or lagged) version of itself. For a time series $(L_1, L_2, \ldots, L_N)^T$ the ACF at lag k is defined as

$$\rho(k) = \frac{1}{N\sigma^2} \sum_{i=1}^{N-k} (L_i - \mu)(L_{i+k} - \mu) = \frac{R(k)}{R(0)} = \frac{R(k)}{\sigma^2}, \tag{6.40}$$

where μ is the sample mean and σ^2 is the sample variance for the time series (see Sect. 3.5 for more details). The important lags can be found by examining an autocorrelation plot, which plots the autocorrelation as a function of the number of lags. The autocorrelation plot for the GEFCOM 2014 hourly demand data is shown in Fig. 6.9. It is clear that there is strong daily seasonalities in the data given the cyclical nature of the ACF and the bigger peaks at lags of multiples of 24 h.

A drawback of the autocorrelation function is that values at shorter lags contribute to the value of the autocorrelation function at longer lags. The partial autocorrelation

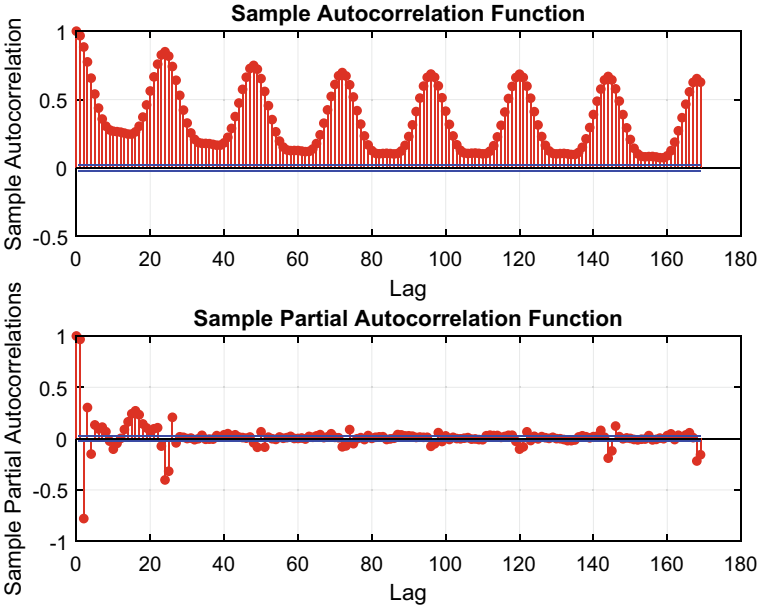


Fig. 6.9 Example of autocorrelation (top) and partial autocorrelation (bottom) for the hourly load data from GEFCOM 2014 [1]

function (PACF) at lag k can reduce this effect by removing the effects of the lags at $1, \ldots, k-1$ (see Sect. 3.5 for the formal definition). In Fig. 6.9 the partial autocorrelation for the GEFCOM load series is also shown in the bottom plot. Notice now, that the sizes of the PACF values are much lower at the daily lags of 48, 72, ... but the value increases slightly at lag 168, which corresponds to a full week and indicates weekly correlations in the time series. This means that the lag at hour 24 is still significant, as is the weekly lag, but other daily lags i.e. at two, three, four days previous etc. are perhaps less significant since the correlation shown in the PACF is much weaker.

Large autocorrelations or partial autocorrelations at particular lags can inform which historical data to include in a forecast model. For example, if, as above, there is some relatively strong correlations at weekly lags it is worth considering including the data from previous weeks from the same time period of day, in the forecast model. Historical inputs from the same time series to a model are often called **autoregressive components**. As will be shown in Sect. 9.4, the ACF and PACF plots play an important role in the parameter selection of ARIMA models.

As shown in Sect. 3.5 we can also consider the **cross-correlation** between two separate time series. This not only shows the correlation between them, but also the correlation between lags of the two series. For example, although heating demand may be driven by cold temperature, there may be a delayed response (especially in homes with good insulation) and hence it may be important to include the temperature as input together with the values from a few time steps previously. Consideration of the cross-correlation (Sect. 3.5) can be used like the autocorrelation plots to identify significant lags to include within your model. An example between the temperature and load data for the GEFCOM data is shown in Fig. 6.10. The x-axis shows the lags (up to a 24 h either way) between the series. Notice they are allowed to be negative here since either series can be lagged, the negative lag means the correlation is between lagged (historical) values of the temperature against the load without lag. First notice the cross correlation is negative but not too strong. in fact it would be much more negative if comparisons had been made between the series in Winter, and similarly been positive in Summer, due to the heating versus air-conditioning patterns as shown in Fig. 6.7. However, since the cross correlation measure linear correlation the coefficient values are much weaker due to conflating the positive and negative correlations. Another thing to notice is that the values are not symmetric around zero lag. This is because there is likely a delay between the effect of temperature on the overall load.

6.2.5 Basic Functions as Features

The feature selection methods above have largely required manual investigation and visual analysis. This can be quite time consuming and impractical for forecasting large numbers of time series. A more automatic way to train a model is to build it from individual components, which will be called *basis functions*. This is particularly

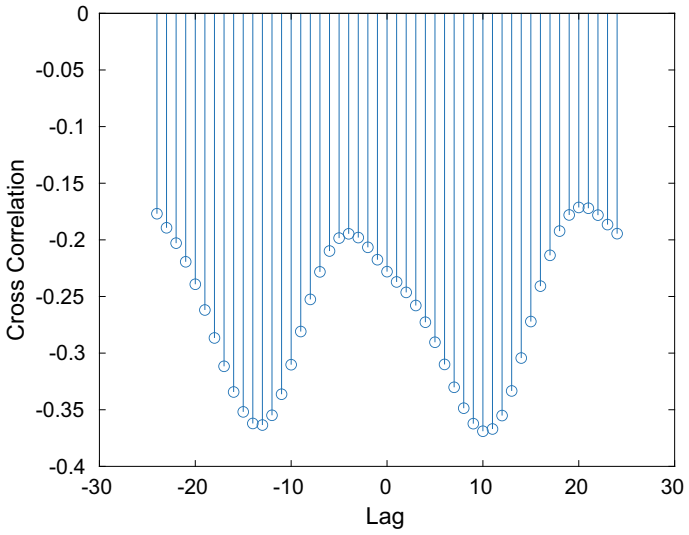


Fig. 6.10 Cross correlation coefficients between hourly temperature and hourly load for 2004 in the GEFCOM 2012 data [1]

useful for time series exhibiting periodic behaviour as is the case with energy demand time series. In other words a time series (or segment of a time series) Y_t can be written as a linear combination of simple functions/vectors $\phi_k(t)$,

$$Y_t = \sum_{k=1}^{\infty} \alpha_k \phi_k(t). \tag{6.41}$$

Where α_k are the coefficients that must be found. The most common example of this is the Fourier Series which has periodic basis functions of the form $\sin\left(\frac{2\pi kt}{K}\right)$ and $\cos\left(\frac{2\pi kt}{K}\right)$ for $t \in [-K, K]$.

Using basis functions as features means that each time series can be trained whilst reducing the development of bespoke features for each time series. Instead of the infinite sum in Eq. (6.41), in practice a finite sum is chosen, however as mentioned in the previous section, this could cause overfitting of the time series to the training data and thus must be carefully chosen. Methods for choosing an appropriate number of terms, are given in Sect. 8.2.2. A special case of basis features, using splines is described for generalised additive models in Sect. 9.6.

6.2.6 Common Features in Load Forecasting

Load is comprised of several different types of consumers, from residential, to small-to-medium enterprises (also called SMEs) such as hairdressers, small shops, etc. There is also larger commercial consumers (larger supermarkets, schools, etc.), and finally industrial consumers which usual comprise of larger demands such as steel production and other heavy industries. Forecasts may be required for the demand of these individual customers or for the load at the substations, or over larger areas and therefore consist of aggregations of these individuals consumers and other connections. These other connections may be anything from street-lighting, but also distributed generation (e.g. solar farms or wind turbines). Therefore there is no simple set of features which model all types of demand. However, there may be some features which are common, or at the very least, worth testing to identify whether they would make useful explanatory variables. This section will discuss some common ones.

First, it is worth mentioning that even the same type of consumer (domestic, SME, etc.) may still have very different demand behaviours from each other with very different drivers. For example, a house that uses electric heating will likely have electricity demand driven by temperature, in contrast one that is gas heated may have electricity demand which has little-to-no influence from the weather. Secondly, the demand of aggregations of consumers will likely becomes more regular the larger the aggregation although the main drivers may be less clear, or alternatively some features could become more pronounced. As shown in both Fig. 1.2 (Sect. 1.2) and Fig. 2.5 (Sect. 2.3), the weekly regularity is improved with larger aggregations.

We have already discussed weather quite a lot in this Sect. 6.2 but as in the gas versus electric heating example mentioned above it may not be obvious if weather will have an effect on the demand without further investigation. Further, as discussed in Sect. 6.2.4 it is also worth checking cross-correlation and the lagged variables as there may be delays in the effect. If a relationship is observed it is unlikely to be completely linear, as illustrated in Fig. 6.7. In this plot of temperature versus demand, demand increases below 50 °F demand due to heating but there is also an increase for temperatures above 60° due to air conditioning. In this case it may be worth modelling the relationship as a piecewise linear model, a polynomial, splines, or other basis functions (Sect. 6.2.5).

In addition to temperature, other weather variables can also be important for demand forecasts:

1. Wind Speed: higher wind speeds can increase the effect of *experienced* temperature, e.g. colder temperatures will feel colder the faster the wind speed. This may have a knock on effect that the heating is turned on sooner. This variable is known as **wind chill** and is engineered by combining the temperature and wind speed.
2. Humidity: Similarly higher humidity's can increase the felt temperature. One way to describe this is the **humidity index**, a combination of humidity and temperature.

3. Solar Radiance and visibility: the sun radiates on the earth and increases the temperature of the earth. If there is a lot of cloud then less radiation reaches the surface and the temperature will be lower than on a clear day.

Daily and weekly periodicity is often common in electricity loads. Residential and commercial demands are driven by human behaviour and needs and hence follow daily and weekly patterns. However, of course there are exceptions such as doctors and nurses who may have different daily patterns in their behaviour due to the different shifts they work. Including autoregressive effects at daily and weekly lags (e.g. a lag of 24 and 168 respectively for hourly timeseries) is one way to include the periodicity in the model. However, this doesn't model the general day of the week effects. To include an effect for "Monday", "Tuesday", etc. means including the effect of seven categorical variables into a model of electricity demand which is a continuous variables. This requires adding an update, or change, to the demand which is different depending on the day. A common way to do this is via so-called **dummy variables**. Dummy variables can take values of zero or one depending on the falsity or truth of the presence of a variable. For example, say $W(k)$ is a variable which is one if the time step k occurs on a day which is a weekday (Monday, Tuesday,..., Friday), and zero if not, i.e.

$$W(k) = \begin{cases} 1, & \text{if time step k occurs on weekday} \\ 0, & \text{otherwise} \end{cases}.$$

This is called the dummy variable for a weekday. If instead the model requires the effect of each weekday to be represented, it is required to include a corresponding dummy variable for each day. In this case, define the dummy variables $D_j(k)$, for $j = 1, \ldots, 7$ to be

$$D_j(k) = \begin{cases} 1, & \text{if time step k occurs on day } j \text{ of the week} \\ 0, & \text{otherwise} \end{cases}.$$

It is often not desirable to define N dummy variables to represent the entire set of N possible values. For example, there is only seven days of the week, which means the seventh day is simply defined as not being any of the other six days. Only six variables are needed since they can be added as a correction to the default seventh day. Since one variable can be modelled by the other six if all seven variables are included in the modelling this creates colinear variables (Sect. 13.6.1) which can cause issues with the training of parameters in the final model. This is also known as the **dummy variable trap**. Note that, the term "dummy variables" is often used in statistical modelling whereas in machine learning, it is often called **one-hot encoding**.

We've already seen in Sects. 5.1 and 6.1.1 there is often long term changes in the demand time series signal. This could be large scale annual seasonality or linear trends for example. Demand series is often changing, there is new technologies, or more efficient versions of the same appliances, or there may be changes in which customers are connected to the network (a hairdressers turning into a convenience

store). If these differences are clearly observed in the time series then it may be worth including them as explicit variables within your model. A linear trend in the series could be included by simply including the time step indicator in the model. In a simple linear model (Sect. 9.3) a trend is included by adding a term such as bt where t is the time step and $b \in \mathbb{R}$ is a parameter to be trained.

Another common feature of demand time series is annual seasonal trend. Energy demand often increases in the winter due to increased demand, and in the summer may be at its lowest level since the temperature may be warm enough so no heating is required (in hotter counties there is often an increase in demand due to air conditioning appliances). These patterns represent periodic patterns and hence should be included in the models. One option would be to simply add the day or time period of the year, by, for example generating a large number of dummy variables. However, in this case there wouldn't be many historical examples to train the parameters and the model may not generalise very well. It is often preferable to use periodic variables which can estimate the seasonality with fewer parameters. One very basic example is to use basis functions (Sect. 6.2.5). Trigonometric functions such as $\sin at$ and $\cos bt$ are one option. The parameters $a, b \in \mathbb{R}$ can be chosen (or preferably trained) to ensure that the period is appropriately chosen to match the pattern within the signal. Multiple trigonometric functions (with different periods) can be chosen to improve the fit and generate complicated patterns, see Fig. 6.11. There are more complicated choices as well such as wavelet functions, which are not explored here. We give an example of using trigonometric functions within a linear model in the Case Study in Sect. 14.2.

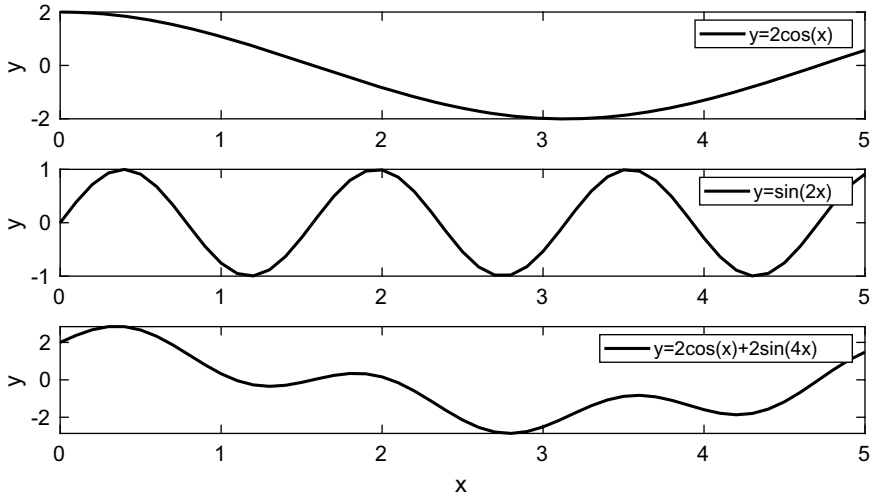


Fig. 6.11 Demonstration of how to model seasonal patterns with trigonometric functions. Two seasonal functions (top and middle) have been combined to generate a more complicated seasonal pattern (bottom)

6.3 Questions

For the questions which require using real demand data, try using some of the data as listed in Appendix D.4.

1. Consider different methods for imputing missing values. Select a demand time series. Simulate missing values by removing them from the time series (save them for comparison later). Now consider filling them in using some of the techniques given in Sect. 6.1.2. Comparing to the real values. What methods seems to perform the best? Why do you think this may be?
2. Select a demand time series, and select four weeks worth of data. Calculate the sample mean and standard deviation. Which values are more than two standard deviations from the mean? What about three standard deviations? Do these values look unrealistic or too large? Now calculate the median and interquartile range. How many interquartile ranges from the median are the largest values?
3. Take a time series with average kW or kWh values at half hourly resolution. Convert the data into hourly by averaging the data over each pair of consecutive half hours. Reconvert the data back to half hourly by linear interpolation. What is the difference in the reconstructed half hourly data compared to the original data? Where is the error largest, why is this? Try using a higher order polynomial (e.g. cubic) for the interpolation. Is this more accurate? How does it compare using household smart meter data versus system level data (e.g. GEFCOM 2014)?
4. Other than temperature what may be some other important weather variables which may affect the electricity demand within a home, or more general for the national demand? What about non-weather data, what else would be good indicators of demand?
5. Consider the London Smart meter data.[3] Plot a scatter plot of the different weather values against demand. Which variables have the strongest relationship with demand? Are the weather variables related to each other? What is the correlation between them? Which has the largest correlation value.
6. Take a demand time series. Plot the autocorrelation and partial autocorrelation of the series. At what lags is the correlation strongest? Plot a scatter plot of the demand against lags of the demand series. Include a lag of one time step, and also the lags which gave the largest autocorrelation values. Are the relationships linear? If they are linear calculate the adjusted coefficients of determination. Which lags give the biggest values?

[3] https://www.kaggle.com/datasets/jeanmidev/smart-meters-in-london?select=weather_hourly_darksky.csv.

Reference

1. T. Hong, P. Pinson, S. Fan, Global energy forecasting competition 2012. Int. J. Forecast. **30**(2), 357–363 (2014)

Chapter 7
Verification and Evaluation of Load Forecast Models

What makes a good forecast? This section aims to introduce some of the main tools for evaluating the quality of time series forecasts. It is worth noting that this is still a very active research area, especially in the developing area of probabilistic load forecasts. Obviously error measures can only be calculated after the actual observations have become available, although in practice forecasts are evaluated on the historical data by splitting it into training and testing periods (see Sect. 8.1.3).

Of course, when a forecast is required for a particular application why is it not more appropriate to simply evaluate the forecast based on its performance for that application? One of the reason's is that the performance of an application (See examples of applications in Chap. 15) is not usually defined in a simple way and may be computationally infeasible, especially if multiple evaluations are required. Instead simpler, easier to calculate measures, such as those introduced in this chapter are used. However, it still does not mean that any measure can be used and it is always preferable that one is chosen which aligns to the application as closely as possible.

This section will begin by introducing error measures which are used for both evaluating the accuracy of the forecasts but are also to compare and select between various models (Sect. 8.2). Before looking at specific error metrics and measures it is worth noting that the measures have to be different depending on whether we are considering point, or probabilistic forecasts (Sect. 5.2) with the latter having several different forms which may require different measures. The next section considers point forecast measures, and then probabilistic error measures are discussed in Sect. 7.2. These measures can be used to define skill scores, an important evaluation method for forecast skill, and are considered in Sect. 7.4. The chapter then finishes by illustrating ways to improve a forecast based on residual checks and other forecast correction methods.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_7

7.1 Point Forecast Error Measures

To define the error measures, consider two h-step-ahead point forecasts

$$\hat{\mathbf{L}}^{(1)} = (\hat{L}^{(1)}_{n+1}, \hat{L}^{(1)}_{n+2}, \ldots, \hat{L}^{(1)}_{n+h})$$

and

$$\hat{\mathbf{L}}^{(2)} = (\hat{L}^{(2)}_{n+1}, \hat{L}^{(2)}_{n+2}, \ldots, \hat{L}^{(2)}_{n+h})$$

for a time series with actual values given by $\mathbf{L} = (L_{n+1}, L_{n+2}, \ldots, L_{n+h})$. The errors between the forecasts are defined by

$$\mathbf{e}^{(k)} = \mathbf{L} - \hat{\mathbf{L}}^{(k)} = (L_{n+1} - \hat{L}^{(k)}_{n+1}, L_{n+2} - \hat{L}^{(k)}_{n+2}, \ldots, L_{n+h} - \hat{L}^{(k)}_{n+h}) = (e^{(k)}_1, e^{(k)}_2, \ldots, e^{(k)}_h), \tag{7.42}$$

where k is 1 or 2. How can these forecasts be scored and these errors summarised in order to compare which one is 'closer' to the actual values and hence which is more accurate? The answer is not obvious as there are several ways to choose how to measure this (unless $\mathbf{e}^{(k)} = \mathbf{0}$ of course, in which case you've achieved a perfect forecast!).

As an initial choice, consider norm functions, a common way of measuring the distance between vectors. Given any real-valued vector $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, the p-norm of $\mathbf{x}$ is defined to be

$$||\mathbf{x}||_p = \left(\sum_{k=1}^{N} x_k^p \right)^{1/p} = (x_1^p + x_2^p + \cdots + x_N^p)^{1/p}, \tag{7.43}$$

where $p \geq 1$. The most common norms are the 1-norm (i.e. the absolute sum),

$$||\mathbf{x}||_1 = |x|_1 + |x|_2 + \cdots + |x|_N, \tag{7.44}$$

the 2-norm (known as the standard Euclidean norm),

$$||\mathbf{x}||_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_N^2}, \tag{7.45}$$

and also the ∞-norm which is defined as $||\mathbf{x}||_\infty = \max_{k \in \{1,\ldots,N\}} |x_k|$. The p-norms are **metric functions** which have the following useful properties which make them well-defined and intuitive for measuring the difference between two vectors:

1. **Positive Definite**: $||\mathbf{x}||_p \geq 0$ and $||\mathbf{x}||_p = 0$ if and only if $\mathbf{x} = \mathbf{0} = (0, 0, \ldots, 0)$. In other words the sizes are always positive and only zero if all the elements of the vector have no size.
2. **Triangle Inequality**: For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ then $||\mathbf{x} + \mathbf{y}||_p \leq ||\mathbf{x}||_p + ||\mathbf{y}||_p$. This has the intuitive interpretation that the distance from A to B and then B to C will always be longer than the distance directly from A to C.

Table 7.1 A comparison of different p-norm values applied to two vectors as described in the main text (values to the nearest 2 decimal places)

p-norm	$\mathbf{e}^{(1)}$	$\mathbf{e}^{(2)}$
1	1.2	1.2
2	1.01	0.69
∞	1	0.4

The choice of p determines the emphasis of the p-norm on different components of the error, with larger p values meaning that the norm more strongly represents the larger error components. To illustrate this consider two error vectors $\mathbf{e}^{(1)} = (1, 0.1, 0.1)$ and $\mathbf{e}^{(2)} = (0.4, 0.4, 0.4)$, produced by two different 3-step ahead forecast models. The first model has a relatively large peak error whilst the second has no such large errors and has constant errors at each time step. The errors scores for each forecast for three p-norms with different values of p are shown in Table 7.1. First, notice that the sum of the absolute errors are equal for both forecasts and hence $||\mathbf{e}^{(k)}||_1 = 1.2$ for $k = 1, 2$. Thus the 1-norm evaluates both forecasts as having the same errors. In contrast the ∞-norm only focuses on the largest value and hence gives values of 1 and 0.4 for forecast model 1 and 2 respectively, but doesn't take into account any information about the other errors. Choosing a value of p between these extremes will produce an error value which includes contributions from all error values but with stronger influences from the larger values the larger the p value. In this example it can be seen that the 2-norm produces a similar value for $\mathbf{e}^{(1)}$ as the ∞-norm but has a value for $\mathbf{e}^{(2)}$ which is between both the 1-norm and ∞-norm. Hence the 2-norm includes a contribution from all components of the error but the larger errors contribute slightly more than the 1-norm. The point of this example is that the choice of error measure is an important aspect of the application being considered.

Despite the potential subjectivity in the choice of error measure there are some common methods which are applied in time series, and in particular load forecasting. On their own norms are not usually appropriate as error measures as they don't scale with the problem. The size of the errors will grow with the length of the series which inhibits the comparison of different forecast horizons. For this reason they are often combined with normalisations. One of the most common error measures is the mean absolute error (MAE), defined using the 1-norm as

$$\text{MAE}(\mathbf{L}, \hat{\mathbf{L}}) = \frac{1}{h}||\mathbf{L} - \hat{\mathbf{L}}||_1 = \frac{1}{h}\sum_{k=1}^{h}|L_{n+k} - \hat{L}_{n+k}|. \tag{7.46}$$

The MAE gives an average absolute error for all time points in the forecast horizon, $n + 1, \ldots, n + h$. A useful property of absolute error measures is that the units of the error are often the same as the data being considered which simplifies interpretations.

Since in much of the examples considered in this section the data will be in energy units (e.g. kWh), the errors will be in the same units as well.

Another common error measure, which also preserves the units, is the root-mean-square error (RMSE) which is defined in terms of the 2-norm as

$$\text{RMSE}(\mathbf{L}, \hat{\mathbf{L}}) = \frac{1}{\sqrt{h}}||\mathbf{L} - \hat{\mathbf{L}}||_2 = \sqrt{\frac{\sum_{k=1}^{h}(L_{n+k} - \hat{L}_{n+k})^2}{h}}. \tag{7.47}$$

As seen earlier in this section, the power on the error measure can play an important role in what type of errors the measure focuses on. The higher the power the more focus the measure has on larger errors. Hence for RMSE, the larger errors will contribute *relatively* more to the overall score than with the MAE. This can be important if you are interested in assessing which forecast may be more suitable in accurately estimating extreme values, such as peaks in demand.

A drawback of absolute-type error measures such as MAE and RMSE is the difficulty in making comparisons of accuracy when time series have different magnitudes. For example, an error of 1 kWh in a day ahead forecast is quite significant when the daily demand is only 2 kWh but negligible for substation feeders which regularly have daily demands of 100 kWh or more. A more accurate comparison of these errors may be to present the percentage errors *relative* to the size of values in the time series. In this case the 1 kWh error is 50% of the overall demand for the substation with 2 kWh daily demand but only 1% for the substation feeder with 100 kWh daily demand.

One of the most commonly used scores for evaluating the relative accuracy of a point forecasts is the mean absolute percentage error (MAPE) defined by

$$\text{MAPE}(\mathbf{L}, \hat{\mathbf{L}}) = \frac{100}{h}\sum_{k=1}^{h}\frac{|L_{n+k} - \hat{L}_{n+k}|}{|L_{n+k}|}. \tag{7.48}$$

The individual error at each time step, $|L_{n+k} - \hat{L}_{n+k}|$, is divided by the absolute demand $|L_{n+k}|$ and averaged to give a relative score. The score is often multipled by 100 in order to provide a percentage score. The MAPE is not appropriate for series which have zero or very small values, for example, household level electricity demand, or on a feeder with a lot of localised generation. Small L_k values will inflate the size of the errors at time t_k, masking the true accuracy of the forecast. The MAPE is not even defined when the true value is zero, $L_k = 0$. One alternative employed in this book is to instead replace the denominator with the average value $\frac{1}{N}\sum L_k$. This is known as the weighted absolute percentage error (WAPE). The same scaling can also be applied to the RMSE and MAE to create relative error measures. Above are some of the most common error measures used in load forecasting but of course there are several other measures which could be used, including those which avoid the issue of dividing by zero.

Directly comparing error measures between forecasts can help compare forecast accuracy but they can be complicated if the underlying time series has varying levels of predictability. In Sect. 7.4 **skill scores** are discussed which are very useful for comparing forecast models by utilising a common benchmark to help with interpretation.

It is important to carefully select the error measure that suits the application or purposes of the forecast. A special case will demonstrate this in Sect. 13.3, which presents household level load forecasts. Many standard error measures (including the ones presented here) may be inappropriate for providing an objective score for evaluating the accuracy of a forecast. Instead a novel approach is considered showing that there is no need to restrict your evaluation methods to the most popular or common ones such as RMSE or MAPE.

7.2 Probabilistic Forecast Error Measures

The above scores are only applicable to point forecasts and are not appropriate for assessing probabilistic forecasts. These forecasts are less straightforward to evaluate due to the increased complexity and the various forms that probabilistic forecasts can take (quantile, density, ensembles, etc.) as described in Sect. 5.2. In this section, the focus will be on scores for univariate[1] probabilistic forecasts. Multivariate scoring functions are only discussed in passing but suggestions for further reading can be found in Appendix D.2.

The aim with a probabilistic forecast is to accurately represent the *distribution* of the variable. Since there is only usually one observation per time step, to assess a probabilistic forecast usually means comparing the single observation against the estimate of the distribution. This makes the situation much more complicated compared to point forecasts, which can compare the single observation to the single point estimate value. Ideally the aim is to use a scoring function for which the *minimum* score is only achieved by the true distribution, these are called **proper scoring functions**.

One of the more popular proper scoring functions is the **pinball loss score** or **quantile score** which measures the accuracy of a quantile forecast (Sect. 5.2). Recall that the quantile $\tau \in [0, 1]$ of a CDF, F, is simply the value z_τ such that $F(z_\tau) = \tau$, or in other words, for a univariate distribution the probability of a random variable being less than z_τ is τ (see also Sect. 3.2 for more details on quantiles). Given an estimate of the τ quantile, z_τ, and an actual observation L from the distribution being estimated, the pinball loss function is given by

$$L_\tau(L, z_\tau) = \begin{cases} \tau(L - z_\tau) & L \geq z_\tau \\ (1 - \tau)(z_\tau - L) & L < z_\tau \end{cases}$$

[1] Recall univariate means a single variable whereas multivariate is more than one.

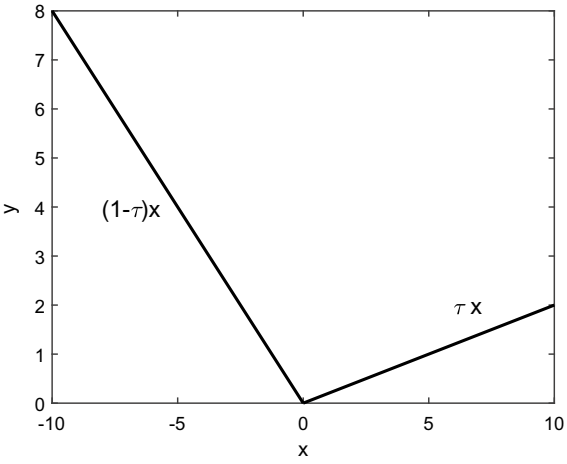


Fig. 7.1 Example of the weighting given by the pinball loss function for $\tau = 0.2$. If the input is positive then the weighting is τ, if negative, then the weighting is $1 - \tau$

The pinball function is an asymmetric function which takes the difference between the observation and the quantile and then weights the difference differently depending on whether the value is positive or negative. This asymmetry is important since an accurately estimated quantile will have, on average, a proportion, τ, of the observations below z_τ. The pinball function and its weighting is illustrate in Fig. 7.1. Typically quantile forecasts are estimated for a series of quantiles z_{τ_k}, for each time step $k = 1, \ldots, N$ in the forecast horizon and these quantiles split the range of the distribution into evenly spaced points $\tau_1, \ldots, \tau_N$ (for example popular values are the deciles $0.1, 0.2, \ldots, 0.9$, or ventiles, $0.05, 0.1, \ldots, 0.9, 0.95$). The **pinball loss score** (PLS) is simply the average over each individual loss over each quantile

$$\text{PLS} = \frac{1}{N} \sum_{k=1}^{N} L_{\tau_k}(L, z_{\tau_k}). \tag{7.49}$$

Another common proper scoring function is the continuous ranked probability score (CRPS). Consider a cumulative distribution $\hat{F}(z)$, which is an estimate of the distribution at some time for which there is an observation, defined as L. The CRPS is defined as

$$\text{CRPS}(L, \hat{F}) = \int_{-\infty}^{\infty} (\hat{F}(z) - \mathbf{1}(z - L))^2 dz = \mathbb{E}(|Z - L|) - \frac{1}{2}\mathbb{E}(|Z - \tilde{Z}|), \tag{7.50}$$

where $\mathbf{1}$ is the Heaviside step function

$$\mathbf{1}(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}. \tag{7.51}$$

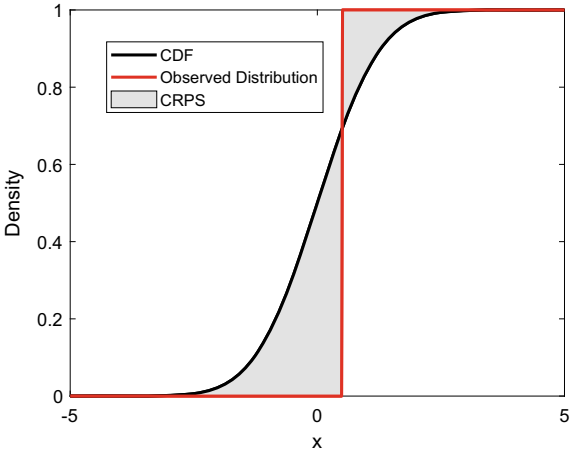


Fig. 7.2 Example of the CRPS which is the area between the CDF and the empirical distribution formed from a single observation

The first, integral form, measures the difference between the estimated distribution $\hat{F}(z)$ and the empirical cumulative distribution function for a single observation, given by $\mathbf{1}(z - L)$. The CRPS for a single observation and the estimated CDF is illustrated in Fig. 7.2, and is equal to the shaded area between the CDF and the empirical distribution (see Sect. 3.4) for the observation. The aim is to minimise this shaded area, and this is achieved by accurately estimating the true distribution.

Notice that the two terms in the second form of the CRPS describe two components of the error. The first term, $\mathbb{E}(|Z - L|)$ is the (expected) absolute difference between the observations and the forecasts. The second term, $\mathbb{E}(|Z - \tilde{Z}|)$, is a measure of the spread, i.e. the *sharpness*, of the probabilistic forecast. For a point forecast the CRPS reduces down to the first term only, i.e. the Mean absolute error. This second equivalent form of the CRPS in Eq. (7.50), $\mathbb{E}(|Z - L|) - \frac{1}{2}\mathbb{E}(|Z - \tilde{Z}|)$ suggests another way of estimating the CRPS using sample means calculated by generating random draws, $\tilde{Z}$ and Z, from the estimated distribution $\hat{F}$. For multiple observations the final CRPS is simply the average of the individual CRPS values.

The CRPS and pinball score are only suitable for univariate densities. For ensemble/scenario forecasts the second form of the CRPS given in Eq. (7.50) can be adapted to cope with ensemble forecasts which estimate a multivariate distribution. Consider a multivariate probability distribution $\mathbf{F_Z}$ which is defined for a N-dimensional random variable $\mathbf{X} = (X_1, X_2, \ldots, X_N)^T$. Given a single observation vector $\mathbf{L} = (L_1, L_2, \ldots, L_N)^T$ then the **energy score** is defined as

$$\text{ES}(\mathbf{L}, \mathbf{F}) = \mathbb{E}(||\mathbf{Z} - \mathbf{L}||_2) - \frac{1}{2}\mathbb{E}(||\mathbf{Z} - \tilde{\mathbf{Z}}||_2), \tag{7.52}$$

where $\mathbf{Z}$ and $\tilde{\mathbf{Z}}$ are independent copies of random draws/samples from the multivariate distribution. To calculate this in practice the samples, $\mathbf{Z}$ and $\tilde{\mathbf{Z}}$, are taken from the generated forecast ensembles and the sample means are used to estimate the expected

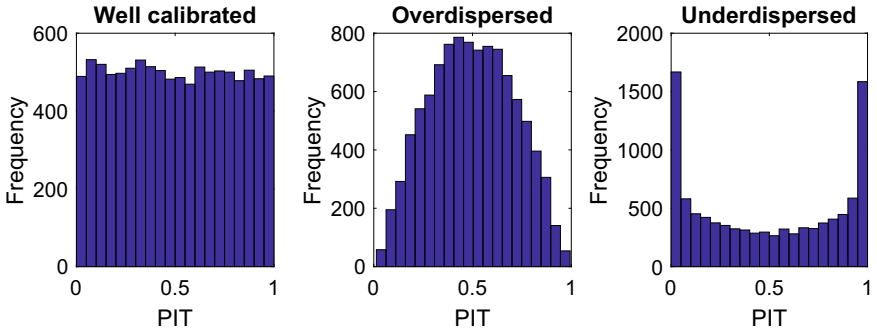


Fig. 7.3 Example of the histograms for the PIT from applying (left) the true Gaussian CDF of mean 2 and standard deviation 0.25, (middle) from applying a Gaussian CDF of mean 2 and standard deviation of 0.4 and (right) from applying a Gaussian CDF of mean 2 and standard deviation equal to 0.15

values. For the pinball score, CRPS and energy score, a smaller value implies a more accurate probabilistic forecast.

Probabilistic forecasts can also be assessed visually. Consider a CDF, F, for a continuous random variable X, then the probability integral transform (PIT) of the data, is defined by the application of the CDF to the observations $p_t = F(X_t)$. The histogram of the PIT should be uniformly distributed if the correct CDF has been chosen. To understand this consider a quantile forecast which estimates the demi-deciles, i.e. the q quantiles where $q = 0.05, 0.1, 0.15, \ldots, 0.9, 0.95$ of a continuous cumulative density function $F(x)$. If the forecast was correctly calibrated 5% of the observations should fall between any consecutive quantiles, $F^{-1}(q)$, and $F^{-1}(q + 0.05)$. In other words, the histogram of the PIT defined by this quantile estimate should be uniform with 5% of observations within each bin.

An example of the PIT histogram is shown in Fig. 7.3 for three different Gaussian CDFs (with different standard deviations) applied to random samples from one of the distributions. When the true CDF is applied (left in the Figure) then the histogram is uniform as expected. When a Gaussian CDF is applied which has a larger standard deviation than the true data then the PIT has too many observations in the centre of the histogram, and the distribution is called overdispersed (middle plot). Alternatively, if a PIT is applied using a Gaussian CDF with a smaller spread (smaller standard deviation) then there is too many observations at the edges of the histogram and the distribution is called underdispersed. Other shapes of the PIT can suggest other biases or inaccuracies in the probabilistic estimate.

An equivalent method for visualising the quality of a probabilistic forecast is a **reliability plot** (or reliability diagram). For this the quantiles of the probabilistic forecast are plotted against the observed relative frequency. In other words, take the τth quantile $F^{-1}(\tau)$, for a probabilistic forecast with CDF, F, and suppose there are observations $y_1, y_2, \ldots, y_N$. These points can be used to define an empirical distribution function $F_E(y)$ which is a step function defined by

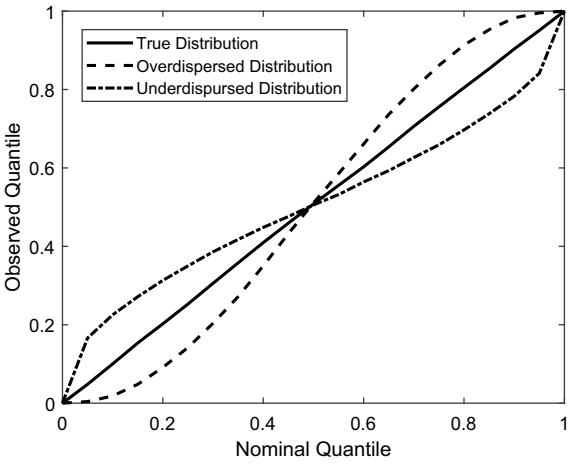


Fig. 7.4 Reliability diagram for the three different estimates for the distribution for the same data as in Fig. 7.3

$$\hat{F}_E(y) = \frac{\text{number observations less than } X}{N} = \frac{1}{N}\sum_{k=1}^{N} \mathbf{1}_{y_k < y}, \tag{7.53}$$

where $\mathbf{1}_S$ is the indicator function which takes the value 1 if the statement S is true and 0 otherwise (also see Sect. 3.4). A reliability diagram is simply a comparison of the quantiles of the estimated distribution, F, with the empirical distribution, F_E. The quantiles should be similar (for the same probability value τ) if the estimate F is an accurate representation of the distribution of the observations (as estimated by the empirical CDF, F_E). The reliability diagram for the same distributions as in Fig. 7.3 are shown in Fig. 7.4. This is for 1000 observations from the true normal distribution with mean 2 and standard deviation 0.25. Notice that in the reliability diagram if the observations are from the true distribution they should be close to the diagonal $y = x$. In contrast the line for the overdispersed distribution has a small gradient for the under-represented tail quantiles but high gradient in the middle for the over-represented quantiles. The opposite is true for the underdispersed distribution which has high gradients at the tail quantiles and a lower gradient in the central quantiles.

It is worth noting, that a uniform PIT (or equivalently a reliability plot lying on the $y = x$ line) is only a necessary condition and not a sufficient condition for the distribution to be the true underlying distribution for the data. In other words, uniform PITs can still occur even if the estimated distribution is not a true representation of the underlying distribution.

7.3 Causes of Forecast Error

There will always be some error in the forecasts no matter the sophistication of the model. However, there are some common causes of forecast error it is worth briefly mentioning to prevent jumping to conclusions and assist in interpreting the models and their errors.

Even when an accurate model has been generated with both low-bias and low-variance the forecasts errors are likely to increase with the forecast horizon, this is illustrated in Fig. 7.5. This is because there is usually a interdependency between values which are close in time. This is particularly true in energy demand behaviour where appliances are used over several hours (heaters), or similar actions are performed together (a morning shower followed by boiling a kettle for a cup of tea). Hence if comparing the errors for forecasting tomorrow, versus forecasts for the following day and so on, there would be an expected upward trajectory in the forecast errors/scores.

However, things may not be as simple as this. In the Case study in Chap. 14, the forecast errors vary within a day (see Fig. 14.7 in particular). This is because there is more variation in demand (and hence larger errors on average) during certain periods of the day compared to others. However, even in this case the average daily errors do seem to be increasing. This highlights another source of forecast error which is the volatility of particular periods.

Another source of forecast error may be due to their dependence on the input variables. Many load forecasts are strongly related to weather (e.g. see Sect. 6.7) and therefore weather forecasts are utilised within the load forecast models. However, if these inputs are inaccurate (for example through measurement, forecasts or even calibration errors) then the load forecast will also be inaccurate. In other words, for surprising or unusual errors in the data, the input variables should also be considered as possible causes.

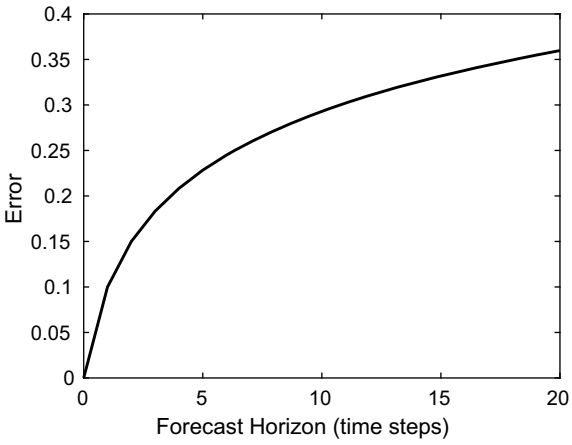


Fig. 7.5 Expected forecast error as a function of forecast horizon

Benchmarks also serve a useful function of determining the causes of forecast errors. Since they may include different inputs than the main models they can confirm which variables may be sources of large errors. Benchmarks are also useful for comparing models and understanding improvements over time, even when the underlying data changes. This is explored in the next section.

7.4 Skill Scores

Even if an error measure is appropriate to the application, it may not be easy to compare or evaluate forecasts, especially if comparing on multiple datasets. For example, consider two forecast models where one model produces an estimate for one dataset and the other model produces an estimate for another, dataset. If these datasets have different volatilities (for example it could be that the data is for different seasons, where say heating appliances may make Winter demand behaviour more volatile) then it will not be clear how to compare the accuracy of these forecasts. Similarly, how do you keep track of the improvement (or degradation) in the same forecast over time, which will be using more and/or newer data?

One way to help discriminate between forecasts in cases like the above and others, is to use a **skill score**. A skill score measures the accuracy of a forecast *relative* to some benchmark score. They are very common in numerical weather prediction applications, where they are used to show the improvement of forecast models over time.

Skill scores can take many forms but a common format is

$$SS(\hat{\mathbf{L}}, \mathbf{L}_b) = \frac{E_f - E_b}{E_p - E_b}, \tag{7.54}$$

where E_f, E_b, E_p represent the error scores for the main forecast, the benchmark forecast and the perfect forecast respectively. The error measure could be any of those presented above, such as RMSE, or MAE for point forecasts, or CRPS for probabilistic forecasts.

Often the error should be zero for a perfect forecast, and in this case the skill score reduces to

$$SS(\hat{\mathbf{L}}, \mathbf{L}_b) = 1 - \frac{E_f}{E_b}. \tag{7.55}$$

The skill score can obtain a maximum value of 1 if the forecast is perfect ($E_f = 0$), but is equal to zero if is only as good as the benchmark, and of course the score can be negative if the forecast is worse than the benchmark.

The benchmark here is often called a standard, or reference, forecast, but the important point is that this method is kept constant to allow more appropriate comparisons. Since the benchmark methodology stays the same then this allows a comparative analysis of the forecast across different datasets as well as over different

periods in the same dataset. If two datasets have very different "predictabilities" then you can compare the performance on them better via a skill score since the forecast error on the less predictable dataset will be scaled according to the common benchmark, which will also perform more poorly on this data set relatively to the other. As a consequence, the relative performance on the datasets can be more easily compared and will not simply be based on bad luck due to the features of the test dataset that is used.

The main question when creating a skill score then, is what is an appropriate benchmark to use? The following is some suggested criteria:

1. It should be quite simple and not require too much data or additional data sets to produce. This enables the model to be used in most circumstances.
2. It should be easy to implement so that other forecasters can easily replicate it.
3. It should be easy to interpret to help with model evaluation and improvement.
4. It should not be too sophisticated, or state-of-the-art. It is only needed for comparison and hence there is no need for a complicated or "difficult to beat" model.

For many applications, the simple benchmark models described in Chap. 9 will be sufficient. The persistence model is quite common. Other considerations about choosing appropriate benchmarks are given in Sect. 8.1.1.

7.5 Residual Checks and Forecast Corrections

Ideally a forecast is a good estimator of the true load but for various reasons may require some corrections. Common in climate modelling is a model bias where the mean (or expected) value of the prediction is consistently shifted from the actual values. Analysing the residuals of the final forecast model is a common way to both evaluate your model and identify possible improvements for future implementations.

Whatever models are created for time series forecasting there may still be some structure remaining in the residuals which could be exploited to further improve the accuracy of the forecast. Suppose a forecast model is generated for the time series L_t over the time steps in the training data $t = 1, \ldots N$. Let $\hat{L}_t$ represent a forecast estimate fitted to the training data and recall from Sect. 5.2 that the residual time series can be defined as $r_t = L_t - \hat{L}_t$ for $t = 1, \ldots N$. A desirable feature for a forecast model is that this residual series is essentially random noise, since any remaining patterns/relationships could be used to improve the forecast.

The first check should be to plot the residual time series and look for any remaining patterns or features. If the model has correctly explained the data, then the residual series should be random noise,[2] in other words, their values are independent and identically distributed with zero mean.

[2] The noise can follow a particular distribution even if it is random. White noise, is noise which is distributed according to a Gaussian function.

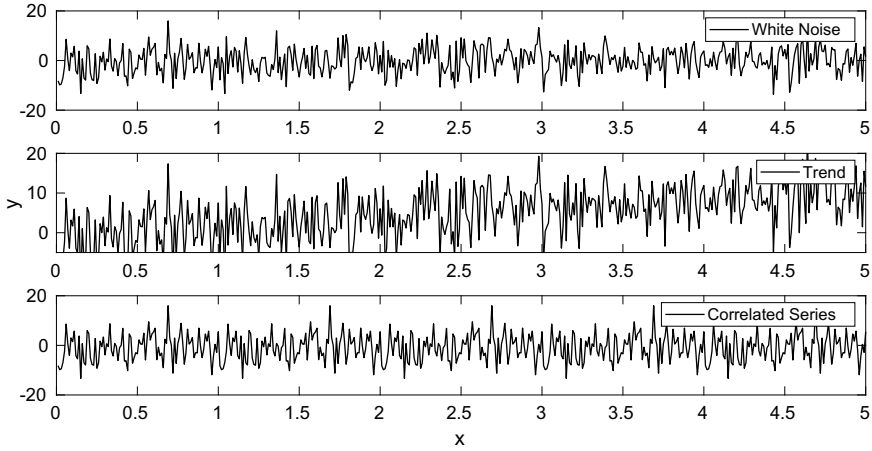


Fig. 7.6 Examples of three time series: white noise (top), white noise plus trend (middle), and a periodic time series based on the first white noise series (bottom)

If two random variables X and Y are independent it means, unsurprisingly, that the value of one is unrelated to the value of the other. In the language of probability this means for all x, y the events $X \leq x$ and $Y \leq y$ are independent. In other words the joint distribution, $F_{X,Y}(x, y)$, of X, Y is related to the individual distributions via

$$F_{X,Y}(x, y) = F_X(x)F_Y(y), \tag{7.56}$$

where F_X and F_Y are the cumulative distribution functions for X and Y respectively (see Sect. 3.3). For an independent variable the correlation between the values is zero. However, note that the reverse is not necessarily true, zero correlation does not imply they are independent. However, it can be used as evidence for independence, or at least increase the plausibility that they are independent.

It is usually quite easy to tell if the data is not white noise but not trivial to test if it is. A plot of the time series should give an initial indication of which case may be true. Examples of a few residual time series are shown in Fig. 7.6. In this example, the values at all time steps have the same variance, hence the only thing to check is whether they have zero mean and are independent.

In this Fig. 7.6 there are two that look like white noise and one that is quite clearly not white noise. The top plot is white noise, the middle plot is the original white noise series but augmented with an increasing trend. The bottom series looks like it is white noise but however is formed by repeating a chunk of 100 data points from the top series. Therefore in fact it has strong autocorrelation despite not being directly obvious.

The autoregressive features in this series can be confirmed by looking at the autocorrelation plot (As when creating the ARIMA model in Sect. 9.4). In this case the periodic time series does not have components which are independent as seen by

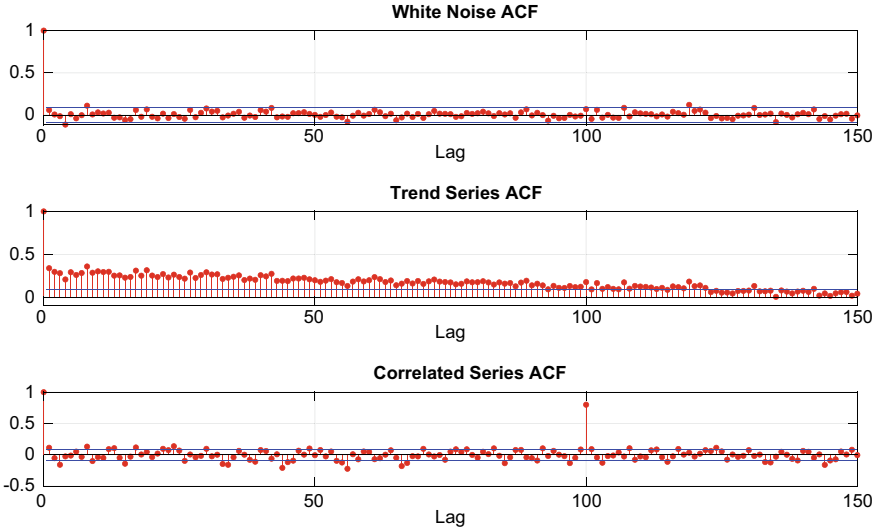


Fig. 7.7 ACF's of the three "white noise" time series: white noise (top), white noise plus trend (middle), and a periodic time series based on the white noise series (bottom)

the spike at lag 100 in Fig. 7.7. The ACF plot shows that the true white noise series has no autocorrelation as expected, but the noise with trend in the middle does show up via the slow decay in the ACF as a function of lag.

Any autocorrelation which remains in the residuals can be removed by including extra autoregressive components to the residual series (alternatively can be added to the original model) via

$$r_k = \sum_{k=1}^{p_{\max}} \phi_k r_{t-k} + \epsilon_t \tag{7.57}$$

for some assumed Gaussian error ϵ_t and optimal autoregressive order p found by minimising the Akaike information criterion (AIC) or Bayesian information criterion (BIC) (see Sect. 8.2.2) over $p \in \{0, \ldots, p_{\max}\}$, for some maximum order $p_{\max}$. After training the coefficients $\phi_1, \ldots, \phi_{\max}$, a forecast for the residual can be produced via $\hat{r}_k = \sum_{k=1}^{p_{\max}} \phi_k r_{t-k}$ and the original forecast can be updated to achieve a new forecast for the load series via $\hat{L}_t + \hat{r}_t$. If sufficient lags have been included in the updated model, the new residual series $\tilde{r}_t = L_t - \hat{L}_t - \hat{r}_t$ should now have no significant autocorrelations. Additionally it is hoped that the new models will have improved forecast accuracy. Of course another autocorrelation check of the residuals can be performed on the new forecast and the process repeated if not all of the autoregressive features have been accounted for.

Another form of bias in a time series forecast is whether the residuals are centred around zero. The random noise with trend is one such example. The simplest form of bias is where the noise is centred around a non-zero constant. This suggests a

simple bias correction can be applied to reduce the mean error to zero by shifting by the sampled mean of the residual. I.e. set $\hat{r}_t = r_t + b$ where $b = -\frac{\sum_{t=1}^{N} r_t}{N} \in \mathbb{R}$. Alternatively one can directly update the forecast itself, i.e.

$$\hat{\hat{L}}_t = \hat{L}_t + b. \tag{7.58}$$

Thus by definition the new residual series has sample mean equal to zero. The residual time series with linear trend can also be corrected in a similar way except by detrending with the line of best fit through the points (similarly as seen for the outlier detection in Sect. 6.1.1). More generally, where the trend is obvious, similar detrending approaches can be applied.

There may also be assumptions concerning the distribution of the residuals for particular models. For example linear regression and ARIMA models assume Gaussian distributions. If the residuals are not distributed symmetrically then a transformations of the data may be required (Sect. 6.1.3). Further non-constant variance of the residuals suggests that methods which assume fixed variance may not be appropriate. Instead, alternative approaches such as the GARCH type models introduced in Sect. 11.6.2 may be required.

In general applying forecast correction and checking for independence is not straight forward. As shown above, time series plots of the residuals should be the first consideration and then checks for constant mean and variances can be performed by calculating them on fixed intervals of the residual time series and comparing them to the full sample mean and variance. Finally, the autocorrelation and partial autocorrelation functions should be plotted to check for moving average and autoregressive components and identify dependence between points in the time series.

The above methods are primarily focused on point estimates. However, for probabilistic forecasts there are also corrections which can be applied, but they are often more complicated than point forecast corrections. For a simple case recall in Sect. 7.2 that a probabilistic forecasts should have a uniform probability integral transform, but if this is not the case, then the PIT can also suggest ways to inform possible corrections. For example, as seen in Sect. 7.2, overdispersed (alternatively underdispersed) forecasts produce a wider (or narrower for underdispersed) PIT distribution than is desired, which means the model could be improved by squashing (or stretching for underdispersed estimates) the distribution. More generally we can look at the PIT to see which areas of the distribution are over or under represented. There are more sophisticated calibration methods such as quantile mapping which have traditionally been applied in climate and weather modelling, further reading in these areas are given in the Appendix D.

7.6 Questions

For the questions which require using real demand data, try using some of the data as listed in Appendix D.4.

1. Take a few days from a real demand time series and create basic forecasts by shifting the profile by full day. Calculate the MAPE, MAE and RMSE. Compare them. Take a hundred smart meter time series and calculate the errors based on the same seasonal persistence forecast model. Produce a scatter plot of the errors against the size of the demand (e.g. the average half hourly or daily demand). Is there a pattern you notice in the plots? If you plot the time series of the profile against the forecasts can you identify the sources of error for those with the best and worst accuracy?
2. Take a half hourly household demand profile with a peak in the evening. Take a day and shift the profile by an hour in one direction (add the shifted points that fall off the end to the other side). Now calculate the RMSE error between them. Next produce a flat profile by taking the average half hourly value and setting all half hours of the day to this value. Calculate the RMSE between this and the original profile. Compare the two error values. Which is smaller? Try this with several other forecasts. Is the flat profile producing smaller errors than the shifted in some cases? This is explored more in Sect. 13.3.
3. Sample 5000 points from a univariate Gaussian distribution. Select quantiles at $0.05, 0.1, 0.15, \ldots, 0.95$ and plot the PIT. How many points should be in each quantile range? Now delete 5–10 points from the middle five quantiles of the distribution. Plot the PIT again, how has the shape changed? Is it underdispersed or overdispersed? Repeat the experiment but remove values from the tails of the distribution. Replot the PIT and check whether the shape is underdispersed or overdispersed. Now plot the reliability diagrams for all three samples (this will require calculating the empirical quantiles for each sample).
4. Sample 1000 points from a univariate distribution of your choice. Create three empirical distributions from these samples by deleting the same number of points (say 10%) (a) randomly, (b) from the centre of the distribution, and (c) from the tails of the sample. Use the samples to calculate quantiles which will now define your probabilistic estimates. Calculate the pinball loss score for these three distributions on the original sample of points. Repeat the calculation for the CRPS. Which has the best (lowest) score?
5. Consider forecast errors with horizon. Take some half hourly or hourly demand data. Create a simple forecast of the next two weeks by repeating the daily profile for one day, for the next fourteen days. Calculate the RMSE error for each day. How does it change with horizon? Repeat this with other time series and observe the change with horizon. Does it change smoothly with how many days ahead? Or is there a change depending on the day of the week?

6. Take the forecast used in the last question. Produce the residual time series. Plot the autocorrelation and partial autocorrelation plots. Which lags produce the biggest coefficient values? How many lags would you therefore expect to need to correct for this in an autoregressive update to this model? If you know how to apply linear regression try adding these terms to your model and repeat the forecast again. How have the errors changed? If you don't know how to apply this, you can wait until you've read Chap. 9 and come back to this part of the question!

Chapter 8
Load Forecasting Model Training and Selection

Chapters 5 and 6 have shown how to define a time series forecast, how to prepare the data, and how to generate inputs for the models. Chapters 9 to 11 will show several methods for forecasting the demand. However, although Chap. 7 provided us the tools for measuring the accuracy of a forecast, the following questions remain largely unanswered: **How do we train and select a model which will consistently produce accurate forecasts?**

This chapter will investigate this question by looking at some of the most important aspects for creating a good forecast including proper utilisation of benchmarking, and how to use cross-validation to properly train your model. Underlying cross-validation is one of the most important aspects of a creating a good forecast, the so-called **bias-variance trade-off** principle, discussed in Sect. 8.1.2. This ensures that the model is not over (or under-) trained and allows the model to better generalise to new, unseen data. Next, in Sect. 8.2, methods for training the models are considered, including ways to select the best model from a selection of models. One important set of techniques covered in Sects. 8.2.4 and 8.2.5 is regularisation, which helps to reduce overfitting, but also how to find the appropriate hyperparameters within a family of models.

8.1 General Principles for Forecasts Trials

In the previous sections the general form of a forecasting problem was introduced as well as methods for scoring the forecast accuracy through error measures. This section introduces some general principles with the aim to aid the practitioner to properly design and develop a forecast trial. This includes considerations on why choosing appropriate benchmarks is important to better understand the accuracy of your model; why it is important to avoid over/under-fitting your model to the data; and how to split the data in order to properly train and test your models.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_8

8.1.1 Benchmarking

An error score (see Chap. 7) for a model is not very informative on its own. The accuracy of a forecast can only be understood in the context of other, well-designed forecasts. **Benchmark** models are a vital component for creating useful and accurate forecasts. They enable informative comparisons and help to better understand important (and unimportant) features and relationships in the data. Often simple benchmarks can be quite effective as their strong performance can suggest important features or drivers for the forecast accuracy. How much your model(s) improve compared to the benchmarks can also be used as performance indicators (See skill scores in Sect. 7.4).

Most benchmarks fit into the following categories:

1. **Simple or Naïve benchmarks**. These are very basic benchmarks models which have minimal features and parameters. They serve as the lowest bar for which your main forecast model should outperform. If they don't, then, due to their simple form, these benchmarks should be able to suggest improvements to the current model or indicate flaws in the chosen model. A selection of several of these simple benchmarks can also highlight some of the most important features in the underlying data. At least one of the benchmarks in a forecast trial should be simple.
2. **Common benchmarks**. Different applications will have some models which are commonly used as benchmarks. For example, this could be ARIMAX or simple linear regression models (Forecast models will be introduced in detail in Chap. 9). This can be helpful since it allows some degree of comparison between different models across different experiments even though the underlying data or situation is completely different.
3. **State-of-the-art benchmarks**. Often it will be desirable to compare to the current best methods available and implement a version of the state-of-the-art in the selection of different models. Even if the model doesn't quite outperform the best in the business, confidence can be given to a model which performs similarly to models which been tried-and-tested and shown to work well over several experiments and data sets. In many cases it may be difficult to identify any single model which performs well in general and instead at least one, well-known, competitive model should be chosen for comparison in your experiment.

In addition to choosing a naïve or common model, a simple way to choose a benchmark is to base them on at least one feature/relationship which appears to be important for the dependent variable of interest. In load forecasting, there is often weekly or daily seasonalities, and therefore it is common to pick a benchmark model which includes these features. Several common benchmark methods for load forecasting will be introduced in Sect. 9.1.

It should be highlighted that just because a model has the smallest error there is no guarantee it will achieve the best performance when used within the chosen application. However, it is often not computationally viable to assess the model

by testing each forecast model in the chosen application (e.g. storage control as introduced in Sect. 15.1). That is why it is important to carefully select the forecast error metric which reflects the aims of the forecast (see Chap. 7).

8.1.2 Bias-Variance Tradeoff

The **bias-variance** tradeoff is one of the single most important concepts in creating an accurate forecast. As seen in Sect. 5.2 and Eq. (5.27), a time series forecast is essentially a function which takes various inputs to give the desired outputs. The nature of the function is determined by a number of parameters which must be trained on historical data. How to properly choose and train the parameters can have a large impact on the overall accuracy of the forecast.

As introduced in Chap. 4 machine learning was defined as algorithms that learn from data to improve prediction performance. However, there is no practical value if a machine learning model is only capable of predicting accurately based on instances from the data it was trained on. Here, a model that simply memorised all the training data can, in theory, achieve perfect performance. However, this is meaningless for all practical problems, as it is typically infeasible that all possible inputs can be measured (e.g. if the variables are real-valued). Therefore, the central challenge is to train a machine learning model that performs well on new, previously unseen inputs. The ability to perform well on previously unobserved inputs is called **generalisation**.

At the one extreme it may be desirable to choose a model with a large number of parameters and train it so it fits very closely to the training data. However, the more parameters, the more likely the model is to fit to spurious noise in the time series signal and hence cannot be extrapolated very well to new data. This is often called **overfitting** the model to the data. In this case, small changes in the input to the model will produce large errors and hence the model is said to have **high variance**. A high variance model does not generalise well to new data. In contrast a model with very few parameters will miss some of the core features of the time series and **underfit** the data. It means that on average the errors will be quite large and the model is said to have **high bias**.

The bias and variance can be expressed in more precise mathematical terms. Consider a model which relates the true relationship between a dependent value L (e.g. Load), and an independent variable Z (say temperature), via a function f

$$\hat{L} = f(Z) + \epsilon, \tag{8.1}$$

with noise ϵ (with assumed zero mean). The aim is to develop a model $\hat{f}(Z, \boldsymbol{\beta})$ that estimates the true function $f(Z)$, by learning the parameters $\boldsymbol{\beta}$ over some training data of observed values. Now suppose this estimate is produced by minimising the mean squared error, a common error measure for time series forecasts (see Chap. 7),

$$MSE(f(Z), \hat{f}(Z, \boldsymbol{\beta})) = \mathbb{E}[(f(Z) - \hat{f}(Z, \boldsymbol{\beta}))^2]. \tag{8.2}$$

It turns out that this can be broken down as follows

$$\mathbb{E}[(f(Z) - \hat{f}(Z, \boldsymbol{\beta}))^2)] = (\mathbb{E}[\hat{f}(Z, \boldsymbol{\beta})] - f(Z))^2 + \mathbb{E}[(\hat{f}(Z, \boldsymbol{\beta}) - \mathbb{E}[\hat{f}(Z, \boldsymbol{\beta})])^2] + \sigma^2, \tag{8.3}$$

where σ^2 is the variance of ϵ. The first term is the square of the bias $(\mathbb{E}[\hat{f}(Z, \boldsymbol{\beta})] - f(Z))^2$ and describes the difference between the model output and the output from the true function. The bias term will be large if the model is too simple to capture the pattern in the data. The second term $\mathbb{E}[(\hat{f}(Z, \boldsymbol{\beta}) - \mathbb{E}[\hat{f}(Z, \boldsymbol{\beta})])^2]$ is the variance, and describe the dispersion of the model outputs around the mean. In practical terms this measures how spread out the errors are around the mean. Finally there is the **irreducible error** defined by σ^2. This is the error that can never be reduced which limits how much the MSE can be reduced. The key to producing consistently accurate forecasts is to get low bias and low variance, i.e. a model which captures the main features in the data but also generalises well to new data.

As an example consider a simple model $y = x^3 - 15x^2 + 66x - 60 + \epsilon$, with irreducible errors, ϵ, which is chosen to be Gaussian with mean zero and variance $\sigma^2 = 20$. This function takes as input x, and gives the observed outputs, y. Points are generated from the true model $x^3 - 15x^2 + 66x - 60$ to give pairs of input-outputs (x_k, y_k) for $k = 1, \ldots, 50$. To replicate a real system, random error samples from the Gaussian distribution, ϵ_k, are added to each true dependent variable, y_k, to give observed points $\hat{y}_k = y_k + \epsilon_k$, for $k = 1, \ldots, 50$. Hence the true outputs are unknown to the modeller who only sees the inputs with the noisy outputs, i.e. $(x_k, \hat{y}_k)$. This means it will be impossible to create a perfect match between any model and the original observations.

Now consider fitting polynomials of different orders to these noisy points. The first model is a simple linear one of the form, $a_1 x + a_0$, this is an underparameterised model and is expected to have high bias but low variance. The second model is a cubic polynomial of the form $a_3 x^3 + a_2 x^2 + a_1 x + a_0$, and should be a good balance between matching the general shape of the data without overfitting the noise. The final model is a polynomial of the order 20, i.e. of the form $a_{20} x^{20} + a_{19} x^{19} + \cdots a_2 x^2 + a_1 x + a_0$ which would be expected to overfit to the data and thus have high variance. In each case the coefficients (The $a_i's$) are trained to find the best fit to the points for that model (how to train the fit will be covered in Sect. 8.2).

The results are shown in Fig. 8.1 for the three different models. As expected the best fitting model is the cubic model which is very close to the original curve and the noisy observations (red circles). The highly parameterised polynomial of degree 20 clearly overfits to the noise and will not generalise (i.e. will not estimate the correct output) very well to new inputs, it has high variance. In contrast the linear polynomial on the left does not fit the data very well but gets the general level. It has high bias and clearly underfits the data.

There are several strategies for avoiding over- or under-fitting the data. Several techniques will be introduced in this chapter, including cross-validation in the following section, regularisation methods (Sect. 8.2.4) and information criterion (Sect. 8.2.2).

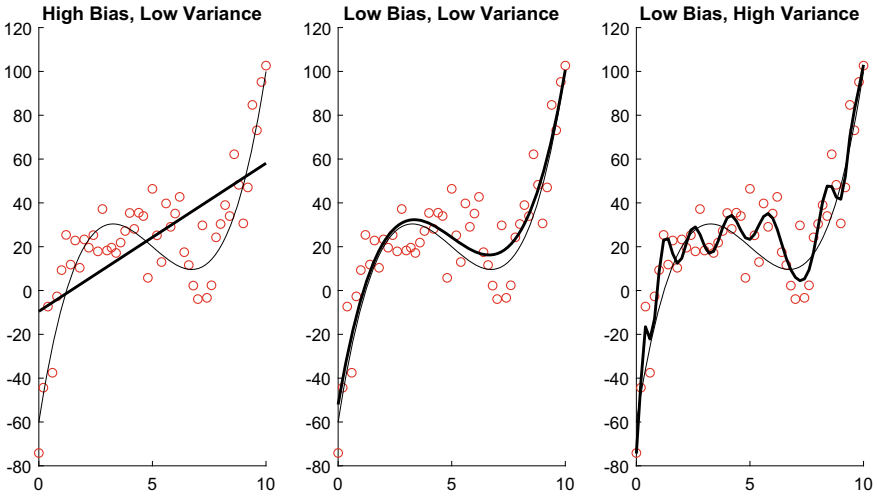


Fig. 8.1 Example of bias-variance trade-off by fitting different polynomials (thick black curve) to the observations (red circles). The true polynomial which generated the data is also shown as a thin black line. The left hand plot shows the fitting using a polynomial of degree 1, the middle plot using a polynomial model of degree 3 and the right plot using a polynomial of degree 20. Full details are in the main text

8.1.3 Cross-Validation Methods

In order to understand how a forecast model will perform in practice the available training data must be split into appropriate components. Time series data, the focus of this book, adds an extra potential restriction due to the chronological order of the data. This section will discuss some of the motivations and principles of cross-validation.

Forecast models must be tested on unseen data to ensure that the forecaster is not unrealistically tailoring (subconsciously or otherwise) the model to score higher than would be possible in practice. In real applications the future data is not available and forecasters would not have the advantages of knowing the actual values in advance. Hence designing a forecasting trial is very much like designing a blind experiment in medicine in order to test a particular hypothesis for whether a treatment is effective or not.

Another, related, reason for splitting the data is to choose a model with a good bias-variance trade-off (see Sect. 8.1.2). **Cross-validation**, the topic of this section, is one way to select a model so that it is not over- or under-fitted to the data, i.e. that it generalises well to unseen data.

For these reasons the data in machine learning trials is split into a unseen part, called the **test set**, or **hold-out set**, and a part for training the parameters and hyper-parameters of your model, often called the **training set**. For time series, the ordering of the data is often relevant and hence the test set typically follows chronologically from the training set. We will call this a **time-series split** (other approaches

will be discussed shortly). For example, consider a time series $L_1, L_2, \ldots L_{N+k}$, for $N, k > 1$ and suppose the aim is to produce 1-step ahead forecasts for the test set consisting of data at the time steps $N+1, N+2, \ldots, N+k$. Any data (including any other explanatory variables, see Sect. 5.2) prior to the start of the test set is part of the training data. The following steps are then implemented:

1. The first forecast value $\hat{L}_{N+1}$ is produced by training a model (see Eq. (5.27)) on the current training data $L_1, L_2, \ldots L_N$ (as well as any other explanatory data).
2. The next step ahead forecast $\hat{L}_{N+2}$ is then produced by retraining[1] the data on $L_1, L_2, \ldots L_{N+1}$ (i.e. the last observation at $N+1$ is now included in the new training data).
3. This continues until the kth time step of the test period has been reached.

Note if a multistep ahead point forecast is being produced from a one-step ahead forecast then instead forecasts from the previous time steps are used as inputs to the model rather than the actual observations, e.g. for the mth step ahead the forecast would use as inputs $L_1, L_2, \ldots, L_N, \hat{L}_{N+1}, \ldots, \hat{L}_{N+m-1}$.

To find the most accurate forecast, a large number of models could be trained on the training set and the the errors on their predictions could be compared. However, this is often computationally infeasible. Further, the trialing of a large number of models increases the possibility that one particular forecast will have high performance by chance alone rather than due to its particular suitability for predicting the data behaviour. Hence, it is more practical and reliable to test a relatively small number of models. Recall in Sect. 8.1.2, that a core goal when creating a forecast is to balance the bias and variance, and find a model which accurately generalises to new data. This means not overfitting to the training data set by using a very complicated model, but also not using a very simple model which under-fits the data.

One of the most common ways to do this is to split another set, called the **validation set** off the end of the training set, and use this to help select a well-trained model and to select appropriate **hyperparameters** (Sect. 8.2.3). Hyperparameters are parameters of the algorithm which have to be chosen before the remaining parameters such as weights are determined in training and influence the training. Examples are the regularisation parameter used in regularisation methods in Sect. 8.2.4, and the number of layers and nodes in artificial neural networks (Sect. 10.4). The original shortened data set is now simply renamed the training data.

To use the validation data set for model selection, a family of models, are fit on the training data with different hyperparameter values (for example for neural networks, the number of nodes in each layer, number of layers, see Sect. 10.4) are used to generate a forecast for the data in the validation dataset (i.e. in a similar way as they are generated for a test set). The performance of the models are then compared, e.g. using the error measures introduced in Chap. 7, and the best performing models are selected to produce forecasts on the test set, often after being retrained on the combined training and validation set. The additional testing of the models introduced

[1] The model may not have to be retrained at every new time step, especially if it retains the same accuracy and if it is too expensive to retrain.

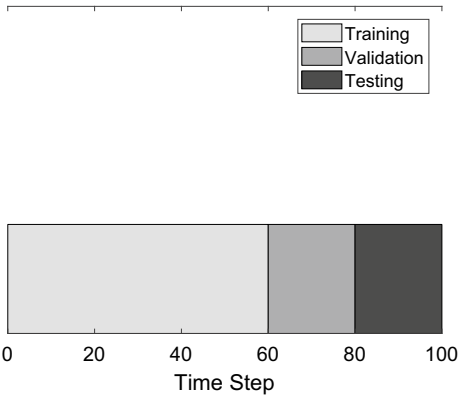


Fig. 8.2 Example illustrating a timeseries split into Training, Validation and Test set in a 3:1:1 ratio

by using a validation set improves the bias-variance tradeoff by eliminating models which over- or under-train. In addition, good performance on both the validation and test set increases confidence with using these models. Further hyperparameter selection methods are considered in Sect. 8.2.3.

A common split of the data is about 60%, 20%, 20% for the training, validation and testing set respectively, although this can be adjusted depending on the problem. As mentioned, more testing data increases the confidence in the performance of a model but sufficient training data is required to improve the chances that the models generalise as much as possible. An illustration of splitting the data into a training, validation and testing set is shown in Fig. 8.2.[2]

Given the above split of the data the following is a typical procedure to generate forecasts:

1. Given a family (or families) of forecast models, and suitable benchmarks (Sect. 8.1.1), train the model parameters on the training data.
2. Produce a forecasts over the validation set and compare the models (using the appropriate error measures as will be introduced in Chap. 7) within the same family to select a set of optimal values for the hyperparameters.
3. On the selected models (which may include one or two choices of hyperparameters for each family of models), re-train the models on the combined training and validation set.
4. Produce a forecast for the test set.

To illustrate the process consider a simple example. Take the model $y = \sin(x) + 0.2\sin(2x) + 0.4\sin(4x)$ which is used to generate values y_k at $x_k = 0.2(k-1)$ for $k = 1, 2, \ldots, 100$ to give 100 points (x_k, y_k). To make the data more realistic add random samples from a Gaussian distribution with mean 0 and variance 0.4 to the y

[2] Note there are ways to train models without using a validation set, for example, automatic model selection using information criteria (see Sect. 8.2.2) and this can be preferable if the amount of data available for training is quite low.

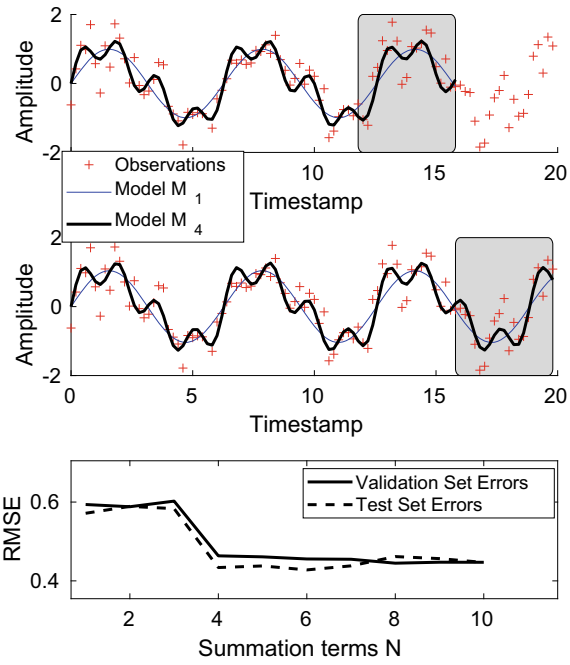


Fig. 8.3 Illustration of validation and testing for a simple example as described in the text. **a** Shows the forecasts of model M_1 and M_4 for the validation set (shaded), **b** Shows the same models for the test set. **c** Summarises the RMSE errors for the models M_N for $N = 1, \ldots, 10$ (Eq. 8.4) for both the validation and the test set

values to produce the updated input-output pairs $(x_k, \hat{y}_k)$. Next, consider fitting a set of models to the noisy observations, of the form

$$M_N(x) = \sum_{n=1}^{N} a_n \sin(nx), \tag{8.4}$$

for $N = 1, 2, \ldots, 10$ which for larger values includes more higher frequency terms to the model. Note that N is a hyperparameter for this family of models. Let the first 60% of the points be the training set (defined at $x = 0, 0.2, \ldots, 11.8$), the next 20% as the validation set ($x = 12, 12.2, \ldots, 15.8$) and the final 20% as the test set (points $x = 16, 16.2, \ldots, 19.8$). The models are trained for each N on the training data set.

Figure 8.3a shows the models M_1 and M_4 trained on the training set including its prediction on the validation set (shaded box). The noisy observations are shown as red circles. It's clear from this plot that the simplest model M_1 captures the main periodic behaviour but misses the higher frequency oscillations. In contrast the model M_4 (which matches the order of the true model) is much more accurate, as would be expected. Similarly Fig. 8.3b shows the prediction on the test set for the same models. This prediction has now been formed by training on the original training data set together with the validation set.

The RMSE errors (Chap. 7) for the ten models are shown in Fig. 8.3c for both the validation and test set. The accuracy drops down for $N = 4$ for both validation and

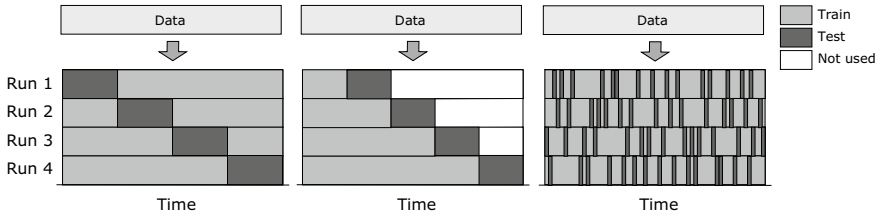


Fig. 8.4 Comparison of various cross-validation schemes. Figure from From Tutorial 1: Building Load Forecasting with ML licensed under CC BY 4.0

test set. This is because the complexity of the models required a periodic term of at least $\sin(4x)$. In addition, for these models the a_1, a_2, a_4 coefficients have the largest magnitude, as would be expected since they coincide with the terms in the actual underlying model, $y = \sin(x) + 0.2\sin(2x) + 0.4\sin(4x)$. Notice that the errors on the test set are smaller than on the validation set which could be because there is more training data available to better refine the coefficients. Although all models have been applied to the validation and test set, in practice only the best performing models may be carried forward to forecast on the test set, especially if the models are computationally expensive to train.

There is several other cross-validation methods which split the data in different ways. These are illustrated in Fig. 8.4. In the middle is the Time-Series split as illustrated in detail above but also shown are the **blocked** split (left) and **shuffled** split (right). The blocked approach splits the data into test blocks (typically of the same size) however, unlike the Time-Series split, this split uses data before and after the test block to train the data. The shuffled split, uses random samples to generate the test and training sets.

The blocked and shuffled splits have the advantage of utilising more data with which to train the models but for time series problems this may be less realistic since data is not typically available after the period of interest. Hence these approaches may be more appropriate when considering cross-sectional models or for a time-series model when data availability is quite limited and it is difficult to properly train using the time-series split.

Finally it is worth noting, that if there is insufficient data available, it may be that no choice of splits will be appropriate to create an accurate forecast. For example consider day or week ahead forecasts for hourly time series data which is known to have annual seasonality (e.g. as is usually the case with electricity demand). If there is only one year of data available it will be unlikely you could train a model which will be able to accurately capture the annual seasonality. Even if there is two years of data this could still be difficult since one of those years could have been a particularly unusual year and may not be representative of a typical year which the forecaster is trying to model (for example, consider the 'Beast from the East' an unusual cold wave occurring in Great Britain in 2018). However, it obviously may not be known ahead of time whether there is insufficient data for an accurate forecast and may only become apparent as more testing is performed and more data becomes available.

8.2 Training and Selecting Models

Once the data has been pre-processed (Sect. 6.1), features to include have been decided (Sect. 6.2.2) and the initial forecast model(s) are chosen (See Chap. 9), the parameters/coefficients of the models must be trained and the final models selected. This is often done via cross-validation (see Sect. 8.1.3). This section dives deeper into how to train a forecast model as well as further techniques for selecting accurate models.

The performance of the predictions on the instances in the training set are called the training error. Reducing this training error is typically one part of the optimisation task usually solved by gradient descent-type methods (Sect. 4.3). However, what separates machine learning from simple optimisation is that a **generalisation error** should be minimised as well. The generalisation error is defined as the expected value of the error on *new input*, ideally from the distribution of inputs we expect the algorithm to encounter in practice. To simulate this, in the process of training a model and tuning its hyperparameters, the generalisation error is estimated by assessing its performance on a test set of examples that were collected separately from the training set. When creating a test set (or several test sets as in cross validation as discussed in more detail in Sect. 8.1.3) the following so called i.i.d. assumptions are made:

- the examples in each dataset are **i**ndependent from each other,
- the training set and test set are **i**dentically **d**istributed.

Under these assumptions one can expect, that the expected training error is equal to the expected test error for a *random* model. However, in practice since the parameters of a model are chosen to minimise the training set error, the expected test error is larger than (or equal to) the expected value of the training error. Thus in order to achieve an accurate model that generalises well, a model should minimise the training error, but at the same time minimise the gap between the training and test set error. Figure 8.5 illustrates the typical relationship between training and generalisation error against model capacities (i.e. model complexity).

This suggests another way to understand the bias-variance trade-off alluded to in Sect. 8.1.2. If a model is not able to sufficiently minimise the training error then this corresponds to a model which **underfits** the data/observations. Alternatively, if the gap between test and training error is too large, it is **overfitting** the data. These situations are illustrated in Fig. 8.5. Generally, one can control over- and underfitting by trading off variance and bias and this can be determined using cross-validation methods as described in Sect. 8.1.3 but also regularisation methods as described in Sect. 8.2.4.

There are several ways a particular model can be trained but some models typically use specific approaches. For example, linear regression models (Sect. 9.3) will often use least-squares estimation, whereas artificial neural networks (Sect. 10.4) will often be solved via back-propagation techniques. When implementing a particular forecast model in a standard programming package they will be trained based on those techniques which have been shown to be most suitable or typical for that method. To

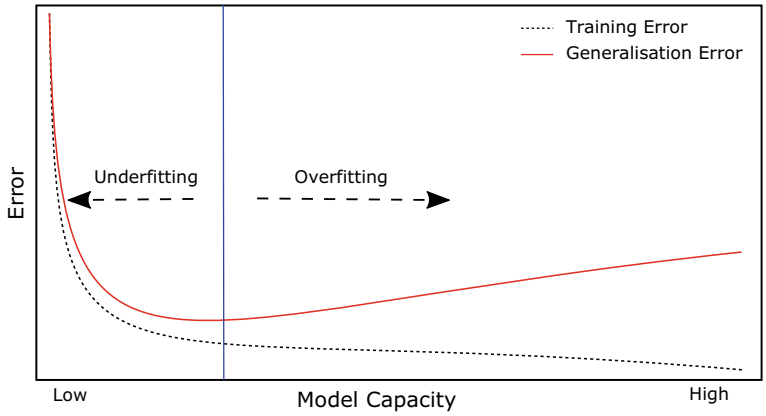


Fig. 8.5 Typical relationship between model capacity (e.g. complexity, or number of parameters) and train and generalisation error

illustrate some of the principles of training a forecast model the following sections consider some common techniques such as least-squares estimation and, maximum likelihood estimation, but also describes some more principles and strategies to control the generalisation error of machine learning models, in particular **regularisation** techniques. Regularisation refers to strategies and model modifications that intend to reduce the generalisation error (but not the training error). In practice multiple strategies are combined.

8.2.1 Least-Squares and Maximum Likelihood Model Fitting

The general aim of training will be to find a good fit between the model and the observations. However, as noted in the previous sections if too many parameters are chosen then the model may overfit on the training set and not generalise very well to the test set (or any other unseen data). A model with features that have been selected appropriately will have a good trade-off between bias and variance and perform better on the test set (see Sect. 8.1.2).

What is a deemed a 'good' fit is relatively subjective but requires a consistent measure of the difference between the observation and models. The best choice is often a balance between practical considerations and what is most appropriate for the application being considered (for example the control of storage devices, as in Sect. 15.1), and therefore models should be evaluated and tested accordingly. Good candidates for such measures are the p-norms introduced in Chap. 7.

As discussed in the corresponding evaluation section (Chap. 7), the choice of p can change the focus of the errors, with larger p values meaning the final error score is more representative of the larger errors compared to p-norms with smaller p.

Further, some norms, such as the 1-norm are not differentiable, which make it more difficult to train the optimal parameters compared to differentiable functions which can be optimised by gradient methods (see Sect. 4.3). The 2-norm is differentiable and is therefore often used as the basis of parameter estimation, in particular it is the core measure used in so-called **least-squares estimation**.

Least-squares estimation (LSE) is one of the most common methods for training parameters, especially for linear regression models. The aim is to minimise the square of the residuals. Recall the residuals are the difference between the real observation, say Y_k, and the model estimate, say $\hat{Y}_k = f_k(Z, \boldsymbol{\beta})$ where Z represents the input variables and $\boldsymbol{\beta}$ are the set of parameters for the model being estimated. The least squares problem can be written

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta} \in \mathcal{B}} \sum_{k=1}^{N} (Y_k - f_k(Z, \boldsymbol{\beta}))^2, \tag{8.5}$$

where $r_k = Y_k - f_k(Z, \boldsymbol{\beta})$ are the residuals. The term arg min simply means finding the *arguments* (parameters) over some feasible set of values (in this case represented by $\mathcal{B}$ and defines the space of reasonable values that $\boldsymbol{\beta}$ could take) which minimises the equation $\sum_{k=1}^{N} (Y_{n+k} - f_{n+k}(Z, \boldsymbol{\beta}))^2$. The process is illustrated in Fig. 8.6 which shows the best fitting line to a set of observations. The vertical distance between the observations and the line shown on the plot are the residuals for the model

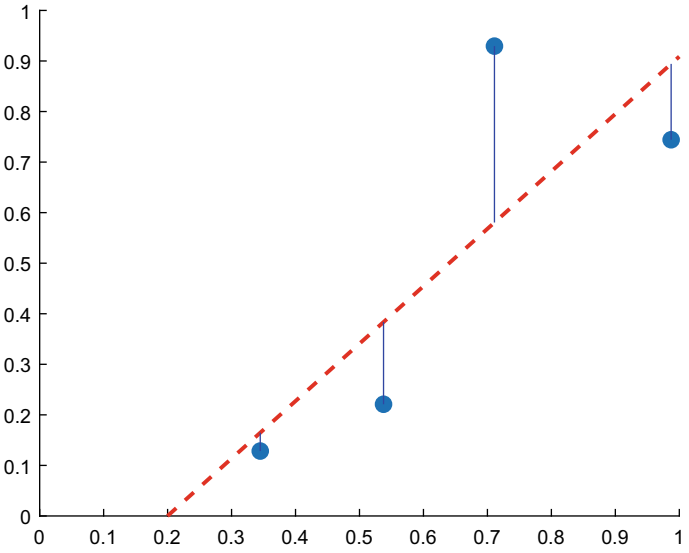


Fig. 8.6 Illustration of least squares estimation for fitting a one variable linear model (dotted red line) to four data points (blue dots). The least squares fit minimises the sum of the squares of the vertical residuals

and the aim in least squares estimation is to minimise the sum of the *square* of these residuals. An advantage of the least squares method is that it is relatively easy to solve, especially for linear models (i.e. those which are linear combinations of parameters—$\sum_{k=1}^{N} \alpha_k \phi_k(x)$ where the α_k's are constant coefficients/parameters to be found), this is because the least squares problem in Eq. (8.5) is differentiable and hence can be solved by differentiating with respect to each element in the set of parameters $\boldsymbol{\beta}$ and setting to zero.

One of the most important statistical methods for training model parameters is by **maximum likelihood estimation** (MLE). This involves setting the model errors within a probabilistic framework which allows further statistical analysis and application of further methods (for example, in Sect. 8.2.2 it will be shown how this enables a method for model selection). The aim of MLE is to create a likelihood function, based on a density function describing the distribution of errors of the model fit. Hence maximising this function finds the *most likely* parameters which minimises the error given the assumed distribution of values. The resultant parameters are called maximum likelihood estimates.

The following discussion will require some basics on univariate probability distributions. To illustrate MLE consider a basic case where the errors follow a normal distribution (See Sect. 3.1), with mean $zero$ and fixed standard deviation σ. Assume finite observation data given by $Y_1, \ldots, Y_N$ and a model $f_k(Z, \boldsymbol{\beta})$ which aims to approximate these observations. The errors are given by the residuals $r_k = Y_k - f_k(Z, \boldsymbol{\beta})$ as before and the probability model can be written as

$$P(r_k, \boldsymbol{\beta}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(r_k - 0)^2}{\sigma^2}\right), \tag{8.6}$$

where the zero is written explicitly to illustrate the standard Gaussian distribution format. The $\boldsymbol{\beta}$ (which in this case simply is the standard deviation σ) is also included to show the dependence of the value on the parameters in the model. The likelihood function is in fact the probability of the model with all observations, in other words

$$L(\beta; Y_1, \ldots, Y_N) = \prod_{k=1}^{N} P(r_k, \boldsymbol{\beta}) = \frac{1}{(\sqrt{2\pi}\sigma)^N} \prod_{k=1}^{N} \exp\left(\frac{-(r_k)^2}{\sigma^2}\right). \tag{8.7}$$

Often it is difficult to solve the likelihood directly, so instead the loglikelihood is solved, which is just the log of the likelihood

$$log(L(\beta; Y_1, \ldots, Y_N)) = -\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{k=1}^{N} r_k^2. \tag{8.8}$$

Since the log function is a strictly increasing function, maximising the likelihood is equivalent to maximising the loglikelihood. Further since in this case the first term $\frac{N}{2} \log(2\pi\sigma^2)$ is constant then this is equivalent to minimising the term $\sum_{k=1}^{N} r_k^2$, in

other words, solving the least squares problem in Eq. (8.5). Of course this is just for a very specific situation where the distribution is assumed to be Gaussian but more complicated distributions can also be considered but they are more difficult to solve analytically.

Both the least squares and maximum likelihood function are examples of **cost functions** which provides a cost between the observed data Y_k and the model estimates $f_k(Z, \boldsymbol{\beta})$. Standard p-norms type errors such as RMSE and MAPE (See Chap. 7) can also be used to define basic cost functions. The cost functions can be very general with different weightings and/or structures which can force the model to fit to different features of the data. For example, as shown in Chap. 7 the pinball loss score can be used as a cost function to produce a quantile estimate whereas using the least squares can produce an estimate of the expected value. The aim, as with the least squares and MLE is to optimise the parameters $\boldsymbol{\beta}$ of the model to achieve the optimal value of the cost function. A common way to find the optimal fit is through gradient methods which were introduced in Sect. 4.3 and these are commonly deployed when fitting machine learning models.

As discussed in Sect. 8.1.2, overfitting the model to the data will produce large generalisation errors. One way to avoid this is to use cross-validation and use a validation set to choose models which don't overfit (Sect. 8.1.3). However, in some cases alternative techniques are often employed to reduce the effect of overfitting a forecast model. These are explored in the following sections.

8.2.2 Information Criterion

The likelihood framework introduced above facilitates the use of information criterion for model selection and is particularly useful for selecting the model orders for ARIMA methods (Sect. 9.4) and for selecting amongst linear regression models. Consider the likelihood function, L, e.g. such as that as in Eq. (8.7) when the residuals are assumed to be Gaussian distributed, evaluated at the maximum likelihood estimated parameters $\hat{\boldsymbol{\beta}}$ which in turn provides the maximum likelihood value $\hat{L}$. In this case the best fit can be framed in terms of the information they provide through the so-called Information Criterion, the most famous examples are the Bayesian Information Criterion (BIC) defined as

$$BIC = M.\ln(N) - 2\ln(\hat{L}) \tag{8.9}$$

and the Akaike Information Criterion (AIC) defined as

$$AIC = 2M - 2\ln(\hat{L}), \tag{8.10}$$

where M is the number of parameters in the model, and N is the number of observations. Both of these create a cost function which is a mix of the loglikelihood (a measure of the fit between the data and the model) and a penalty term which penalises

the number of parameters in the model. Hence by optimising the information criterion a balance is made between a good fit and the number of inputs in the model, and reduces the chance of overfitting the model to the data. Such a model is called **parsimonious**. As can be seen from the equations the AIC penalises the likelihood more than the BIC and therefore typically optimises models with lower number of parameters. For predictive modelling the AIC has been suggested to be more appropriate than the BIC [1]. Note there are several other information criterions but the BIC and AIC are the most common.

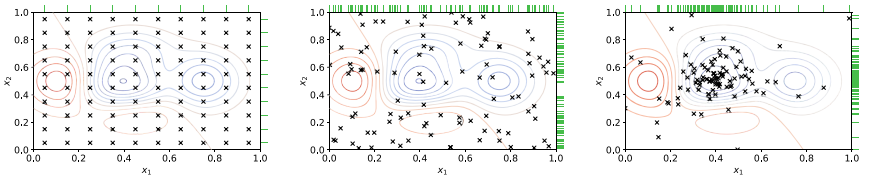
For models which can be set within a likelihood framework, information criterion methods are often used instead of a validation set (Sect. 8.1.3), especially where there is insufficient training data available. However, if there is sufficient amount of data, and a test can be created which sufficiently represents a reasonable sample of real observations, then cross-validation may be preferable.

8.2.3 Hyper-Parameter Tuning

The main objective of optimising a machine learning model is to find an optimal set of parameters, like the weights of a neural network, or the coefficients in a linear regression. However, some parameters have to be chosen that influence the optimisation itself, like the different parameters that have been introduced in other sections like step size, batch size, activations functions and regularisation parameters (Sects. 8.2.4 and 8.2.5). Additionally, models may introduce even more parameters like the architecture of a neural network (number of layers and number of neurons per layer), or the order of the polynomials in a polynomial regression. Such parameters are called **hyperparameters.**

In Sect. 8.2.2 information criterion was shown to be a way to select the optimal order for linear models. For example, by selecting the model with the minimum AIC (or BIC) a more parsimonious model can be chosen which does not sacrifice the model fit. However, information criteria models are not applicable to other types of models such as neural networks or tree-based models.

For simpler models with few hyperparameters, a common approach is to exhaustively search the best configuration among a grid of sensible parameters within each dimension. Real-valued parameters are typically sampled linearly or logarithmically across the feasible values. However, parameters can also be categorical or binary. This approach is referred to as **grid search**. However, this approach is impractical when tuning many hyperparameters of a large neural network, where it may take several hours or even days and weeks to train. Here, one can randomly sample from each dimension of the hyperparameters. This approach is called **random search**, which has been shown to be superior to grid search, especially when only a small number of hyperparameters affect the model performance, i.e., the optimisation problem has a low intrinsic dimensionality. Random search has a further advantage that it is an **any-time** algorithm, as one can stop the algorithm after a specific calculation budget (i.e., a specific number of draws or computation time) is reached. The best solution



(a) Grid Search (b) Random Search (c) Bayesian optimisation

Fig. 8.7 The exemplary performance of a model based on two hyperparameters. For each hyperparameter, ten different values are evaluated and compared. Blue contours indicate regions with strong results, whereas red ones show poor results. *Source* Alexander Elvers, CC BY-SA 4.0

found during the search is then selected. It is also straightforward to parallelise. By choosing a specific distribution, one can also include prior knowledge to help focus the search.

Since hyperparameter selection is an optimisation problem, different general optimisation methods, including meta-heuristic methods, such as evolutionary algorithms and other population-based algorithms, can be used. A popular and successful class of such algorithms are **Bayesian optimisation** approaches. Bayesian optimisation builds a probabilistic model of how hyperparameter values map to model performance determined over a validation set. The probabilistic assumptions are iteratively updated to include more and more information about the search space as it becomes available. Algorithms differ in how they balance the **exploration** of hyperparameters for which the outcome is uncertain and **exploitation** is increased by sampling hyperparameters that are expected to be close to the optimum. Figure 8.7 shows examples of how grid search, random search and Bayesian optimisation sample the search space. Here one can see that Bayesian optimisation does explore the search space more effectively. There is a denser cluster of observations close to the true optimum, while fewer points are sampled in regions that perform poorly. This can generally lead to better or similar results than grid and random search but in quicker time.

8.2.4 Weight Regularisation

Much like the information criterion presented in Sect. 8.2.2, regularisation also uses a penalty based on the number of parameters to try and reduce overfitting and, as will be seen with LASSO, can also be used for feature selection. The regularisation methods presented here are typically applied to linear regression and artificial neural networks (Sect. 10.4) models, although the principles can be generalised to any cost function.

The principle is best demonstrated with an example. Consider a linear model for $n \geq 1$ input/dependent variables $X_{1,t}, X_{2,t}, \ldots, X_{n,t}$ which are assumed to be linearly related to the load L_t at time t. This can be written

$$L_{N+1} = \sum_{k=1}^{n} \beta_k X_{k,N+1}. \tag{8.11}$$

For compactness, the linear model $\sum_{k=1}^{n} \beta_k X_{k,N+1}$ can be written as the matrix-vector multiplication $\mathbf{X}\boldsymbol{\beta}$ where $\mathbf{X} \in \mathbb{R}^{N \times n}$ is the matrix where the ith row and jth column corresponds to the ith time step for the jth variable, i.e. $X_{j,i}$. Finally, $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)^T \in \mathbb{R}^n$ is the vector of parameters. Given a vector of dependent variables $\mathbf{L} = (L_1, L_2, \ldots, L_n)^T$, a regularised least squares regression can be written as

$$\min_{\boldsymbol{\beta}}(||\mathbf{L} - \mathbf{X}\boldsymbol{\beta}||_2 + \lambda||\boldsymbol{\beta}||_p) \tag{8.12}$$

for some hyperparameter $\lambda \geq 0$ (also known as a Lagrangian). Recall from Sect. 7.1 that $||.||_p$ represents the p-norm. This hyperparameter must be found via the validation set as defined in Sect. 8.1.3 and controls the size of the penalty on the coefficients. If $\lambda = 0$ then the problem reduces to the standard least-squares estimation, Eq. (8.5). For large values of λ the parameters become small. The most common forms used are either $p = 1, 2$ the so-called **LASSO** (least absolute shrinkage and selection operator) or ridge regression respectively. LASSO is particularly popular at it can reduce the number of inputs as it will often set many of the coefficients in a linear regression to zero due to the 1-norm penalty. In this case a large number of unimportant inputs can be eliminated. Hence LASSO can be used for both training a model and feature selection.

To understand why LASSO can be used for feature selection consider the illustration in Fig. 8.8. The figure compares the ridge regression with the LASSO regression. Both plots show the contours (lines of the same value) of the least squares cost function within the parameter space $\boldsymbol{\beta} = (\beta_1, \beta_2)^T$ (assuming only 2 dimensional problem). In the centre of these contours is the least squares estimate $\hat{\boldsymbol{\beta}}$, i.e. the parameter values which gives the smallest values of the least squares cost function.

In LASSO or ridge regression, there is effectively a penalty on the size of the parameters and this penalty changes based on the size of the λ hyperparameter. The larger the λ the smaller the values of the parameters. Thus the parameters are constrained within a bounded area of size $0 < C \in \mathbb{R}$ given

$$||\boldsymbol{\beta}||_p = (|\beta_1|^p + |\beta_1|^p)^{1/p} \leq C, \tag{8.13}$$

where $p = 1$ or 2 depending on whether LASSO or Ridge regression is being considered. These constrained regions are shown as the shaded areas in Fig. 8.8. Notice that due to the different norm values used the shapes are very different. The constrained region in the 2-norm (Ridge regression) is a spherical ball, but with the

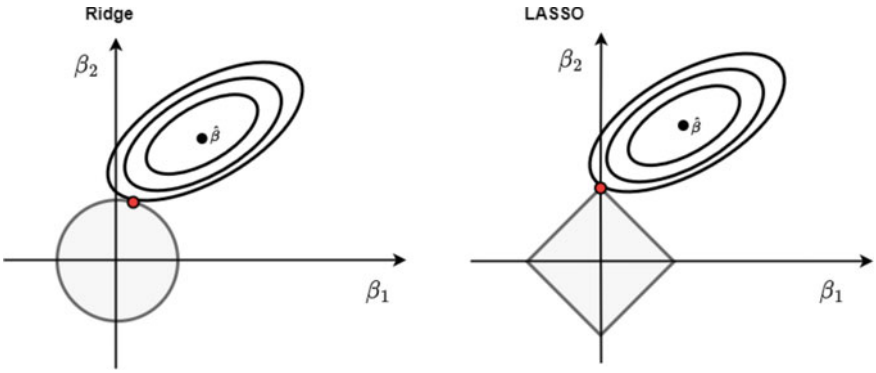


Fig. 8.8 Demonstration of how LASSO regularisation (right) can be used to select features. This is compared to ridge regression (left). The chosen parameters are represented by the red dot showing the smallest value of the least-squares cost function with respect to the feasible parameters (the shaded shape). The figure is explained in the main text

1-norm (LASSO) is a square. The minimum value for the regularised cost function is therefore given by the $\boldsymbol{\beta}$ within the shaded feasible region closest to the optimal least squares estimate $\hat{\boldsymbol{\beta}}$, shown as the red dot in the figure.

Due to the shape of the constrained region for LASSO regression this will often lie on the 'corners' of this region, which means that often many of the parameters will be equal to zero, effectively selecting the parameters in the process.

8.2.5 Other Regularisation Methods

Reducing Model Capacity For many algorithms one can control with hyperparameters whether a model is more likely to over- or underfit by altering its capacity, i.e., its complexity or more generally its available degrees of freedom. By choosing certain hyperparameters (see Sect. 8.2.3) the hypothesis space, i.e., the set of functions that the algorithm is capable of selecting as a solution, is affected. For instance, for neural network models, the number of trainable parameters determines the ability to fit a wide variety of functions. Similarly, in random forests, the number of trees determines its complexity. Models with low capacity may not be able to properly fit the training set (they have high bias). Models with high capacity will overtrain on the training data set and don't generalise to the actual underlying process (hence are not represented in the test set).

Data Augmentation Overfitting can generally be avoided by training a model on more data. In practice the amount of data may be small or data collection can be expensive. Instead, one way to improve model performance is to create synthetic data samples in the training set. For image data this can often easily be achieved by adding transformations to images like mirroring, rotating, shifting, as problems

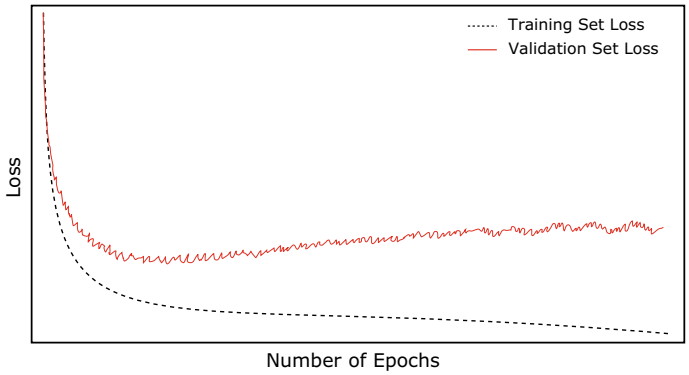


Fig. 8.9 Exemplary learning curve to show relationship of training and evaluation loss over time in training process for a model with high capacity

such as object detection are expected to be invariant to such operations. This is not trivial for many machine learning tasks, in particular time series forecast where the real patterns and interdependencies in the data need to be preserved. However, where possible, data augmenting strategies should be explored.

Early Stopping Training neural network models with large capacity on tasks which are too simple can lead to overfitting. One popular diagnostic tool to prevent this are **learning curves**, i.e., the calculation of the error on the training set at regular intervals in the training process (e.g., after each training epoch). If the hyperparameters are chosen reasonably well the training error should decrease. To monitor generalisation in the training process, one can create a validation dataset and calculate the errors. If the validation error increases, while the training error decreases this is an indication of the model starting to overfit. Figure 8.9 gives an exemplary learning curve of a model with high capacity. Thus one popular regularisation strategy is to stop training if the validation error does not improve beyond a specific number of iterations. At the end of the process, the model that has the smallest validation error is returned rather than the final model configuration. This requires the algorithm to store checkpoints of the model configurations during the training process (e.g., the values of the weights in a neural networks).

Batch Normalisation Another popular improvement to the training process of neural networks is batch normalisation, or batch norm. It was introduced as a method to speed up the training of neural networks and make it more stable by normalising each of the layers' inputs by re-centering and re-scaling (standardising). However, besides providing faster and more stable training, batch normalisation also has a regularising effect. Further, the training becomes more robust to different initialisation schemes and the choice of the learning rates (i.e., a larger learning rate can be chosen).

Dropout In dropout a certain share of artificial neurons and its weights are randomly omitted during the training process of a neural network (Sect. 10.4). This

process effectively creates an ensemble of simpler neural network architectures. This is related to ensemble methods such as random forests that combine the predictions of simple decision trees (see Sect. 10.3.2 on random forests). Dropout has the effect of adding noise to the training process. It has been shown that a reasonable default for a wide range of tasks is to use a dropout of 0.5 for each layer. Dropout can be used and configured for each layer of the neural network, and works with different kinds of layers such as dense fully connected layers, but also convolutional and recurrent layers (see Sects. 10.5 and 10.4). However, it should not be used in the output layer. When adding dropout only to the input layer, this is related to the idea of adding noise as it has been used in denoising autoencoders. It is computationally cheap and an effective regularisation method to reduce overfitting and improve the generalisation error in many kinds of deep neural networks.

8.3 Questions

1. Create your own bias variance experiment. You could repeat the polynomial fit in Fig. 8.1. Alternatively choose another polynomial of a different degree. Generate 100 samples from the polynomial and add noise (say with a Gaussian distribution). Now fit polynomials of a variety of degrees, say from 1 to 20. Calculate the training errors. Plot the training errors for each polynomial as a function of degree. How does the error change? Is the smallest error at the correct polynomial degree? For higher degrees does the error increase? Now resample the polynomial (and add noise with the same distribution as before). Measure the error between the fitted polynomials and this new data? This is the generalisation error. Plot the errors against degree again. What is the optimal degree? Compare this plot with the original one with the training errors. What is the difference between them?
2. Repeat the above experiment but sample just 15 points this time. How do the training and generalisation errors change? What about reducing the number of sampled points to 5?
3. Perform your own grid search. Generate points from a simple model, say a line $y = ax + b$, with known coefficients a, b. Add a small amount of noise to the points. Select a rectangle around the coefficients, i.e. $(a, b) \in [A_1, A_2] \times [B_1, B_2]$. Generate a grid of $N_1 \times N_2$ points (say $N_1 = N_2 = 10$) within this rectangle by simply choosing uniformly spaced values on each side of the rectangle, i.e. the kth a value $a_k = A_1 + (k-1)\frac{(A_2-A_1)}{N_1-1}$, similarly $b_l = A_2 + (l-1)\frac{(A_2-B_2)}{N_2-1}$ for $k = 1, \ldots, N_1$, and $l = 1, \ldots, N_2$. For each pair of coefficients in the rectangle calculate the errors between the sampled data and the associated line. Which pair of coefficients give the lowest errors? How close are they to the true values? In addition, sample random pairs from within the rectangle. How many samples did you need to produce smaller errors than the grid search.
4. Show that the mean squared error for a model can be broken down in bias, variance and irreducible error as in Eq. 8.3.

Reference

1. G. Shmueli, To explain or to predict? Stat. Sci. **25**, 289–310 (2010)

Chapter 9
Benchmark and Statistical Point Forecast Methods

The previous sections have described in detail the steps required to develop a time series forecast including: how to generate useful explanatory variables; how to train the model; how to avoid overfitting; and how to evaluate the accuracy of the model. What has not been investigated is the models themselves. This chapter will be the first of three chapters looking at a wide range of models and some of their properties.

This chapter and the next will look at point forecast methods, and then in Chap. 11, probabilistic forecasts will be examined which provide models for handling highly uncertain data, something which is often required for low voltage feeders and substations (Chap. 2).

Of the point forecasting chapters, this chapter looks at traditional statistical methods, whereas Chap. 10 will look at what are sometimes referred to as machine learning models. Each type of models has advantages and disadvantages, some of which have already been described in Sect. 5.3, but further criteria will be described in Sect. 12.2. In short, statistical models are typically more transparent and easier to interpret and understand. That makes them not only useful for investigating some of the core features of the data, but also makes them good benchmark candidates.

The majority of the models presented in this chapter are easily implemented through packages in open source programming languages for scientific computing such as Python and R as well in popular proprietary software such as MATLAB. However they can be easily derived and trained from scratch (since they are often linear functions and hence can be easily trained using e.g. linear least squares, see Sect. 8.2), which may be preferable when you want to extend the models or make bespoke adjustments.

This chapter starts by considering some simple models and then introduces progressively more complicated ones (in terms of more parameters and computational expense) starting with exponential smoothing (Sect. 9.2), multiple linear regression models (Sect. 9.3), ARIMA and SARIMA models (Sects. 9.4 and 9.5 respectively), and then finally generalised additive models (Sect. 9.6).

Before diving into the models it is worth highlighting the context for these forecasts: **short term load forecasts** (STLF). A common way to categorise a load forecasts is in terms of the forecast horizon. Short term forecasts estimate the demand

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_9

between a day and a week ahead (sometimes two weeks). In contrast, those from 1 week up to a year are referred to as **medium term load forecasts**, and those beyond a year, **long-term load forecasts**. Note these definitions can vary slightly depending on the context but are typically in the ranges specified. Models which are good for STLF may not be suitable for medium and long term load forecasts, and vice-versa. Hence the models presented here are specifically chosen for their use in shorter term load forecasts which will usually heavily rely on the most recently observed information.

9.1 Benchmarks Methods

This section will begin with considering basic and commonly used benchmark methods. As discussed in Sect. 8.1.1 developing appropriate benchmarks is essential for any well-designed forecast experiment. As usual throughout this chapter a time series of the form $L_1, L_2, \ldots,$ will be considered and the aim is to produce estimates $\hat{L}_{N+k}$ for the time steps $N + k$ where $N \in \mathbb{N}$ is the forecast origin and $k \in \mathbb{N}$ the forecast horizon (See Sect. 5.2 for further details on these terms).

One of the simplest benchmarks is the persistence model, which can be described as

Persistence: $\hat{L}_{N+1} = L_N$

This model assumes that the demand of the next time step is simply the current load. Additional future time steps can be estimated by simply repeating this value.

This can be an effective model if the data has a single strong autocorrelation at lag one (see Sect. 6.2.2). However for most applications, with more variable demand, the method will provide very little accuracy. Instead, energy demand often has strong daily, weekly and annual seasonal components (Sect. 5.1). Hence, effective adjustments can be made to the simple persistence model to produce a much more accurate forecast model which presumes that the behaviour at the current step is the same as that at exactly one seasonal cycle away. These are called seasonal persistence models and have the following form:

Seasonal persistence: $\hat{L}_{N+k} = L_{N+k-s_1}$

where $\hat{L}_{N+k}$ is the k-step ahead prediction, N is the forecast origin, while s_1 denotes the seasonal period (note it is assumed the last seasonal point is observed, i.e. occurs before the forecast origin, so $N + k - s_1 \leq N$). As an example, consider the case of half hourly data, seasonal persistence models for daily, weekly and yearly seasonality can be produced by setting s_1 to 48, 336 or 52×336 respectively. These seasonal persistence forecasts are very easy to implement and require no training data whatsoever. An example of day ahead persistence and seasonal persistence models are shown in Fig. 9.1, were the seasonal persistence model uses daily seasonality (i.e. yesterday is the same as today).

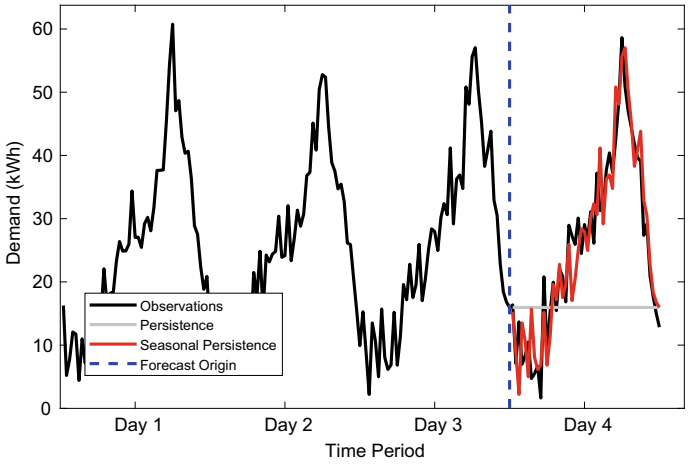


Fig. 9.1 The plot shows a simple persistence forecast (grey flat line) and a daily seasonal persistence (red line) for the fourth day of the half hourly data. The observations are shown as a black line. The data in this example has daily seasonality and hence the seasonal persistence model picks up important features of the data

For seasonal data, an extension (and usually an improvement) to these methods is to include several historical observations at the same period and take a simple seasonal moving average. In other words:

Seasonal Moving Average (SMA): $\hat{L}_{N+k} = \frac{1}{p}\sum_{i=1}^{p} L_{N+k-is_1}$.

As with the seasonal persistence, often a weekly period ($s_1 = 336$ for half hourly data) is used. The weekly simple averages often perform much better than the equivalent seasonal persistence models since it smooths out the random week-to-week aberrations around the expected value and therefore better replicates the typical weekly behaviour. An example is illustrated in Fig. 9.2 for daily seasonal data. Day 3 had unusually large demand, hence a daily seasonal persistence model would not be as accurate as when it was used in the previous example in Fig. 9.1. Instead the simple average over the Days 1–3 reduces the effect of the unusual day 3 and hence provides a better estimate of day 4. For the simple average method, slightly more training data is required than the persistence models, and in addition a validation period is required to choose the most appropriate value of the hyperparameter p (Sect. 8.1.3). However, the model is very quick to calculate and in practice typically only requires setting $p = 4$ or 5 weeks to optimise the model and offer significant improvements over the persistence models.

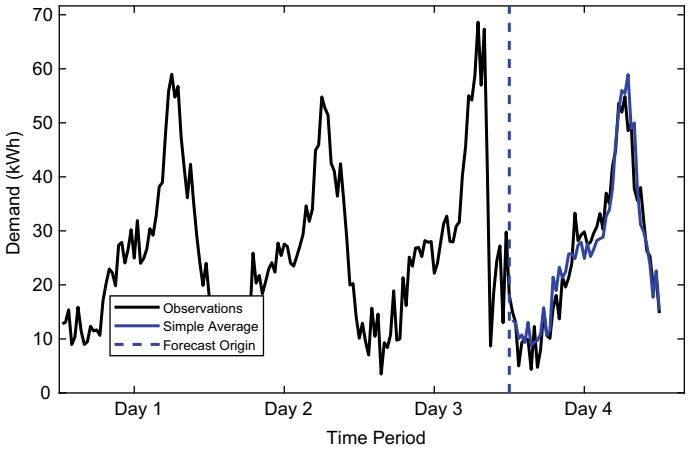


Fig. 9.2 Plot shows a daily seasonal moving average forecast (blue line) over three historic days to generate a forecast for the fourth day of the half hourly data. The observations are shown as a black line. The data in this example has daily seasonality but more volatile data on the third day of the data. Hence in this case the daily seasonal persistence would not produce an accurate forecast for the fourth day compared to the simple moving average which smooths out the errors over the three days

9.2 Exponential Smoothing

Despite their simplicity, the benchmarks introduced in Sect. 9.1, especially the seasonal moving average, can be surprisingly accurate. However, one of their disadvantages is that each historical week they utilise is given equal weighting whereas it would be expected that older data is less relevant to the current forecast period. In other words, older data should contribute less than more recent data to the final forecast. This is particularly relevant for load data as it is strongly driven by seasonalities and trends. For example, it would be expected that data from a few months ago in, say summertime, is less relevant to the winter period.

Exponential smoothing methods take weighted averages of past observations but where the weights decay for older observations. To illustrate this, consider the simplest form of exponential smoothing which creates a smoothed 1-step ahead output $\hat{L}_{N+1}$ which is updated at each step using the latest observation, L_N, in the following way

$$\hat{L}_{N+1} = \alpha L_N + (1-\alpha)\hat{L}_N = \hat{L}_N + \alpha(L_N - \hat{L}_N), \tag{9.1}$$

where $\alpha \in (0, 1)$ is a smoothing constant to be optimised in the validation period (see Sect. 8.1.3). The estimate, $\hat{L}_{N+1}$, of the next observation L_{N+1} is a weighted average of the current estimate $\hat{L}_N$ and the most recent observation L_N. Similarly, the previous estimate is also a weighted average of the previous observation L_{N-1}

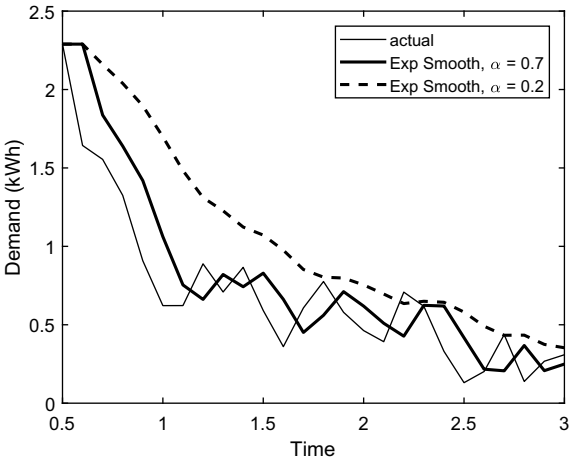


Fig. 9.3 Simple example of exponential smoothing for different values of the smoothing parameter

and the estimate $\hat{L}_{N-1}$ before that, and so on. In other words Eq. (9.1) can be written in the expanded form

$$\hat{L}_{N+1} = \alpha(L_N + (1-\alpha)L_{N-1} + (1-\alpha)^2 L_{N-2} + \ldots (1-\alpha)^{N-1} L_2) + (1-\alpha)^N L_1, \tag{9.2}$$

a geometric sum. Since α and hence $1-\alpha \in (0,1)$ then older observations are given less weight and thus contributes less to the final estimate. In the special case of $\alpha = 1$ then the forecast is simply the last observation and is equivalent to the simple persistence model as given in Sect. 9.1. This method is a 1-step ahead forecast and hence if multiple steps are required the forecasts are fed back into the model in place of the unobserved values. The optimal parameter can be found by a minimisation the sum of square errors for the 1-step ahead forecasts (over the validation period) but in addition to α an initial estimate must also be produced. This can be generated as a simple average over previous values. The sum of square errors is a nonlinear equation due to the nested application of the smoothing constant and therefore has to be optimised using numerical methods rather than being solved directly.

To illustrate the exponential smoothing method consider a basic example given in Fig. 9.3. Two exponential models are applied using two different values of α to produce a 1-step ahead forecast. The model that uses $\alpha = 0.7$ is less smooth and is driven mainly by the most recent points. The model that uses $\alpha = 0.2$ is the smoothest and takes a weighted average which has more contributions from older historical values. In this case less smoothing (higher α value) is more useful for prediction since the data has a decreasing trend and hence older points are much less relevant to the recent data.

In this basic form, exponential smoothing is relatively limited since it ignores trends or seasonalities which are important components of demand. A more advanced exponential smoothing algorithm that does take seasonalities into account is the

Holt-Winters-Taylor (HWT) exponential smoothing method and models two levels of seasonality. This method estimates the load $\hat{L}_{N+1}$ at time t using the following set of equations:

$$\begin{aligned}
\hat{L}_{N+1} &= l_N + d_{N+1-s_1} + w_{N+1-s_2} + \phi e_N \\
e_{N+1} &= \hat{L}_{N+1} - (l_N + d_{N+1-s_1} + w_{N+1-s_2}) \\
l_{N+1} &= l_N + \lambda e_{N+1} \\
d_{N+1} &= d_{N+1-s_1} + \delta e_{N+1} \\
w_{N+1} &= w_{N+1-s_2} + \omega e_{N+1},
\end{aligned} \tag{9.3}$$

where the parameters $\phi, \lambda, \delta, \omega$ must be trained on the historical data. The load is broken down into three core components, a level l_t which corresponds to the first order correlation, and two seasonal terms, d_t and w_t, which in load forecasting often correspond to intraday and intraweek seasonality respectively (although of course different periods can be used depending on the data). The intraday seasonality period s_1 and intraweek period s_2 are the number of time steps covering one day or week, and for hourly data would be 24 and 168 respectively. Notice that each of the level and seasonal terms has their own simple exponential smoothing equation as in Eq. (9.1). The error terms $\epsilon_{N+1} = L_{N+1} - \hat{L}_{N+1}$ are assumed to be normally distributed with zero mean. At each time step $N + 1$, a forecast, $\hat{L}_{N+1}$, is made using the current values for the level and seasonal terms l_N, d_N, w_N as well as the first order error term e_N. Given this new estimates the other terms values can be then updated using their respective smoothing equations as described in Eq. (9.3). Due to the recursive nature of the algorithm the older values contribute less to the updates and the amount of contribution is determined by the size of their respective parameters, ϕ, λ, δ, and ω.

Training the model parameters can be achieved by numerical optimisation of the one-step ahead, sum of squared errors (i.e. Eq. (8.5)) over the training data (Sect. 8.2) as before. However, note that there must be an initial estimate for the level and seasonal components before the parameters can be trained. There a few ways to do this but a simple method is to take an average over the oldest observations to ensure that there is initial data to train the algorithm. An example of the double seasonal exponential smoothing model will be given in the case study in Sect. 14.2.

9.3 Multiple Linear Regression

Standard regression is a statistical process for estimating the relationship between single or multiple variables. One of the simplest and most common of such models is multiple linear regression as it is easy to explain, fast to compute and very versatile. Suppose there is $n \geq 1$ input variables $X_{1,t}, X_{2,t}, \ldots, X_{n,t}$ which are assumed to be

linearly related to the load L_t at time t, in other words the following forecast model is constructed

$$\hat{L}_{N+1} = \sum_{k=1}^{n} \beta_k X_{k,N+1}. \tag{9.4}$$

The coefficients (or regression parameters), ϕ_k, describe the explanatory power of each of the variables in modelling the load L_t (although this does depend on each variable having similar magnitude). The independent variables are assumed to be uncorrelated with each other,[1] as are the 1-step ahead errors $\epsilon_t = L_{N+1} - \hat{L}_{N+1}$ which are often assumed to be distributed as a Gaussian (see Eq. (3.5) Chap. 3) with mean zero and constant variance (the constant variance means the errors are homoskedastic—see Sect. 11.6.1). These assumptions simplify the training of the coefficients and modelling of the prediction intervals. However, as always it is a good idea to check these assumptions by plotting the residuals as well as their ACF (see Sect. 7.5 for further details).

If there is only one explanatory variable then the model is simply called linear regression, whereas with more than one it is called **multiple linear regression**. Multiple linear regression can often be written in a more succinct vectorised form

$$\hat{L}_{N+1} = \boldsymbol{\beta}^T \mathbf{X}_{N+1}, \tag{9.5}$$

where $\mathbf{X}_t = (X_{1,t}, X_{2,t}, \ldots, X_{n,t})^T$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)^T$ are the vectors of independent variables and regression parameters respectively.

Although Eqs. (9.4) and (9.5) only show independent variables $X_{k,t}$ at the same time step t as the independent variable, $\hat{L}_t$, the equations can of course include lagged time points and autoregressive variables.

As an example, consider the situation in Fig. 9.4 where the best regression fit for the observations (in red) is the curve $Y = (X-2)^2 + 1.2 = X^2 - 4X + 5.4$. Notice, that although the function contains a quadratic term X^2, it is still clearly linear in the coefficients with independent variables $\mathbf{X} = (X^2, X, 1)^T$ and corresponding regression parameters $\boldsymbol{\beta} = (1, -4, 5.4)^T$. Hence it is important to understand that *nonlinear* relationships can still be modelled within *linear* regression. For an example in demand forecasting, notice that the nonlinear relationship between demand and temperature in Fig. 6.7 in Sect. 6.2.2 could be modelled by a linear regression using a polynomial (if chosen with sufficient order).

Linear regression is also well suited to model the impact of categorical/discrete variables through the use of dummy variables (see Sect. 6.2.6). This is particularly useful in load forecasting which often require day-of-the-week or time-of-the-year effect. For example, often different days of the week are likely to have different demand characteristics, in which case the model should include the effect of the

[1] If the errors are correlated this makes fitting this model much more complicated and the least-squares estimator for estimating the coefficients may not converge to the correct values. This is beyond the scope of this book, but there are other books such as [1] which dive into this in more detail.

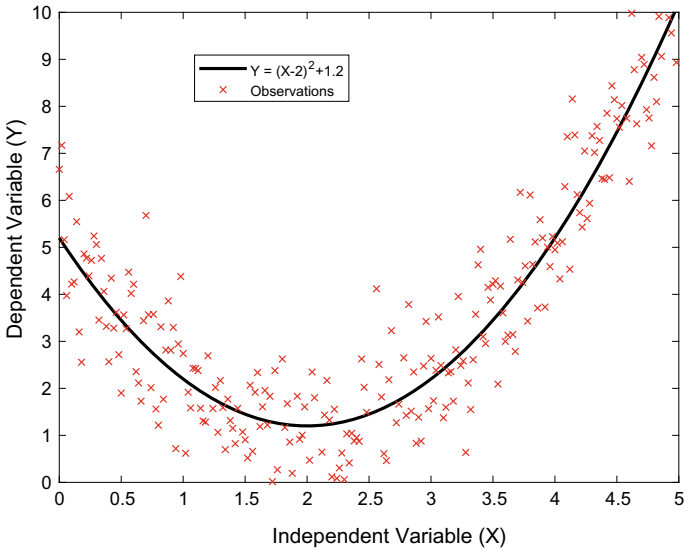


Fig. 9.4 Linear regression line $Y = (X-2)^2 + 1.2 = X^2 - 4X + 5.4$ (black) and noisy, Gaussian observations (red crosses) around the line

different days. In multiple linear regression this is done by including the dummy variables $D_j(k)$ for $j = 1, \ldots, 7$ (one for each day of the week—with Monday represented by $j = 1$ and Sunday by $j = 7$ etc.) which indicate the day of the week, defined by

$$D_j(k) = \begin{cases} 1, & \text{if time step k occurs on day } j \text{ of the week} \\ 0, & \text{otherwise} \end{cases}$$

In a linear regression model we often use six of the dummy variables as inputs to avoid the **dummy variable trap** (see Sect. 6.2.6) since in fact we can model the effect of one of the days by the other six (the seventh day is modelled by simply setting the other six to zero, presuming there is at least another term such as a constant which will ensure that its effect can be modelled).

Another useful feature of linear regression is that we can include interaction terms. This is where we model the effect of two or more variables on the dependent variable. For example, it may be that temperature T_k has an effect on demand, but only for a particular hour of the day, say 2–3 pm. In this case we can include a term for the temperature variable but multiplied by a dummy variable which indicates the time of day and is zero at all times except the hour 2–3 pm. In the linear regression the interaction term is often denoted as multiplication of the two terms, e.g. $T_k D_j(k)$ or $T_k * D_j(k)$. The case is similar if the simultaneous effect from more than two

variables are modelled. An example of interaction terms in a linear regression model will be given in the case study in Sect. 14.2.

Given the assumptions on the errors, the coefficients of a linear regression model are often found by minimising the least squares estimate (see Sect. 8.2) and are therefore quite easy, and quick, to train. Recall, that since the errors are assumed to be Gaussian with constant variance, the least squares estimate of the model are also the maximum likelihood estimate as shown in Sect. 8.2. This is particularly convenient since the loglikelihood (see Eq. (8.8)), and hence the Bayesian information criteria (BIC) and Akaike information criteria (AIC), are both easy to calculate. Recall from Sect. 8.2.2, that identifying the models with the smallest values of AIC or BIC is one way to choose the best models on the training data, which have a tradeoff between accuracy and model complexity helping to limit the potential for overtraining the models.

As described in Sect. 8.2.2, linear models can be easily adapted to regularisation frameworks such as LASSO and ridge regression. Much like the AIC and BIC these techniques penalise the number and/or size of the coefficients by including a penalty term on the normal least squares regression. In particular LASSO can be used as a model selection technique as it tends to set the coefficients of irrelevant (or less influential) explanatory variables to zero. Finally, of course, as with all the methods, the models can also be selected through cross-validation and finding the model which minimises the error on the validation set. This can be quite inefficient if there is a lot of independent variables being considered.

Given the final trained model, the simple linear structure the coefficients provide a useful way to interpret the effect of each variable (assuming they are independent). Essentially they tell you how much the expected value of the dependent variable will change given a unit change in the independent variable assuming all the other independent variables are fixed. The interpretation becomes a little more complex when there are interaction terms as the effect size will now be dependent on the value of the other variable(s). In these cases inserting a range of reasonable values for these other variables may help to show the range of effects.

9.4 ARIMA and ARIMAX Methods

The autoregressive moving average (ARMA) technique is a traditional linear time series model which has been extensively used in time series forecasting. An ARMA (p, q) model for a time series is a linear model described by

$$\hat{L}_N = C + \sum_{i=1}^{p} \psi_i L_{N-i} + \sum_{j=1}^{q} \varphi_j \epsilon_{N-j}, \tag{9.6}$$

where ϵ_t is a time series of error terms and C is a constant. ARMA models depend on the time series L_t to be stationary (see Sect. 5.1) however this may not always be

the case. When the series is not stationary, differencing can be applied to time series until the final series is stationary. If d differences are applied this can be written

$$L_N^{(d)} = L_N^{(d-1)} - L_{N-1}^{(d-1)}, \tag{9.7}$$

where the differences are applied iteratively d times. When differencing is used, the ARMA(p, q) model is now referred to as a ARIMA(p, d, q) model (autoregressive integrated moving average, with the integrated part meaning the differencing) and can be written

$$\hat{L}_N^{(d)} = C + \sum_{i=1}^{p} \psi_i L_{N-i}^{(d)} + \sum_{j=1}^{q} \varphi_j \epsilon_{N-j} + \epsilon_N. \tag{9.8}$$

A convenient and concise way to write ARIMA models is in terms of the Backshift operator B (also known as the lag operator), where B is defined on elements of a time series by $BL_t = L_{t-1}$. By definition the lag operator can therefore be written $B^k L_t = L_{t-k}$. Thus the ARMA(p, q) model can be written as follows

$$\left(1 - \sum_{i=1}^{p} \psi_i B^i\right) L_N = \left(1 + \sum_{j=1}^{q} \varphi_j B^i\right) \epsilon_N + C, \tag{9.9}$$

and an ARIMA(p, d, q) model can be written as

$$\left(1 - \sum_{i=1}^{p} \psi_i B^i\right) (1 - B)^d L_N = \left(1 + \sum_{j=1}^{q} \varphi_j B^i\right) \epsilon_N + C, \tag{9.10}$$

where $(1 - B)^d L_N$ is the dth order difference.

ARIMA models are quite versatile, being able to estimate a wide range of time series. They consistent of three main components: the difference d, an autoregressive (AR(p)) component, $\sum_{i=1}^{p} \psi_i L_{t-i}^{(d)}$, of order p, and a moving average, $MA(q)$, term, $\sum_{j=1}^{q} \varphi_j \epsilon_{t-j}$, of historical white noise error terms of order q. As with multiple linear regression, the error terms are generally assumed to be Gaussian distributed (although other distributions can be used), with mean zero and uncorrelated with each other.

Autoregressive models, AR(p), and moving average models, MA(q), are special cases of ARMA models (actually ARMA(p, 0) and ARMA(0, q) models respectively) and they are worth considering in a bit more detail before looking at the full ARMA model. Autoregressive models are ARMA processes but with $\varphi_j = 0$ for all j and hence these are simple models in which the p past values of the time series influence the current values. An AR(p) can be written

$$\hat{L}_N = C + \sum_{i=1}^{p} \psi_i L_{N-i}. \tag{9.11}$$

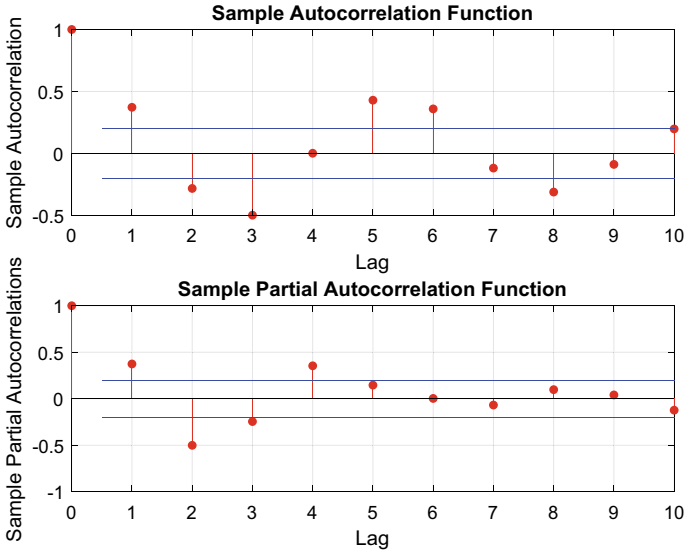


Fig. 9.5 Example of autocorrelation (top) and partial autocorrelation (bottom) for a simple AR(4) model

Recall from Sect. 3.5 the partial autocorrelation is a measure of autocorrelation between the time series and its lagged values but with the influence from the inbetween lags removed. This means the PACF is an natural way to identify the order, p, of an AR process since the PACF should be zero for lags $k > p$. In practice, the order can be detected by considering the sample PACF plot from the available time series and identifying when the lagged values are *effectively* zero, i.e. are consistently within the 95% confidence bounds which are often plotted with the PACF. Further, the ACF should also exponentially decay to zero for an AR(p) process. An example for a simple AR(4) model is shown in Fig. 9.5. Notice that in the PACF there are no correlations which are outside the confidence bounds beyond lag 4 as expected.

In contrast, moving average models are influenced by past values of the error values, so large past deviations can have an influence on the current time series values. One of the useful properties of a pure MA(q) process is that the autocorrelation function should be zero from lag $q + 1$ onwards. So the ACF plot can be used to identify a MA time series and its order. It should be noted that, although the ACF and PACF can be used to identify AR and MA models and their orders, in practice the sample version of these functions are used, applied to real observed data, and hence the results may deviate from the more clear-cut theoretical solutions. In other words the autocorrelations may exceed the confidence bounds but these may be spurious and simply occur due to random chance.

Now consider an ARIMA model, where the optimal orders, p, q and d must be found in order to train the coefficients ψ_i, φ_j of the final model. Typically this done by comparing the AIC values (see Sect. 8.2.2) from a range of different choices for

p, q and d. It is impractical to compare all possible values and hence typically a search is only performed around a good approximation to the orders. A common method for finding the best orders for an ARIMA model to historical time series data is the **Box-Jenkins** method which utilises the ACF and PACF to identify the autoregressive and moving average orders as discussed above. The process typically consists of the following steps:

1. Check if the time series is stationary. If it isn't perform differencing until the final series is stationary. Stationarity can be checked in many ways. In addition to a time series plot, another indication of non-stationary time series is a slowly decaying auto-correlation function as a function of lags (see Chap. 3). However, there are also stationarity tests as outlined in Appendix A.
2. Identify the orders of the autoregressive (AR) and moving average (MA) terms. This can be estimated by examining the autocorrelation function (ACF) and partial autocorrelation function (PACF) plots (see Chap. 3 and Sect. 6.2.4). In particular, if the model has an AR component of order p then the PACF should be effectively zero from lag $p + 1$ and above. Similarly for an MA model of order q, the ACF should effectively be zero from lag $q + 1$ and higher. In practice these orders can be found by looking at the respective plots and considering whether they are above the 95% confidence interval (which are usually included on the plot, see Sect. 6.2.4).
3. Using the ACF and PACF as an approximation for the correct orders, check the AIC (or BIC) values for a selection of ARIMA models with different p, d, q values (around the approximate values). The final orders are those that give the smallest AIC (BIC) values.

It should be emphasised that the ACF and PACF do not often give a definitive answer on the correct orders, and hence in practice they are used to approximate the correct orders which are then tested in step 3 using the AIC/BIC.

The Box-Jenkins methodology is illustrated here for a specific example using an ARIMA(3, 0, 1) (or equivalently an ARMA(3, 1)) model given by $y_t = 0.14 + 0.609y_{t-1} - 0.5y_{t-2} + 0.214y_{t-3} + 0.624e_{t-1} + e_t$. The time series is shown in Fig. 9.6 and was generated using the Matlab *simulate* function.[2] The e_t is the error series which are distributed according to the standard normal distribution. In this case the series is stationary so there is no differencing required. To check the autoregressive and moving-average orders the ACF and PACF plots are considered, these are shown in Fig. 9.7, together with the confidence bounds for the 95% significance level. The ACF (the top plot) indicates the MA order and shows that the largest correlation is at lag 1, which is as expected, however there are also significant correlations (significant in terms of being clearly outside of the confidence interval) at lags 16 and 17. Notice that the ACF doesn't gradually decrease as a function of lag, this supports the conclusion that the time series is stationary. The PACF indicates the AR order and in this example shows there are significant peaks at lags 1–4 which suggest a slightly larger order than expected. In addition there are smaller peaks outside the

[2] https://uk.mathworks.com/help/econ/arima.simulate.html.

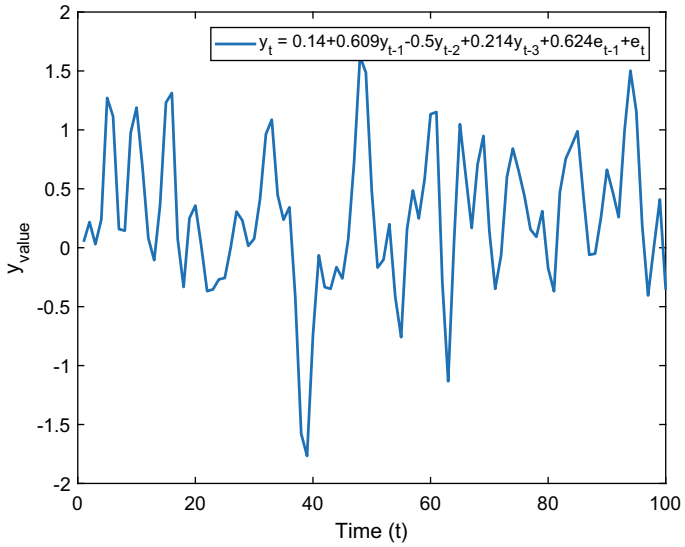


Fig. 9.6 Example time series, generated from the ARIMA$(3, 0, 1)$ model $y_t = 0.14 + 0.609y_{t-1} - 0.5y_{t-2} + 0.214y_{t-3} + 0.624e_{t-1} + e_t$

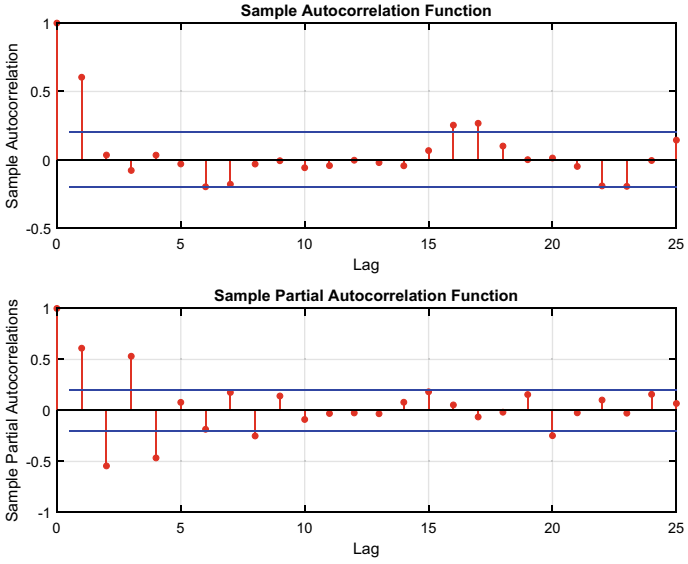


Fig. 9.7 ACF (top) and PACF (bottom) plot for the time series $y_t = 0.14 + 0.609y_{t-1} - 0.5y_{t-2} + 0.214y_{t-3} + 0.624e_{t-1} + e_t$

Table 9.1 Akaike Information Criterion results for different AR (p) and MA (q) values for the ARIMA example given in the text

		q value			
		1	2	3	4
p value	1	67.90	56.03	56.67	54.83
	2	53.95	53.39	52.00	52.04
	3	**50.45**	52.34	52.13	53.19
	4	52.34	53.72	54.90	56.25

confidence interval at larger lags as well. This analysis indicates that ACF and PACF analysis is limited in terms of giving a complete answer to the exact order. In fact, the plots have limitations as it would be expected that 5% of autocorrelations to be outside of the confidence interval by random chance anyway. This means that the ACF and PACF must be interpreted with caution and in conjunction with the AIC.

Using the correlation analysis helps to locate the approximate area of the correct orders. In this example the ACF and PACF have suggested orders of around $q = 1$ and $p = 4$, and a test of the AIC for a variety of combinations of autoregressive and moving-average orders close to these values should be performed. Since the number of parameters for an ARIMA(p, 0, q) model is $p + q + 1$ (the one is due to the constant term) then the Akaike Information Citerion (AIC) has a particularly simple form

$$AIC = 2(p + q + 1) - 2\ln(L), \tag{9.12}$$

where L is the likelihood function of the ARIMA model. The AIC is checked for all combinations of orders with $p = 1, 2, 3, 4$ and $q = 1, 2, 3, 4$.[3] The result for each combination of p and q are shown in Table 9.1 which shows that a minimum AIC value of 50.45 is achieved for $p = 3$ and $q = 1$, correctly identifying the ARIMA(3, 0, 1) model.

It should be noted that any MA model can be estimated by an AR model with a sufficiently large number of lags (p value). Since the coefficients of an AR model can be calculated much more quickly than a full ARIMA model, it can be preferable to replace any ARIMA model with an AR (with differencing if not stationary) with large enough degree. This can also simplify the analysis and interpretation of the models. However, this may also require a relatively large order and thus many more parameters in the AR model compared to a simple ARMA model, reducing parsimony and interpretation.

There are a number of useful extensions to the ARIMA models. One of the most important for load forecasting purposes is to extend the model to include other explanatory variables. This model is then called an Autoregressive Integrated Moving Average with Explanatory Variable (ARIMAX) model. An ARIMAX (p, d, q) model includes extra external variables and is described by Eq. (9.13)

[3] We also test the larger ACF correlations at $q = 16, 17$ but these do not reduce the AIC score and are thus likely spurious.

$$\hat{L}_N^{(d)} = C + \sum_{i=0}^{h} \mu_i X_{N-i} + \sum_{i=1}^{p} \psi_i L_{N-i}^{(d)} + \sum_{i=1}^{q} \varphi_i \epsilon_{N-i} \tag{9.13}$$

with the differencing in (9.7) as before. The $\sum_{i=0}^{h} \mu_i X_{t-i}$ is the explanatory variables term. The model can be analysed by first considering an ARIMA model without exogenous inputs to isolate the orders of the equations, and then fitting the full model with the exogenous variables. Note, when finalising the AR and MA orders, the AIC should be applied to the full ARIMAX equation.

9.5 SARIMA and SARIMAX Models

An important extension to ARIMA models is to include seasonality. Seasonal ARIMA (SARIMA) includes an extra set of hyperparameters, denoted P, D, Q, which extends the model to include autoregressive, differencing and moving average terms at a specified seasonal level. These models are written ARIMA$(p, d, q)(P, D, Q)_S$ where the S indicates the seasonality. For example, for hourly data with daily seasonality, the SARIMA model would be written ARIMA$(p, d, q)(P, D, Q)_{24}$. The ACF and PACF are interpreted differently for seasonal ARIMA models. Consider a simple case where $d = D = q = Q = 0$ but $p = 2$ and $P = 1$. This means the time series would have autoregressive lags at $1, 2, 24, 25, 26$. Notice the combination of p and P terms means intra-seasonal lags $(1, 2)$ are applied onto the seasonal lag 24. An example of an ARIMA$(2, 0, 0)(1, 0, 0)_{10}$ is shown in Fig. 9.8 together with the partial autocorrelation function. Notice the significant spikes on the PACF at lag at the periodic intervals 10, 20, and also 11, 12, 21, and 22.

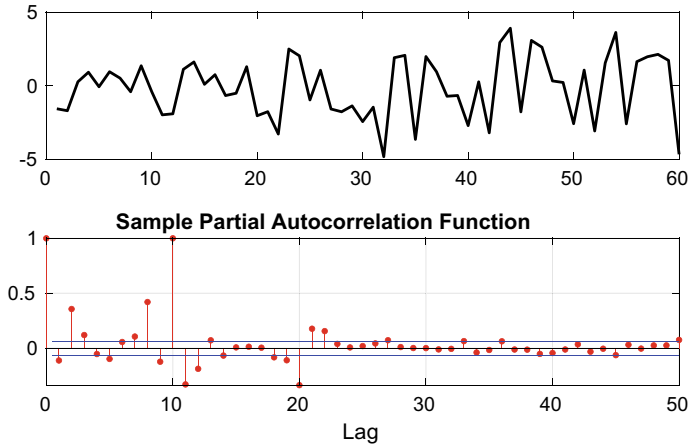


Fig. 9.8 Example of a ARIMA$(2, 0, 0)(1, 0, 0)_{10}$ series (top), and the corresponding PACF

The backshift operator is particular useful for representing SARIMA models. For example, an ARIMA$(p, d, q)(P, D, Q)_{24}$, for hourly seasonal data, can be represented as (Note no constant included here for clarity)

$$\left(1 - \sum_{i=1}^{p} \psi_i B^i\right)\left(1 - \sum_{j=1}^{P} \zeta_j B^{24j}\right)(1-B)^d(1-B^{24})^D L_N = \left(1 + \sum_{i=1}^{q} \varphi_i B^i\right)\left(1 + \sum_{j=1}^{Q} \theta j B^{24j}\right)\epsilon_N, \tag{9.14}$$

where ψ_i are the coefficients for the nonseasonal AR components, ζ_j are the coefficients for the seasonal AR components, φ_i are the coefficients for the nonseasonal MA components, and θj are the coefficients for the seasonal MA components. Note that $(1 - B^{24})$ represents a seasonal difference, i.e. $(1 - B^{24})L_N = L_N - L_{N-24}$. A seasonal difference of $D = 1$ is often sufficient.

For more details on ARIMA and SARIMA models check out [2] as well as other literature listed in Appendix D.

9.6 Generalised Additive Models

The linear models specified in Sect. 9.3 have various limitations. The two strongest and most common assumptions are that the errors follow a Gaussian distribution and that the model is a simple linear combination of various input variables.

Generalised linear models (GLM) are an extension to simple multiple linear models which include a link function which can allow for more diverse types of relationships. Using the notation as in Sect. 9.3 a dependent variable L_t at time t follows a GLM if for $n \geq 1$ input variables $X_{1,t}, X_{2,t}, \ldots, X_{n,t}$, then

$$g(\mathbb{E}(\hat{L}_{N+1})) = \sum_{k=1}^{n} \beta_k X_{k,N+1}, \tag{9.15}$$

for some (possibly nonlinear) *link function* $g(.)$, and such that the response variables are from a probability distribution from the exponential family (for example Gaussian, binomial or Gamma distributions—see Sect. 3.1). In other words, for a GLM, a transformation (via g) of the expected value of the dependent variable is a linear model. Notice, like the linear model all the linear coefficients, β_k's must be estimated but, in addition, the link function and a probability distribution model for the errors must also be chosen. When the link function is simply the identity ($g(x) = x$), and the dependent variables are assumed to be Gaussian, then Eq. (9.15) reverts to the simple multiple linear regression model as introduced in Sect. 9.3. The choice of link function and distribution you require depends on the problem being considered. For

example, if a dependent variable is non-negative then a log link function could be valid.

In this work general GLMs are not investigated. Instead the focus is on a very specific, and powerful, form of GLMs called **Generalised Additive Models** (GAMs) which have been very successful in load forecasting.[4] A GAM has the general form of

$$g(\mathbb{E}(\hat{L}_{N+1})) = \sum_{k=1}^{n} f_k(X_{k,N+1}), \tag{9.16}$$

for some (possibly nonlinear) smooth functions f_k.

GAMs have several advantages over GLMs, firstly the functions f_k allow the modelling of a much more diverse set of, possibly nonlinear, relationships whereas GLMs are only of the form $f_k(X_k, N+1) = \beta X_{k,N+1}$. In addition, these functions are often modelled nonparametrically, whereas the GLMs often assume parametric transforms and distributions (GAMs can also utilise common parametric forms as well, e.g. log functions, or polynomials for each f_k). Note that GAMs still use a link function g which can be used to transform the dependent variable into a more suitable form for training.

A nonparametric approach for each of the functions (f_k) in the additive model allows the algorithm to learn the relationship between each input variable $X_{k,N+1}$ from the observed data. A most common way to do this is to model each function using basis functions (See Sect. 6.2.5). Hence each function f_k is modelled

$$f_k(X_{k,N+1}) = \sum_{i=1}^{m} \alpha_{k,i}\phi_{k,i}(X_{k,N+1}), \tag{9.17}$$

for basis functions $\phi_{k,i}(X)$. Notice that this form transforms the GAM (9.16) into a GLM since the sum of the additive functions are now sums of linear functions in the bases.

For GAMs, it is common to choose **splines** for these basis functions. A spline is a piece-wise continuous function which is composed of other simpler polynomial functions. One of the simplest examples of a spline is a piecewise linear combination. Examples of a linear and a cubic spline is shown in Fig. 9.9. Note since a spline is continuous, the end of one polynomial must join on the start of the next polynomial. The *knots* specify where the polynomials join to each other. The cubic version is regressed on the observations (red points) between the knots to determine the other two coefficients in each cubic polynomial (two of the coefficients are already found by the interpolation constraints).

In more precise terms, consider the one dimensional case where the aim is to approximate a function $f : [a, b] \longrightarrow \mathbb{R}$ defined on an interval $[a, b] \subset \mathbb{R}$. For m knots at $a = z_1 < z_2 < \cdots < z_{m-1} < z_m = b$ a spline is fitted to some data by a

[4] For example, GAMs were part of the wining method for the Global Energy Forecasting Competition 2014 [3].

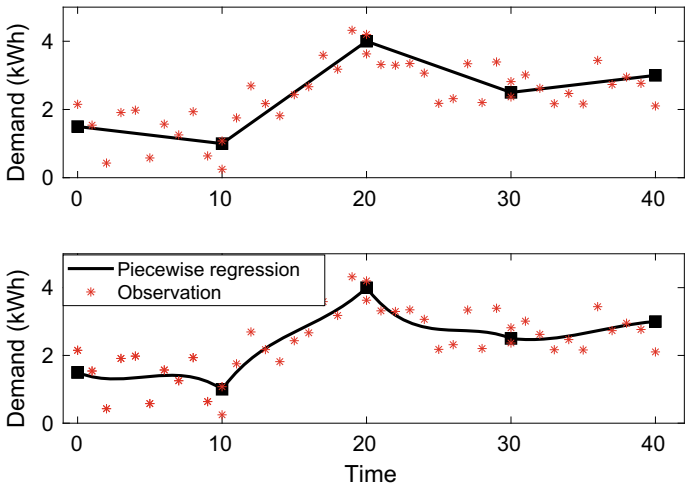


Fig. 9.9 Example of linear spline (top) and cubic spline (bottom). The squared markers are the knots which the polynomials interpolate

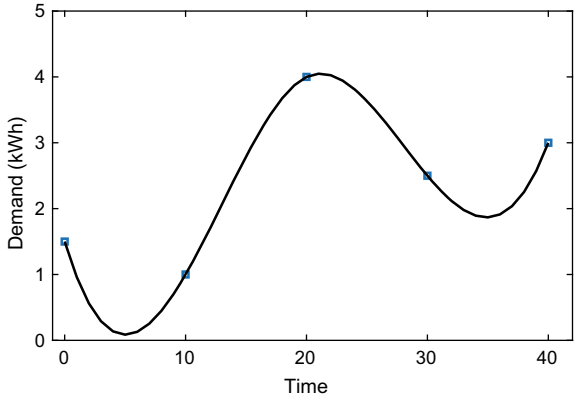


Fig. 9.10 Example of a smooth cubic spline interpolated through the same points as in Fig. 9.9

polynomial $s_i(z)$ on each subinterval $[z_i, z_{i+1}]$. Further, $s_i(z_{i+1}) = s_{i+1}(z_{i+1})$ since the spline should be continuous at the knots.

Other constraints can be applied to the spline to either make it easier to train or to satisfy other criteria. One of the most common requirements for a GAM is to ensure that the spline has a particular level of smoothness. As can be seen the cubic interpolation in Fig. 9.9 is smooth between the knots but not across the knots themselves. Constraining the cubic spline to be smooth whilst interpolating across the knots means all coefficients can be determined uniquely. Another way of saying the spline is smooth is to say that the derivative (up to a sufficient order) is continuous at the knot points. An example of a cubic spline which is smooth across the knots is shown in Fig. 9.10.

Note that the aim in forecasting is to regress on the data, and therefore it is not necessary (or desirable) to strictly interpolate through the observations. However, the principle is still the same and the final spline should be continuous throughout, including at the knots.[5] This is achieved by regressing the basis version of the relationship on the observations (See Eq. (9.17) above).

Certain basis functions, such as B-splines have very desirable properties such as providing smoothness at the knots. Further, although the number and type of the basis functions should be sufficiently flexible to fit the data, without any additional constraints or regularisation (Sect. 8.2.4) large numbers of knots and high polynomial degrees will increase the chance of overfitting to the noise. In addition, this will mean the polynomials will be very "wiggly". To prevent this, one approach is to include an extra term which is often added to penalise the lack of smoothness in the final solution. Recall this is much like the LASSO (Sect. 8.2.4) method and other regularisation techniques used to preventing overfitting.

A trivial example is where the link function is the identify. Since the GAM is linear in the basis functions then a least squares fit (see Sect. 8.2.4) to N observed dependent values $\mathbf{L} = (L_1, \ldots, L_N)^T$ can be considered. In other words, the aim is to minimise

$$\sum_{l=1}^{N} \left(L_l - \sum_{k=1}^{n} \sum_{i=1}^{m} \alpha_{k,i} \phi_{k,i}(X_{k,l}) \right)^2 \tag{9.18}$$

by training the parameters $\alpha_{k,i}$ for $k = 1, \ldots, n$ and $i = 1, \ldots, m$. For a large number of basis functions this model is likely to overfit the data. To prevent this a penalty can be applied, i.e.

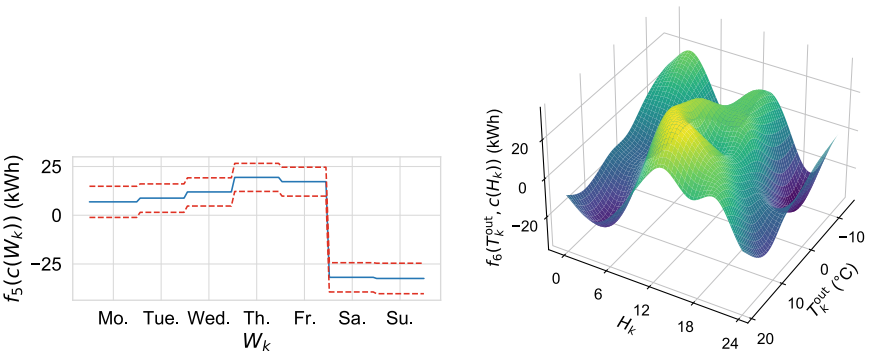
$$\left(\sum_{l=1}^{N} \left(L_l - \sum_{k=1}^{n} \sum_{i=1}^{m} \alpha_{k,i} \phi_{k,i}(X_{k,l}) \right)^2 \right) + K(f_1, \ldots, f_n). \tag{9.19}$$

The function $K(f_1, \ldots, f_n)$ is a penalty based on the individual functions f_k. In order to penalise deviation from smoothness the following penalty is commonly considered given by

$$K(f_1, \ldots, f_n) = \sum_{k=1}^{n} \lambda_k \int f_k''(x_k)^2 dx_k, \tag{9.20}$$

where the size of the penalty for each variable is controlled by the smoothing parameter, λ_k. The minimisation of the sum of second derivative of each function reduces the wiggliness of the function, i.e. encourages more smoothness depending on the value of the lambda's. These smoothing parameters are, as usual, often found by cross-validation (see Sect. 8.1.3) or by optimising information criterion (Sect. 8.2.2).

[5] The knots here do not have a specific y-value since they have to be determined by the regression. However, the x-values of the knots will usually be fixed at particular uniformly-spaced time steps.



(a) Weekday Term (b) Hour and Temp Interaction.

Fig. 9.11 Example of partial contributions for single weekday term (Left) and interaction between hour-of-day and outside temperature. Reprinted from [4] under CC 4.0

Due to the basis function representation in (9.17) it can be shown that the penalty takes a particularly convenient quadratic form

$$\int f_k''(x_k)^2 dx_k = \boldsymbol{\alpha}_k^T \mathbf{S}_k \boldsymbol{\alpha}_k, \tag{9.21}$$

where $\boldsymbol{\alpha}_k = (\alpha_{k,1}, \ldots, \alpha_{k,m})^T$ and $\mathbf{S}_k \in \mathbb{R}^{m \times m}$ is a matrix formed from derivatives of the basis functions evaluated at the input values for $X_{k,l}$.

As in multiple linear regression models, GLMs and GAMs can be used to model the interaction of two or more features, for example

$$g(\mathbb{E}(\hat{L}_{N+1})) = f_1(X_{1,N+1}) + f_2(X_{2,N+1}) + f_3(X_{1,N+1}, X_{3,N+1}) \tag{9.22}$$

In this case the first two functions $f_1(X_{1,N+1})$, $f_2(X_{2,N+1})$ model a single variable each, but the third function models the effect of the interaction of $X_{1,N+1}$, $X_{3,N+1}$. In these cases multi-dimensional versions of spline functions can be used.

The additive nature of the GAMs model makes the model interpretable since the contributions of individual features and interactions can be analysed and visualised, even if complex nonlinear functions are used. Figure 9.11a and b show exemplary visualisations of the contribution of individual terms to the final prediction. Figure 9.11a shows the weekday (W_k) contribution to the demand, indicating that for the specific model, the load is much lower on weekends and is highest on Thursdays. Figure 9.11b shows the combined effect of the interaction of the hour of the day (H_k) and outside temperature (T_k^{out}), for example, influence is lowest over night and for cold temperatures and highest around noon for high temperatures. Plots of the smaller subsets (typically one or two) of the full input variables are called **partial dependence plots** and allow us to examine, and better interpret, the overall effects of the different components.

There are a whole host of different approaches and parameters to choose and many GAM programming packages, such as `gam` or `mgcv` in R and `pygam` in python,[6] work for a selection of splines, smoothing parameter selection methods, and link functions. Often these packages will have their own default settings but in many cases these can be tweaked to ensure a more accurate fit and better performance. In particular if it is known that the errors are not Gaussian, or that a particular independent variable only has a linear relationship to the dependent variable, then these can be specified when implemented. Other parameters or data assumptions should also be checked, but if you are uncertain then several values can be checked via cross-validation methods. Since regularisation is employed within most packages it is better to have more degrees of freedom specified by the splines than too few. As usual residual checks (Sect. 7.5) can be used to evaluate the final models and identify incorrect assumptions or areas of improvement.

Note that there may be additional constraints applied to the basis/spline functions to better model the features in the demand data. In particular, since there is often periodicity in many of the dependent variables (e.g. hour of the day or week), basis functions can be chosen to include these features, e.g. periodic B-splines which are available for some of the aforementioned packages.

The above is a basic introduction to GAMs and a more detailed description for a very complicated area is beyond the scope of this book. Some further reading is included in Appendix D.2.

9.7 Questions

For the questions which require using real demand data, try using some of the data as listed in Appendix D.4. Preferably choose data with at least a year of hourly or half hourly data. In all the cases using this data, split it into training, validation and testing with a $3:1:1$ ratio (Sect. 8.1.3).

1. Select a demand time series. Analyse the seasonalities, (see Sect. 6.2). Generate some simple benchmark forecasts for the test set, including the persistence forecast, and seasonal persistence forecasts, one for each seasonality you found. Calculate the RMSE errors. Which one is lower? How does this compare with the seasonalities you observed? Compare these results to the ACF and PACF plots for the time series.
2. Continuing the experiment from the previous section generate seasonal moving averages using the identified seasonalities. Using a validation set (Sect. 8.1.3) identify the optimal value of seasonal terms, p, to include in the average. If there is multiple seasonalities which one has the smallest errors overall? How does the RMSE error on a test set for the optimal seasonal average forecasts compare to the persistence forecasts in the previous question?

[6] See for example https://cran.r-project.org/web/packages/mgcv/mgcv.pdf and https://pygam.readthedocs.io/en/latest/.

3. Generate a simple 1-step ahead exponential smoothing forecasts (Sect. 9.2) for a load forecast time series (preferably one which has double seasonal patterns, usually daily and weekly). Manual select different values of the smoothing parameter, α. Plot the RMSE errors against the smoothing parameters. Do a grid search to find the optimal smoothing parameter (Sect. 8.2.3). How does the optimal forecast compare to a simple persistence forecast? Now consider the Holt-Winters-Taylor forecast and perform a grid search for the four parameters ϕ, λ, δ, ω.
4. Investigate a LASSO fit for a linear model. Set the coefficients of a model with a few Sine terms, e.g. $\sum_{k=0}^{N} \alpha_k \sin kx$, for N about 5, and $x \in [0, 4\pi]$. Sample 20 points from this data (and add a small amount of Gaussian noise). Now fit a multiple linear equation of the form $\sum_{k=0}^{50} \gamma_k \sin kx$ using least squares regression to find the coefficients γ. Now plot the trained model on 20 new $x \in [0, 4\pi]$ values. Is it a good fit? Now try and minimise the LASSO function using different values of the regularisation parameter λ (See Sect. 8.2.4). How does the fit change as you change the parameter? How many coefficients γ are zero (or very small). Use inbuilt functions to do the LASSO fit such as sklearn[7] in Python, or glmnet[8] in R.
5. Show for the basis representation for GAMs, that the second order penalty term (9.20) takes the form $\boldsymbol{\alpha}_k^T \mathbf{S}_k \boldsymbol{\alpha}_k$
6. Try and generate a linear model that fits a demand profile. Consider what features to use, if time of day is important considering using dummy variables. If weather data is available check if there is a relationship with the demand (see Chap. 14). In the case study in Sect. 14.2 a linear model will be generated for modelling the low voltage demand. Come back to this question once you've reached that part of the book and see what similarities there are. What have you done differently? What would you like to change in the model? Fit the linear model using standard packages in Python and R such as sklearn[9] and lm[10] respectively. Now using the same features implement a GAM. Again these forecasts can be trained using standard Python and R packages such as pygam[11] and mgcv[12] respectively. These packages often have similar syntax to the linear models. Now compare the forecasts and the errors. For the GAM look at the partial dependency plots. What is the relationship for each variable chosen.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html.

[8] https://cran.r-project.org/web/packages/glmnet/index.html.

[9] See https://scikit-learn.org/stable/modules/linear_model.html.

[10] See https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/lm.

[11] See https://pygam.readthedocs.io/en/latest/.

[12] See https://cran.r-project.org/web/packages/mgcv/mgcv.pdf.

References

1. D. Ruppert, S.M. David, *Statistics and Data Analysis for Financial Engineering: With R Examples*. Springer Texts in Statistics (2015)
2. R.J. Hyndman, G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd edn. (OTexts, Melbourne, Australia, 2018). https://Otexts.com/fpp2. Accessed on July 2020
3. P. Gaillard, Y. Goude, R. Nedellec, Additive models and robust aggregation for gefcom2014 probabilistic electric load and electricity price forecasting. Int. J. Forecast. **32**(3), 1038–1050 (2016)
4. M. Voss, J.F. Heinekamp, S. Krutzsch, F. Sick, S. Albayrak, K. Strunz, Generalized additive modeling of building inertia thermal energy storage for integration into smart grid control. IEEE Access **9**, 71699–71711 (2021)

Chapter 10
Machine Learning Point Forecasts Methods

The traditional statistical and benchmark methods presented in Sect. 9.1 often assume some relatively simple relationship between the dependent and independent variables, be that linear trends, particular seasonalities or autoregressive behaviours. They have performed quite successfully for load forecasting, being quite accurate, even with low amounts of data, and can easily be interpreted by practitioners. However, the methods described in Sect. 9.1 may be less suitable for modelling more complex and highly nonlinear relationships. As data has become more ubiquitous due to increased monitoring, **machine learning** methods are becoming increasingly common as they can find complicated and subtle patterns in the data.

Recall from Eqs. 5.27 and 5.28 that defined the functional forms of the 1-step and m-step ahead forecasting problem. It describes the relationship of the load for m steps ahead, L_{n+m}, for forecast origin at time step n, with autoregressive features $L_1, \ldots, L_n$, explanatory features $Z_1, \ldots, Z_k$ and function f. As explained in Sect. 4.2, this function f can be learned from training data, i.e., the load forecasting task can be modelled as a supervised learning task, where a machine learning model is trained to learn the possibly complex relationship of the load with some features. As the load forecasting task is typically expressed as a numeric value, it is in most cases a **regression** problem.

The following sections introduce a few popular machine learning methods that can be used for time series forecasting. Section 10.1 introduces **k-nearest neighbour regression** (k-NN), a relatively simple model that can, together with multiple linear regression, function as a good benchmark model for datasets which are not too large. **Support vector regression** (Sect. 10.2) has been a popular model in the early 2000s as it can provide accurate forecasts with nonlinear relationships, but only on data sets which are relatively small. Tree-based ensemble models like **random forest regression** and **gradient-boosted regression trees** (Sect. 10.3) are powerful, robust models that often perform very good on structured data and are therefore strong contenders for many practical time series problems, even with complex relationships of independent variables with many variables. They also scale well to many observations.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_10

However, as in many other domains, artificial neural networks have become increasingly popular including for time series tasks and, in particular, load forecasting. While regular feed-forward neural networks are relatively capable, recurrent neural networks and their more sophisticated deep variants like the **long-short term memory** (LSTM) and **gated recurrent unit** (GRU) have also been successful for time series tasks since they are able to model the autoregressive relationships (Sect. 10.5). More recently, **convolutional neural networks** (CNN) have also provided state-of-the-art results and been used in favour of recurrent architectures as they can be trained more efficiently. This creates interesting architectures, especially for large time series data sets when training on smart meter data for many consumers and distribution level networks.

This book will only briefly give an overview of more recent developments like transformer networks and specifically designed neural network architectures that have shown promising results. However, as those are most relevant in research, we omit the details and refer the interested reader to some of the core literature.

We note that the machine learning models mentioned above can be used for regression and classification tasks. However, most typically, load forecasting is a regression task, and therefore their functionality is explained within the regression context, which may differ from other explanations. For instance, in k-nearest neighbours or random forests, their predictions are averaged for the regression case, but they may use **majority voting** for the classification case. In artificial neural networks, the difference is the usage of different **loss functions** (i.e., mean square error in regression vs cross entropy loss in classifications) and the **activation function** of the final layers (i.e., linear activation in regression vs softmax function as activation in classification).

10.1 k-Nearest Neighbour Regression

k-nearest neighbour regression (k-NN) and multiple linear regression (see Sect. 9.3) are often considered the two most simple supervised learning methods. **Linear regression** can be considered a **high bias** model as it places strong assumptions on the linear relationship of the variables and the distributions of residuals. In contrast, **k-NN** makes no parametric assumptions and is therefore considered a **low bias** model. However, it is worth noting that the level of bias depends on the data, the choice of parameters (and hyperparameters), and how well a model captures the underlying relationships.

The basic algorithm makes a prediction by finding the k instances in the training data set that are most similar to the instance for which the prediction is made. So consider a training data set $\mathcal{X} = \left\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_j, \mathbf{y}_j), (\mathbf{x}_N, \mathbf{y}_N)\right\}$ and a new instance $\mathbf{x}'$ for which a prediction should be made. Then k-NN looks of the k instances $(\mathbf{x}''_1, \mathbf{y}''_1), (\mathbf{x}''_2, \mathbf{y}''_2), (\mathbf{x}''_i, \mathbf{y}''_i), \ldots, (\mathbf{x}''_k, \mathbf{y}''_k) \in \mathcal{X}$, where the $\mathbf{x}''_i$ are most similar to $\mathbf{x}'$ according to some distance measure. In a regression problem, the prediction $\hat{\mathbf{x}}$ is the average of $\mathbf{y}''_1, \mathbf{y}''_2, \mathbf{y}''_i, \ldots, \mathbf{y}''_k)$ and the majority vote in classification. The

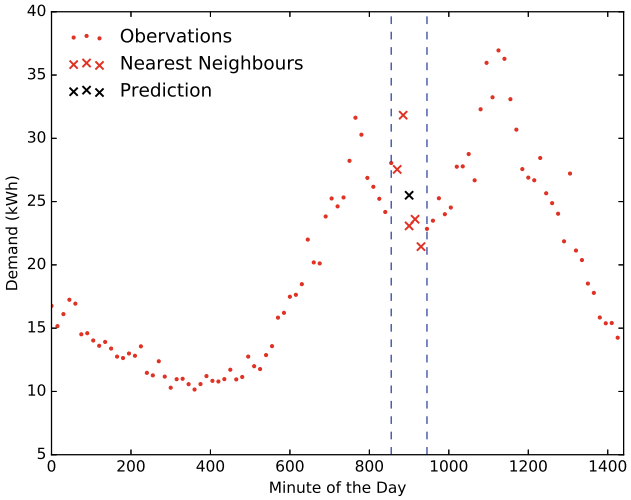


Fig. 10.1 A simple illustration of k-Nearest Neighbours. The k closest points defined by the red crosses are averaged to produce the prediction (black cross)

fact that the choice of distance measure and the aggregation function can be chosen quite flexibly and depending on the circumstances and the application, makes k-NN an extremely versatile method.

In a naïve implementation, a prediction is made by calculating the distance between the test data and all the training points. Then it selects the k number of training points closest to the test data according to the similarity measure. However, in practice, efficient data structures like ball trees make it unnecessary to compare the test data to all training data. Note, that some of these techniques require certain properties of the distance measure, like some of the metric properties (see discussion below). A simple example of a k-NN for an energy profile is shown in Fig. 10.1. The similarity is how close the points are within the day with the k points used to estimate one period (black cross) bounded within the vertical blue lines.

The parameter k is the most important hyperparameter to tune and it controls the under- and over-fitting of the model. Choosing k too small may cause overfitting since the prediction is made based on only a few data points. A k is too large, the estimate is based on too many observations and, therefore, may underfit. An illustration of the effect of the hyperparameter is shown in Fig. 10.2 for different values of k. The larger the k, the smoother the fit but also, the higher the bias, and also notice the peaks are less well approximated.

As the algorithm relies on a distance metric, it is important to normalise the data, as otherwise, the results may depend on the scale of the features (e.g. cause different predictions if the temperature is in °C or °F or load in W or kW). Therefore, the choice of the normalising procedure is, especially for k-NN, an important design choice. For a discussion on normalisation techniques, see Sect. 6.1.3).

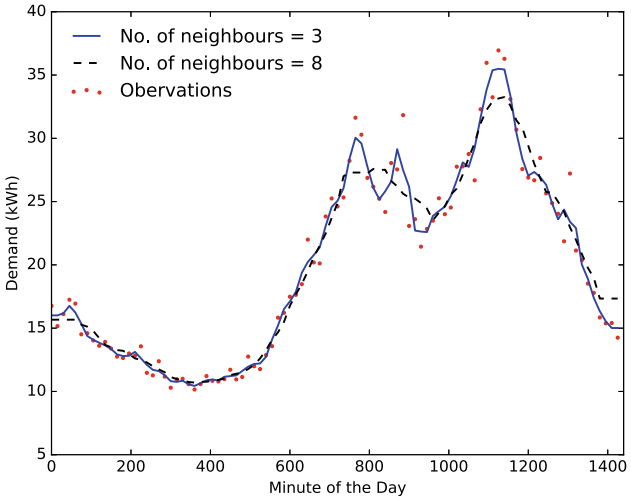


Fig. 10.2 Effect of the value of k on the k-nearest neighbour estimate

As discussed above, there are two main steps of k-nearest neighbour regression: determining the most similar instances and combining the corresponding targets. Both steps can be seen as design decisions of the algorithm and varied for specific applications. For the first step, it is often useful to explore the usage of different distance measures. As a default, k-NN uses the Euclidean distance and combines the selected targets using the arithmetic mean. The Euclidean distance is defined as the squared difference of the elements, i.e., the 2-norm introduced in Sect. 7.1. The arithmetic mean is a natural choice for the Euclidean distance since, for a finite sample, it minimizes the sum of squared distances (Sect. 8.2.1). However, in certain applications, the choice of the medoid, a representative from the sample which has a minimal squared distance to all the other points, can be a reasonable choice.

The Euclidean distance is a lockstep or "point-wise" distance measure since it measures the distance between individual elements of the input sequences before aggregating them. In time series, this means that the evaluation is performed by matching values at the same time step. In contrast, the group of so-called **elastic distance** measures works by first optimally aligning the time series in the temporal domain so that the overall cost in terms of a cost function of this alignment is minimal. This property can be useful when working with load profiles in the low-voltage grid that exhibit high volatility, to avoid the double-penalty effect (see Sect. 13.3 for a special elastic distance measure).

k-NN allows the use of arbitrary distance measures for finding the most similar neighbours. However, if the distance is not a metric, this search may need to resort to brute force and hence may not scale well with larger datasets. In order for a distance measure to be a metric, it must have the following properties:

- Non-negativity: $\mathrm{D}(\mathbf{X}, \mathbf{Y}) \geq 0$
- Identity of indiscernibles: $\mathrm{D}(\mathbf{X}, \mathbf{X}) = 0$
- Symmetry: $\mathrm{D}(\mathbf{X}, \mathbf{Y}) = \mathrm{D}(\mathbf{Y}, \mathbf{X})$
- Triangle inequality: $\mathrm{D}(\mathbf{X}, \mathbf{Y}) \leq \mathrm{D}(\mathbf{X}, \mathbf{Z}) + \mathrm{D}(\mathbf{Z}, \mathbf{Y})$

Recall that the p-norms in Sect. 7.1 are all metrics.

Many algorithmic improvements to speed up similarity search rely on metric properties, most importantly the triangle inequality. Computing a sample mean for an arbitrary metric is often intractable. Hence, approaches resort to using approximate solutions or use the medoid instead of the sample mean. For large datasets, subsets of the training set may be used to reduce computational costs.

The most popular elastic distance measure is **dynamic time warping** (DTW). It was first introduced for the application of speech recognition and has been shown to perform well on many datasets.[1] It is considered an elastic measure, as it finds an optimal alignment between two time series by stretching or "warping" them, minimizing the Euclidean norm between the aligned points. Figure 10.4 shows such an optimal alignment of two time series $\mathbf{X}$ and $\mathbf{Y}$. It maps the first peak of the top profile to the peak of the same height in the bottom profile. Then the second smaller peak is aligned with the peak of the same height occurring later. In contrast, Fig. 10.3 shows the "point-wise" Euclidean distance.

DTW can be recursively defined by:

$$\mathrm{DTW}(X_{:i}, X_{:j}) = \mathrm{D}(X_i, Y_j) + \min \left\{ \begin{array}{l} \mathrm{DTW}\left(X_{:i-1}, Y_{:j-1}\right), \\ \mathrm{DTW}\left(X_{:i}, Y_{:j-1}\right), \\ \mathrm{DTW}\left(X_{:i-1}, Y_{:j}\right) \end{array} \right\} \tag{10.1}$$

Then the DTW distance between time series $\mathbf{X}$ and $\mathbf{Y}$ is

$$\mathrm{DTW}(\mathbf{X}, \mathbf{Y}) = \mathrm{DTW}(X_{:T}, Y_{:T}). \tag{10.2}$$

As mentioned above, D can be any distance function but is generally the Euclidean distance. A naive recursive implementation would lead to exponential run time. The most popular deterministic approach is an algorithm based on dynamic programming that leads to quadratic run time, i.e., scales quadratically with the length of the input. The optimal solution of the DTW algorithm can be represented by the warping path, the path along the cost matrix that contains the cost for each individual aligned points X_i and Y_j (i.e. the i^{th} row and j^{th} column of the matrix is the distance between the elements X_i and $Y_{j:}$). Figure 10.6 (left) shows the cost matrix of aligning each element of the above vectors $\mathbf{X}$ and $\mathbf{Y}$ shown in Figs. 10.3, 10.4 and 10.5. The black line shows the warping path.

[1] DTW is useful for speech applications since vocal patterns may stretch or shrink depending on whether someone speaks slower or faster. DTW allows different time series to be compared despite these scalings, which means the focus is on comparing the arrangement and ordering of features rather than when they occur or if they are stretched (or shrunk).

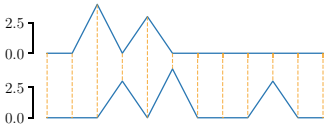


Fig. 10.3 Euclidean distance, no alignment, ED($\mathbf{X}$, $\mathbf{Y}$) = 7.68

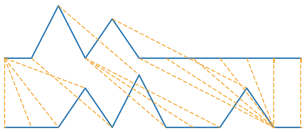


Fig. 10.4 Optimal alignment for DTW distance, DTW($\mathbf{X}$, $\mathbf{Y}$) = 3.00

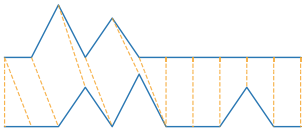


Fig. 10.5 Optimal alignment for cDTW distance with $c = 3$, cDTW($\mathbf{X}$, $\mathbf{Y}$; 3) = 3.32

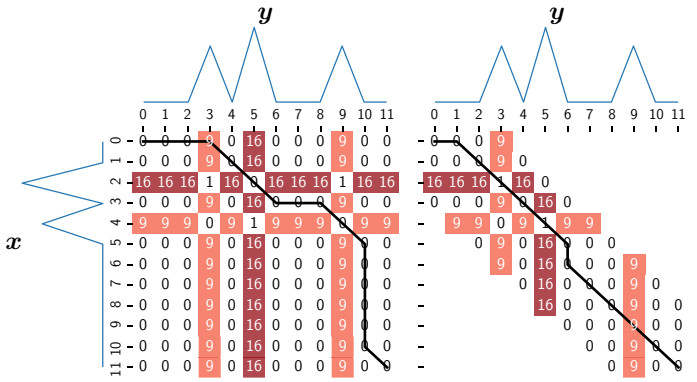


Fig. 10.6 Cost matrix for aligning $\mathbf{X}$ and $\mathbf{Y}$ with the optimal warping path (left), and the cost matrix constrained by the Sakoe-Chuba Band with its warping path (right)

As DTW is popular, many adjustments have been proposed. The most common adaption is the introduction of a constraint that limits the values in the cost matrix to be within some radius r, which is often referred to as the Sakoe-Chiba Band. This version is referred to as constrained DTW, or cDTW. Figure 10.6 (right) shows the constrained cost matrix with $r = 3$ with the resulting warping path. Figure 10.5 shows the associated optimal constrained DTW-alignment of the constrained DTW. In this case, the early peaks are aligned instead of the smaller peak being aligned to the later one of the same height. This results in a slightly larger distance (cDTW($\mathbf{X}$, $\mathbf{Y}$) = 3.32

versus DTW($\mathbf{X}$, $\mathbf{Y}$) = 3.0.) in the example given. Due to its popularity and consistent effectiveness, DTW is often a default choice of benchmark for many problems. In particular, DTW may be one choice for household level forecasts as will be illustrated in more detail in Sect. 13.3.

10.2 Support Vector Regression

Support Vector Regression (SVR) is a popular machine learning method used for time-series prediction. To begin, consider a time series $L_1, L_2, \ldots$ and n explanatory time series variables $X_{1,t}, X_{2,t}, \ldots, X_{n,t}$ at each time step t which are related to the load, L_t, via a simple multiple linear model, as introduced in Sect. 9.3, described by

$$\hat{L}_{N+1} = \sum_{k=1}^{n} \beta_k X_{k,N+1} + b, \tag{10.3}$$

for some constant b. For simplifying the notation write this in the matrix-vector form

$$\hat{L}_{N+1} = \boldsymbol{\beta}^T \mathbf{X}_{N+1} + b, \tag{10.4}$$

where $\mathbf{X}_t = (X_{1,t}, X_{2,t}, \ldots, X_{n,t})^T$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)^T$ are the vectors of explanatory variables and regression parameters respectively.

A standard way to find the relevant parameters, $\boldsymbol{\beta}$, is to minimise the least squares difference, between the model and the observations i.e.

$$\hat{\boldsymbol{\beta}} = \arg\min_{\beta \in \mathcal{B}} \sum_{t=1}^{N} (L_t - \boldsymbol{\beta}^T \mathbf{X}_t - b)^2. \tag{10.5}$$

This can be extended to a regularised form such as LASSO or ridge regression as shown in Sect. 8.2.4 where an additional terms $||\boldsymbol{\beta}||_p$ is added to minimise the overall size of the coefficients.

In contrast, support vector regression (SVR) fits the linear models for all observations whose errors are within a certain threshold, $\epsilon \geq 0$. This can be expressed as

$$-\epsilon \leq L_t - \sum_{k=1}^{n} \beta_k X_{k,t} - b \leq \epsilon. \tag{10.6}$$

The aim of SVR is to minimise the parameter size

$$\frac{1}{2} ||\boldsymbol{\beta}||_2^2, \tag{10.7}$$

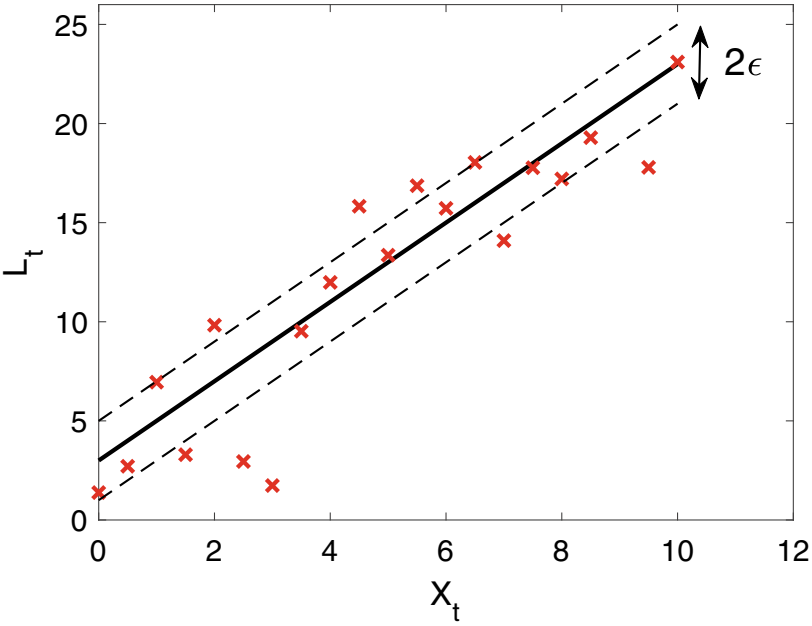


Fig. 10.7 Example of ϵ threshold region for model (black line), threshold bounds (dashed line) and observations (red crosses)

subject to the constraint (10.6). So in SVR the different sizes of the errors don't matter as long as they are within a certain threshold. The optimisation in Eq. (10.7) maximises the 'flatness' or complexity of the model and is comparable to the approach of ridge regularisation for linear least squares models (see Sect. 8.2.4).

To illustrate support vector regression, consider a simple 1-dimensional example for a particular ϵ-precision value as shown in Fig. 10.7. In some cases it may be that there are no points which can be approximated with ϵ precision, or an allowance for larger errors may be desired, in this case slack variables, ξ can be added to make the problem feasible and accept larger errors. In this updated form the aim is to minimise the cost function:

$$\frac{1}{2}||\boldsymbol{\beta}||_2^2 + C\sum_{t=1}^{N}(\xi_t + \xi_t^*), \tag{10.8}$$

with respect to $\boldsymbol{\beta}$,

$$\text{Subject to} \begin{cases} L_t - \sum_{k=1}^{n} \beta_k X_{k,t} - b \leq \epsilon + \xi_t, \\ \sum_{k=1}^{n} \beta_k X_{k,t} + b - L_t \leq \epsilon + +\xi_t^* \\ \xi_t, \xi_t^* \geq 0. \end{cases} \tag{10.9}$$

The constant $C > 0$ is a trade-off between maximising the flatness and minimising the allowable deviation beyond ϵ. Both C and ϵ must be found to implement SVR for a linear model. The optimal parameters can be found via cross-validation by testing a variety of values over the validation set (Sect. 8.1.3).

Often the linear SVR problem is solved more easily in its *dual form*, in which case the forecast model can be shown to be of the form

$$\hat{L}_{N+1} = f(\mathbf{X}) = \sum_{t=1}^{N} \alpha_t < \mathbf{X}_t, \mathbf{X} > + b, \tag{10.10}$$

where $<, >$ represents an inner-product function (for example dot product) and α_t are coefficients derived from the Lagrange multipliers of the optimisation (see [1] for more details).

An advantage of SVR is that it can also be extended to nonlinear regressions. This is achieved by mapping the input features to a higher dimensional space using a transformation function, Φ. In the nonlinear case, the following transformed multiple linear equation is considered

$$\hat{L}_{N+1} = \boldsymbol{\beta}^T \Phi(\mathbf{X}_{N+1}) + b. \tag{10.11}$$

To solve this problem in practice only requires knowing the kernel function $K(\mathbf{X}_i, \mathbf{X}_j) =< \Phi(\mathbf{X}_i), \Phi(\mathbf{X}_j) >$ where $<, >$ represents an inner product as before (see references in Appendix D for more details). As with the linear form the final forecast model can be written in the dual form

$$\hat{L}_{N+1} = f(\mathbf{X}) = \sum_{t=1}^{N} \alpha_t K(\mathbf{X}_t, \mathbf{X}) + b. \tag{10.12}$$

There are several kernels that can be chosen. Some of the most popular are the Gaussian Radial Basis Function (RBF) given by

$$K(\mathbf{X}_i, \mathbf{X}_j) = \exp\left(-\gamma ||\mathbf{X}_i - \mathbf{X}_j||^2\right), \tag{10.13}$$

and the polynomial given by

$$K(\mathbf{X}_i, \mathbf{X}_j) = (1 + < \mathbf{X}_i, \mathbf{X}_j >)^p, \tag{10.14}$$

where p is the order of the polynomial. The larger the order the more flexibility in the regression fit. As usual, to choose the best model and parameters is achieved by comparison on the validation set. The power of the kernel method is that a nonlinear problem has essentially been transformed to a linear problem, simply by transforming the original variables.

10.3 Tree-Based Regression Methods

10.3.1 Decision Tree Regression

Used on their own, decision trees are not particularly accurate and have limited usefulness for forecasting. However, when multiple decision trees are taken together they produce some of the most powerful and accurate machine learning models. This includes random forest (see Sect. 10.3.2), bagging methods and gradient boosted decision trees (see Sect. 10.3.3). Regression trees can be used to either classify discrete/categorical data, or to regress on continuous data. The latter will be of most interest for load forecasting and are discussed in this and the next couple of sections.

As in Sect. 10.4, the aim is to learn a function $f : \mathbf{X} \longrightarrow \mathbb{R}$ based on M inputs $(X_{1,t}, X_{2,t}, \ldots, X_{M,t})^T \in \mathbf{X}$ at time t which predicts the load L_{N+1} at time $t = N + 1$,

$$\hat{L}_{N+1\,N} = f(\{X_{1,t}, \ldots, X_{M,t}\}, \boldsymbol{\beta}), \tag{10.15}$$

where $\boldsymbol{\beta}$ is the parameters necessary for defining the decision tree. As in previous sections, the inputs $X_{1,t}, X_{2,t}, \ldots, X_{M,t}$ are quite general and can include, for example, historical loads L_t for $t \leq N$, or other explanatory variables such as temperature forecasts.

A decision tree defines a function by splitting the training observations in the domain, $\mathbf{X}$, into disjoint subsets which are distinct and non-overlapping. The function is simply the average values of the historical observations of the dependent variable (in this case the load L_t) within each disjoint subset. A disjoint partition of a 2D variable space into four disjoint sets is illustrated in Fig. 10.8 which presents the types of splits that decision trees can produce. The algorithm starts with the full domain, in this case the square area $[-1, 1] \times [0, 2]$, shown in the plot with the black boundary. A split of one of the variables is made which optimises the split of the domain according to some criteria (This will be investigated in detail later, but for example, for regression this could be the split which maximally reduces the mean squared error (RMSE) (see Chap. 7) between the observations and the model). In this illustrative example the best split is to cut when $x = 0$ (represented by the red line). In the next iteration, the process is repeated and tries to find the next best split on the two sections just produced in the last iteration. This turns out to be the horizontal line $y = 0.5$ shown by the yellow line. In the next iteration a final cut at $x = 0.6$ is then chosen which is given by the purple line. The splitting can be written as a tree with each split of the tree representing another partition in the domain. The corresponding decision tree representing the domain partitions in Fig. 10.8 is shown in Fig. 10.9. The final nodes, labelled with $C1, \ldots, C4$ are the end, or **leaf**, nodes and represent the final partitions of the domain.

Now consider a series of pairs $(L_{t+1}, \mathbf{X}_t)$ $(t = 1, \ldots, N)$ of dependent and independent variables with $\mathbf{X}_t = (X_{1,t}, X_{2,t}, \ldots, X_{M,t})$ a set of M attributes at time t, and L_{t+1} an observation at the next timestep, $t + 1$. Note the assumption here is that the observation are continuous real-valued variables (discrete/categorical dependent

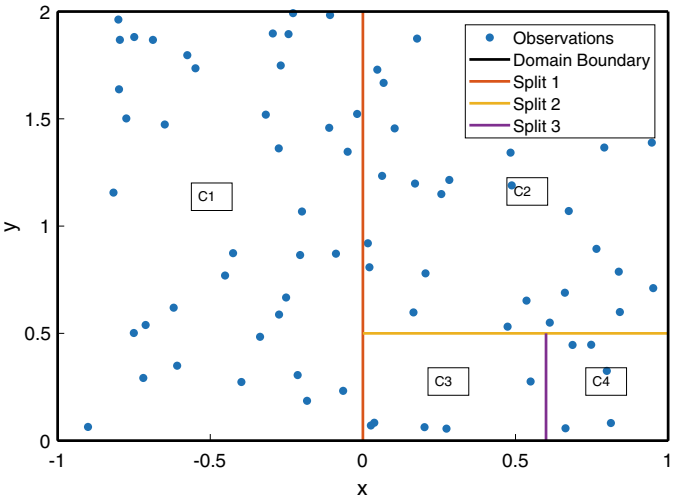
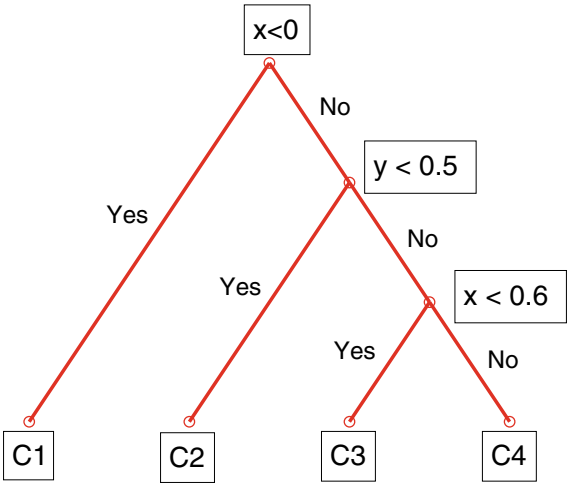


Fig. 10.8 An illustration of how a decision tree may split a 2 variable domain into a disjoint 'optimal' partition

Fig. 10.9 The decision tree which produces the partition in Fig. 10.8. Each split in the tree represents a split in the domain



variables are not usually considered in load forecasting applications but also can be included, e.g. see the dummy variables in Sect. 6.2.6). Given a partition of the domain of $\mathbf{X}$ into P disjoint sets, denoted by $C_1, C_2, \ldots, C_P$ each of which contain $N_1, N_2, \ldots, N_P$ points respectively, define a piecewise function $f_P()$ (where the P is to indicate the dependence on the partition P) which is constant on each disjoint set and can be written as

$$f_P(\mathbf{X}) = \sum_{p=1}^{P} \alpha_p \chi_p(\mathbf{X}), \tag{10.16}$$

where $\chi_p()$ is a characteristic (or indicator) function defined by

$$\chi_p(\mathbf{X}) = \begin{cases} 1, & \text{if } \mathbf{X} \text{ is in set } C_p \\ 0, & \text{otherwise} \end{cases}.$$

and

$$\alpha_p = \frac{1}{N_p} \sum_{t=1}^{N} L_t \chi_p(\mathbf{X}_t), \tag{10.17}$$

is the average of the dependent variables where the corresponding independent variables are within the partition set C_p. Hence for each partition defined by the decision tree a corresponding piecewise function can be defined. This is known as a **decision tree regression**. The cost function used to define the split in a decision tree regression is often the mean square errors between the estimate, defined by Eq. (10.16), and the observations and is given by

$$MSE = \frac{1}{N} \sum_{t=1}^{N} (L_t - f_P(\mathbf{X}_t))^2. \tag{10.18}$$

Of course the decision tree can continue splitting into smaller partitions and reduce the MSE until each set only contains a single observation. However, this would likely lead to overfitting (Sect. 8.1.2) of the regression tree and hence poor forecast estimates. Instead the fit can be controlled by calibrating a number of parameters of the decision tree or defining a stopping criteria. There are several different parameters or combinations of parameters which could be chosen in order to optimise the generalisability of the regression tree, some of the most common are

- Fixing a minimum number of observations $\min_{p \in 1,\ldots,P} N_p$ in each leaf node.
- Stopping when the MSE decreases less than some threshold, τ, when an additional split is added.
- Fixing a maximum number of branch nodes (i.e. maximum value of partitions P).
- Maximum depth of the tree (i.e. maximum number of splits).

The value for these parameters are typically chosen using cross validation (see Sect. 8.1.3) where a variety of different models are trained with different parameters on the training set and the best models are chosen based on how they perform on a validation set.

To illustrate the process for generating a regression tree, consider a simple $1D$ case as shown in Fig. 10.10. Observations are generated by sampling 40 points from the curve and adding a small amount of noise. Two different regression trees are generated using different choices for the minimum number of observations, $\min_{p \in 1,\ldots,P} N_p$, in each leaf node, in this case 10 and 2. These are trained to the noisy observations to produce two functions given by Eq. (10.16). The graphs for the final functions for these two regression trees are shown in Fig. 10.11 together with the original curve.

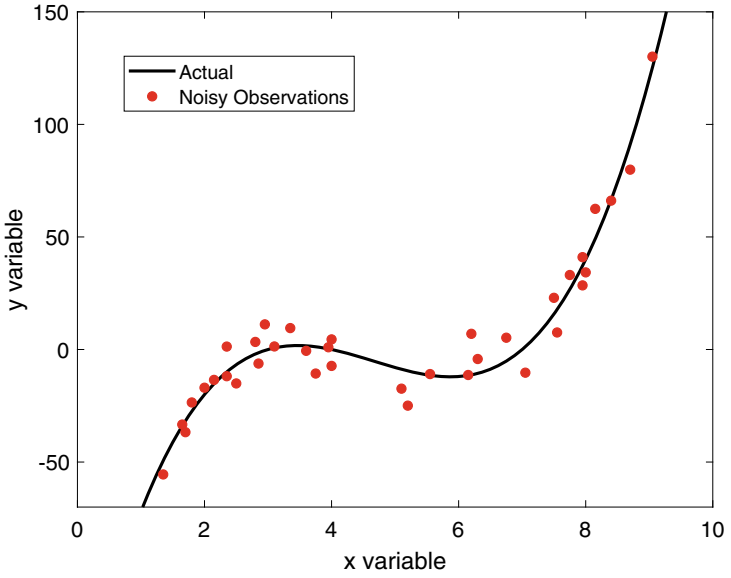


Fig. 10.10 A set of noisy observations are generated from the curve used to illustrate the decision tree regression

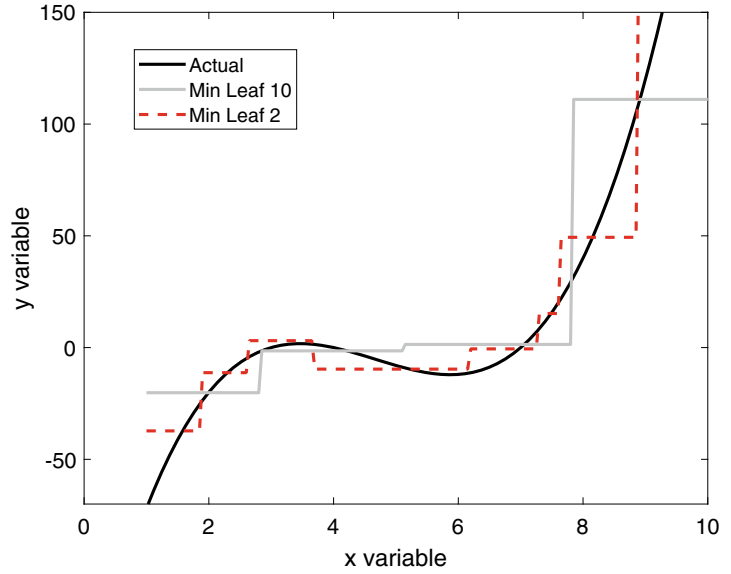


Fig. 10.11 Two regression trees fit to the noisy observations together with the original curve. The regression trees use a different minimum number of observations in the leaf nodes. In this case 10 (grey curve) and 2 (red dashed curve)

Notice the regression trees have finer resolution and the regressions have better matching when the observations are more densely packed. In particular, it should be noted that the ends of the function (at $x < 2$ and $x > 8$) are not accurately estimated. Like many machine learning techniques, the estimates may not accurately extrapolate to points outside the domain of the observations. This can make such methods difficult to estimate outside of the training data in forecast applications.

10.3.2 Random Forest Regression

As mentioned in Sect. 10.3.1, decision trees are often not useful as time series forecasting models and typically produce a models with high variance (see Sect. 8.1.2). However, their power comes from being used as building blocks for other, more powerful methods. One of the most common of such methods is **random forest regression** (RFR) described in this section.

The basic premise of RFR is to generate many regression trees but only applied to a random sample (usually sampled with replacement) of the observations. Further, unlike normal regression trees, each tree only splits on a subset of the variables/features. The final RFR is then an average of the regression functions across all trees generated. By only using a sample of the features in each split, the algorithm prevents the regression from being overtrained on strong predictions and causing correlated trees. Thus each of the regression trees (also called weak learners) focuses on different input features. Random forest is an ensemble technique because it considers an 'ensemble' of weak learners to produce a single strong learner.

Now consider the example from Sect. 10.3.1, with the observations used to train the regression trees given in Fig. 10.10. A random forest regression applied to this data using 100 regression trees is shown in Fig. 10.12. Notice in comparison to the individual regression tree as shown in Fig. 10.11 the RFR fit is much more accurate as well as much more continuous than the regression trees. This is because by randomly sampling the training data and also the variables used in each split, the RFR finds a balance between generalising the function and not overfitting, in other words regression trees often produce a model with a good bias-variance trade-off (See Sect. 8.1.2). This fit would be even smoother if more trees where used on more data.

There are many different parameters in the random forest that can be optimised via cross-validation (see Sect. 8.1.3) with some of the most important being:

1. Number of trees. The more trees the more accurate the model. However, this effects how long it takes to generate the estimates.
2. How many variables/features to select at each node split. For regression a common approach is to select a third of the attributes at each node split. It is best to not use too many variables to avoid overfitting.
3. Minimum number of observations in the terminal/leaf nodes.

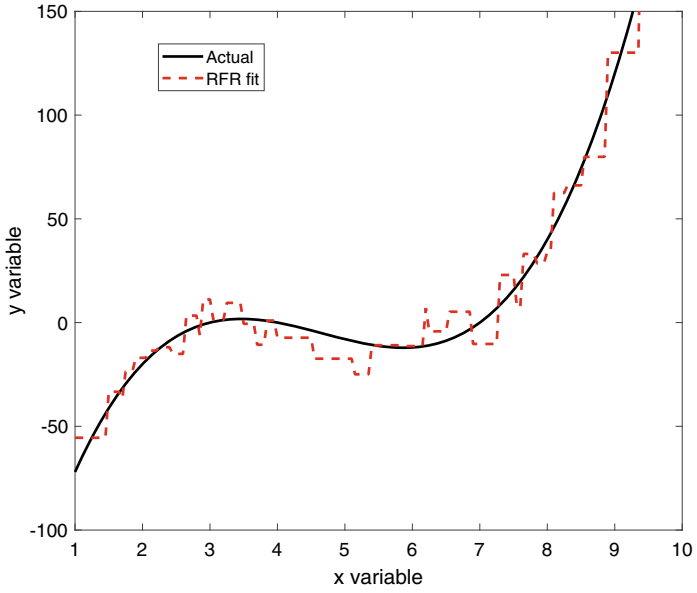


Fig. 10.12 Random forest regression fit to the data from the example in Sect. 10.3.1 generated from 100 regression trees. Also shown in black is the original data from which the observations were generated

The idea for cross validation is to try a large number of regression trees with different selections of the above parameters and choose the mix of parameters which gives the minimum MSE (10.18) on the validation set.

A useful property of random forests is the feature importance tool which can look across all trees to assess the importance of each feature. This can be achieved because not all trees use all variables. Hence a comparison can be made which compares the improvement produced when a feature is included in a model versus when it is not used. For regression, a measure is made of how much the feature reduces the variance. This average across trees gives the final importance of each feature and also helps to interpret the strongest drivers for accurately predicting the outputs.

Random forest is popular because it is easy to implement while maintaining a good bias-variance trade-off. They also have a number of other advantages. They can handle thousands of input variables without overfitting and can be used together with the feature importance to perform feature selection. However, their main disadvantage for use in time series forecasting is that they are not very effective at predictions for out-of-sample data. To illustrate, consider the example in Fig. 10.12. Any estimates outside of the observed domain [1, 10] have the same fixed constant values and are unlikely to be accurate.

10.3.3 Gradient-Boosted Regression Trees

The former section introduced random forest regression, a powerful prediction model that uses an ensemble of simple Decision tree models to produce an accurate forecast. **Gradient-boosted Regression Trees (GBRT)** are also an ensemble technique, using the similar basic idea of combining weak learners to create an accurate strong learner.

There are several related models, but most are variations of the **Gradient Boosting Machine (GBM)** introduced in [2], also referred to as Multiple Additive Regression Trees (MART) and the Generalised Boosting Model. Within random forests, simple decision tree regression models are trained in parallel, and their predictions are combined, e.g., through averaging. In contrast, with GBRT, the base learners are trained in sequence, each trained to reduce the remaining errors in the residual series of the prior iterations. In other words, the main idea of gradient-boosting models is to iteratively improve the model by training new learners that explicitly improve on the current predictions according to some loss function. The optimisation process is guided by the loss function's **gradient**. In a regression problem, like load forecasting, the loss is typically defined as the mean squared error (Eq. (10.18) in Sect. 10.3.1), while in classification tasks, it is the cross entropy. However, gradient boosting is general enough to minimise arbitrary differentiable loss functions. This makes it applicable also for more complicated tasks like predicting quantiles by minimising the quantile loss (see Sect. 7.2).

More precisely, the main objective of gradient boosting is to find the prediction via a weighted sum of weak prediction models which can be represented as

$$f(\mathbf{X}) = \sum_{i=1}^{M} \gamma_i h_i(\mathbf{X}).$$

Finding the optimal weights γ_i and weak learner functions h_i is a computationally infeasible optimisation problem in general. Hence, gradient-boosting finds a solution iteratively with the aim of improving the model over M stages. After each stage i in stages $1, 2, \ldots, i, \ldots, M$ the aim is to find a model f_i that produces an improvement compared to the model of the previous iteration f_{i-1} by adding a new estimator h_i, i.e.,

$$f_i(\mathbf{X}) = f_{i-1}(\mathbf{X}) + \alpha\gamma_i h_i(\mathbf{X}).$$

Here, α is a constant **step-size** or **learning rate** (Sect. 4.3). The gradient-boosting process is guided by the direction of the steepest descent of the loss function (also Sect. 4.3). So let's consider as an example the aim to minimise the mean square error of the ground truth and the last iterations prediction f_{i-1},

$$L_{MSE} = \frac{1}{2}(\mathbf{L} - f_{i-1}(\mathbf{X}))^2$$

Note, the constant factor $\frac{1}{2}$ is introduced for convenience (and without losing generality) to express the loss function's derivative as:

$$\frac{\partial L_{MSE}}{\partial f_{i-1}(\mathbf{X})} = \mathbf{L} - f_{i-1}(\mathbf{X})$$

Define $h_i(\mathbf{X})$ as this derivative of the loss function:

$$h_i(\mathbf{X}) = \mathbf{L} - f_{i-1}(X)$$

In the case of GBRT, regression trees are used to model function h_i. Observe that in the case of the mean square loss, this means the regression tree h_i is being fit to the residuals of the last iterations forecast. The weight γ_i is determined by solving the optimisation problem of plugging the last iterations forecast f_{i-1} and the current iterations residual-fitted model h_i into the mean square error loss function:

$$\arg\min_{\gamma} = \frac{1}{2}\left(\mathbf{L} - \left(f_{i-1}(\mathbf{X}) + \gamma h_i(\mathbf{X})\right)\right)^2$$

The details of solving this optimisation are not part of this book but additional reading is referenced in Appendix D. Generally, the optimisation implemented within gradient-boosting is related to gradient descent (recall Sect. 4.3). The process can be derived similarly for loss functions other than the mean square error, but this discussion is not explored in this book.

GBRT and its variants typically have two types of hyperparameters: ones related to the above-mentioned iterative gradient boosting optimisation process and ones related to the regression trees. One of the most important hyperparameters is the number of regression trees. Similarly to random forests, the depth of each individual tree (sometimes indirectly controlled by a parameter enforcing a lower bound on the number of samples in a leaf) is also relevant. In terms of the gradient boosting optimisation process, the most important hyperparameter is the **learning rate** α or, in the case of gradient-boosting, also referred to as **shrinkage**. It determines how much each newly added tree contributes to the model. So smaller values make the model more robust against influences of specific individual trees, i.e., allow the model to generalise better and avoid overfitting. But a small learning rate requires a larger number of trees to converge to the optimal value and hence is more computationally expensive.

As introduced in Sect. 8.2.5, an important diagnostic tool to evaluate these hyperparameters is the so-called deviance plot that shows the training and testing error as a function of the number of trees. Figure 10.13 shows such a plot. The error on the training set decreases rapidly and then gradually slows down but continues to decrease as further trees are added. In contrast, the error on the test set also decreases but after slowing down and reaching a minimum the loss begins to increase again. This increasing gap between training and test error indicates overfitting of the model and

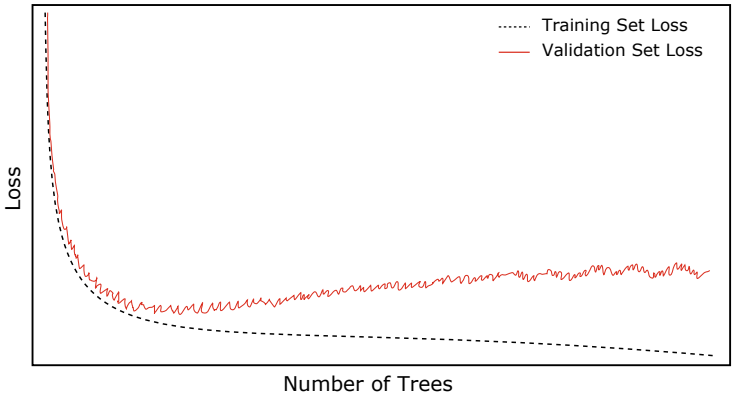


Fig. 10.13 Deviance plot of the relationship of the number of trees and the train and generalisation error as a diagnostic tool

the ideal point is determined by the learning rate and the number of trees. Compare this plot to the more general version of the plot, Fig. 8.5, introduced in Sect. 8.2.

Gradient boosting has a high model capacity and is hence prone to overfitting. Therefore other regularisation parameters may need to be tuned using cross-validation. Similar to Random Forests regression regularisation can be implicitly introduced by fitting models only on a subset of the features and instances, i.e., through **subsampling**. This can be controlled through parameters that limit the number of features and the share of instances used. Different gradient boosting implementations may provide additional explicit mechanisms to prevent overfitting that usually introduce more hyperparameters. Popular choices are, for instance, L1 and L2 regularisation on the weights (see Sect. 8.2.4 on regularisation) and early stopping that stops training if the loss is not improved above a certain threshold after a certain number of iterations (Sect. 8.2.5).

Note that gradient boosting is a general approach that can also be used with other base learners. However, it has become most popular to use decision and regression trees because they are relatively simple and efficient to train, hence not prone to overfit as a base learner, but can still already model non-linear relationships with interactions between the features. Due to the good performance of the approach for tabular data, many different related versions and implementations of the general GBM algorithm [2] have been introduced. See Appendix D for additional reading and the most popular implementations of the gradient-boosting framework. While it may seem discouraging to use gradient boosting methods due to the large number of hyperparameters, they are among some of the most powerful methods for accurate predictions on tabular data and, therefore, also in load forecasting. Unfortunately, the forecast accuracy comes at the cost of limited model transparency. As will be discussed in Sect. 10.6, tree-based methods like random forests and gradient boosting provide scores to assess feature importance. However, this should be seen merely as

an indicator, and the methods don't provide any understanding of the actual effect size of specific variables or their significance, in contrast to methods such as linear regression (cf. Sect. 9.3).

10.4 Artificial Neural Networks

This section introduces **artificial neural networks (ANN),** a machine learning technique loosely inspired by biological neural networks, the building blocks of animal and human brains. ANNs consist of a collection of connected artificial neurons, and like the synapses in the brain, each artificial neuron can send a signal to neighbouring, connected neurons via dendrites. As in biological neuronal networks, "learning" is achieved by adjusting the connection between the neurons. However, the detailed mechanisms, such as the learning mechanism itself (i.e., backpropagation) or the representation as real numbers, are quite different from the biological role model (which is not yet fully understood). Nevertheless, ANNs are a powerful machine learning method which are being applied in numerous applications, from forecasting to image recognition, and many advancements are rapidly being developed. This section introduces the standard form, the feed-forward network, and an adaption designed to handle sequential data called recurrent neural networks.

10.4.1 Feed-Forward Neural Networks

The simplest building block of ANNs is the **artificial neuron**. It is often referred to as a **node**, a **unit** or a **cell** of an ANN. A neural network with one artificial neuron and no hidden layers is called a **perceptron**. The perceptron can be used as a supervised learning algorithm that can learn nonlinear decision boundaries (classification) or functions (regression). Figure 10.14 illustrates how the perceptron, a single artificial neuron, can be used to forecast the load L_t at time t based on n input variables

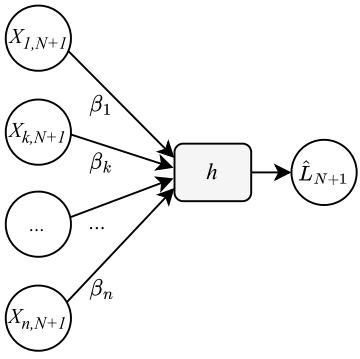


Fig. 10.14 A simple artificial neuron or cell

$X_{1,t}, \ldots, X_{n,t}$ which also correspond to the same time t. The collective n inputs can be denoted as the vector $\mathbf{X_t}$.

The artificial neuron must train the function $h(\mathbf{X_t})$ so that when it operates on the inputs, $\mathbf{X_t}$, it produces an accurate estimate of the final output. The output of an artificial neuron is called an **activation**. To compute the activation, the inputs are linearly combined and passed into an **activation function** g to produce the output signal (the activation):

$$\hat{L}_{N+1} = h(\mathbf{X}) = g\left(\sum_{k=1}^{n} \beta_k X_{k,N+1}\right) = g\left(\beta^T \mathbf{X}_{N+1}\right) \tag{10.19}$$

Note that technically a constant **bias term** is also added, but this is omitted here to improve readability. This can be achieved by concatenating a variable $X_{0,t} = 1$ to the input vector $\mathbf{X_t}$.

In a perceptron, if the activation function, g, is ignored, this is just a multiple linear regression (compare the Eq. (10.19) with Eqs. (9.4) and (9.5)). However, the activation function introduces nonlinearity and increases the flexibility of the model compared to a simple linear regression. There are many choices for the activation function. Popular choices are the sigmoid function, hyperbolic tangent (tanh) and, more recently, versions of the ReLU function. Figure 10.15 shows some popular activation functions and their corresponding derivatives.

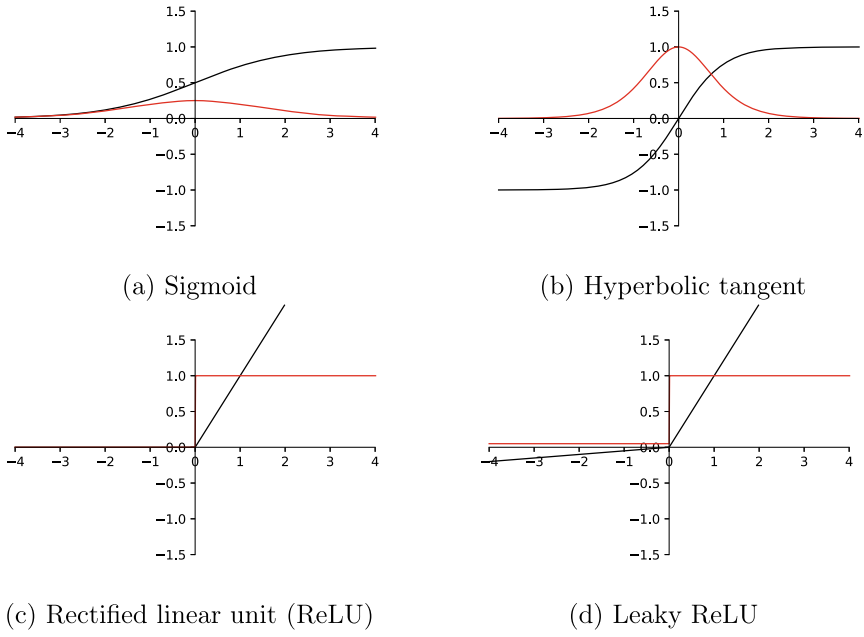


Fig. 10.15 Comparison of popular activation functions (black) with their respective derivatives (red)

The **sigmoid function** is a nonlinear transformation function that maps values to the unit interval [0, 1]. This makes it a popular choice in neural networks since it can be directly used in the output layer of binary classifiers, as its output can be interpreted as a probability of being a member of one of the categories (Sect. 3.1). The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + \mathrm{e}^{-z}}, \tag{10.20}$$

However, it is comparatively computationally expensive and can cause training stability issues when used within a neural network model. This function is now generally only used for binary classification problems. A more generalised form of the sigmoid, the **softmax function**, is used in multi-class classification.

The **hyperbolic tangent** (tanh) is another popular choice as a neural network activation function defined as

$$\tanh(z) = \frac{(\mathrm{e}^{z} - \mathrm{e}^{-z})}{(\mathrm{e}^{z} + \mathrm{e}^{-z})}. \tag{10.21}$$

It is of a similar S-shape as the sigmoid function but is defined between -1 and 1 and maps negative values to negative outputs and zero inputs to zero. However, it has similar stability issues as the sigmoid function and is hence seldom used in modern network architectures.

The **rectified linear unit (ReLU)** is very efficient to compute and does not lead to the same stability issues as the sigmoid and tanh functions, namely the **vanishing gradient problem** (see Sect. 10.4.2). This has made it the default activation function in many deep neural networks. It maps values below zero to zero but is equal to the input itself when it is greater or equal to zero, i.e. a linear activation, and is defined as:

$$\mathrm{ReLU}(z) = \max(0, z). \tag{10.22}$$

The function and its derivative are both monotonic.

The fact that ReLU maps values below zero to zero and does not map larger activations to smaller numbers leads to different stability issues, namely the **dying ReLU** where many activations are zero, or **exploding activations** when repeated activation leads to increasingly larger values. An adjusted version, the **leaky ReLU**, attempts to solve the dying ReLU problem and can pose as an alternative when ReLU produces stability issues in model training. It introduces a positive parameter α as a multiplier for negative values and is defined as:

$$\text{Leaky ReLU}_{\alpha}(z) = \max(\alpha \cdot z, z). \tag{10.23}$$

In the case of the perceptron, the activation of the neuron represents the final prediction $\hat{L}_t$. However, the true power of neural networks comes from stacking several **layers** of neurons, where the activation of one layer can then be passed to the

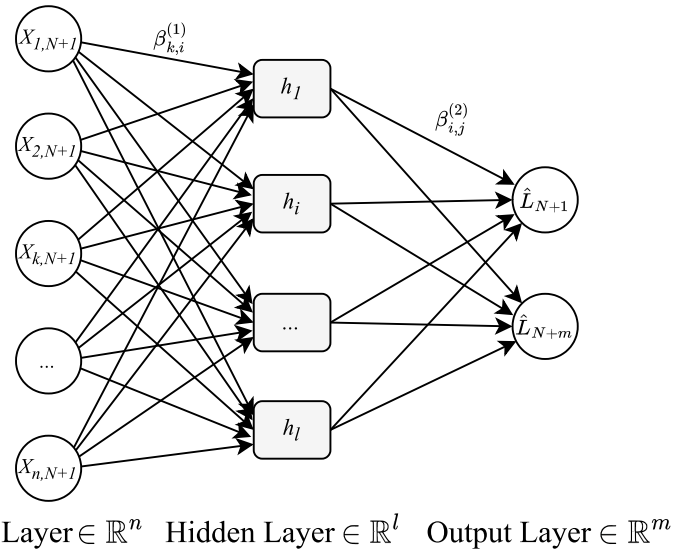


Fig. 10.16 The directed graph structure of a feed-forward neural network

next layer in the network, enabling later layers to use the prediction of earlier layers as a feature. This is how neural networks can learn increasingly abstract representations of simpler features.

Feed-forward networks are the simplest form of multi-layer neural networks. In a feed-forward neural network, each node in a layer is connected to all the nodes in the previous and successive layers (therefore also referred to as a **fully-connected** neural network). They are also referred to as **multi-layer perceptrons** or as **vanilla neural networks**, as in "vanilla" being the plainest and standard kind of ice cream.[2]

The **input layer** has one node per feature in the dataset. The **output layer** has one node per target variable (in multivariate regression) or class (in classification). The layers in between are referred to as **hidden layers** with l hidden neurons. Figure 10.16 shows this basic structure with one hidden layer.

In the context of load forecasting, the ANN is used to forecast future load. In the example shown, the output consists of m values, which in the application of this book would normally be an estimate of the demand for m steps ahead, i.e., the load $L_{N+1}, \ldots, L_{N+m}$. To achieve this prediction, it takes several features as input. In the case of load forecasting, this could, for instance, be past values of the time series itself as well as some past (and possible forecasted) explanatory variables, e.g., the outside temperature.

To understand how ANNs work, consider trying to accurately predict the load L_t at time $t = N + 1$ using n inputs $X_{1,N+1}, X_{2,N+1}, X_{k,N+1}, \ldots, X_{n,N+1}$ by training a model f using the ANN framework.

[2] Whoever came up with that term has not tried the vanilla ice cream in the authors street.

In other words, the aim is to model the following relationship

$$\hat{L}_{N+1\,N} = f(\{X_{1,N+1}, \ldots, X_{n,N+1}\}, \boldsymbol{\beta}), \tag{10.24}$$

where $\boldsymbol{\beta}$ are the weights of the ANN (which will be described below). As in previous cases, the inputs can be historical loads or other explanatory variables. In the context of time series forecasting, the input $X_{k,N+1}$ typically include the prior values of the target $L_N, L_{N-1}, L_{N-2}, \ldots, L_{N-W}$ to model the autocorrelation with past values of the time series. In the context of ANN the amount of historical values up to W is sometimes referred to as **perceptive field** to mirror the biological analogue. Additionally, the features typically include also some other external features related to time step $t = N + 1$ (cf. Sect. 6.2 for more on typical features). ANNs that include past values of the load $\mathbf{L}$ and the external values $\mathbf{X}$ are sometimes referred to as a **nonlinear autoregressive exogenous model (NARX)**.

For simplicity, the following discussion considers only one hidden layer with l nodes as in Fig. 10.16, the extension of the algorithm to further layers is analogous. The activations of the hidden layer can be calculated similarly to the activation of the individual perceptron (see Eq. (10.19)) but extended to each neuron i of the layer:

$$h_i(\mathbf{X}) = g\left(\sum_{k=1}^{n} \beta_{k,i}^{(1)} X_{k,N+1}\right) \tag{10.25}$$

Then we can write the definition of the full neural network as:

$$\hat{L}_{N+j} = g_o\left(\sum_{i=1}^{l} \beta_{i,j}^{(2)} h_i(\mathbf{X})\right), \tag{10.26}$$

where g_o is the activation function applied to the linear summation of the outputs from the hidden layer.

A neural network with a single hidden layer with a large number of units has the ability to approximate very complex functions. To use a neural network for prediction, one needs to determine the optimal values for the weights in the graph, here the ones connecting the input to the hidden layer $\beta_{k,i}^{(1)}$ and from the hidden layer to the outputs $\beta_{i,j}^{(2)}$. As with supervised learning more generally (see Sect. 4.2.1), the aim is to find the parameters of the model which minimise a **loss function**, often denoted as J. In regression tasks, as encountered in load forecasting, this is most typically the mean squared error (MSE), as defined in Eq. 10.18 for decision trees. For ANN the MSE loss function can be written as a function of the current weights of the neural network $\boldsymbol{\beta}$ with Equation (10.15) and some known ground truth $\mathbf{L}$:

$$J_{MSE}(\mathbf{L}, \boldsymbol{\beta}) = \frac{1}{N} \sum_{t=1}^{N} (L_t - f(\mathbf{X}_t, \boldsymbol{\beta}))^2. \tag{10.27}$$

In the process of finding the optimal weights, this ground truth is the training data. This process of **training** the neural network is done by initially starting with random values. Then **batches** of instances of the training set are passed into the neural network and all the layers' activations are calculated. The loss function and its gradient are computed. The weights are updated in the direction of the gradient in order to minimise the loss. Since the updates are calculated layer-wise, propagating backwards from the output layer to the input layer, this process is sometimes called **back-propagation**.

Whereas in linear regression this optimisation can be done in closed form based on the whole dataset, or through a simple least squares regression (Sect. 8.2.1), the task of finding the optimal weights in neural networks is more complex. Recall from Sect. 4.3 that this loss function is typically non-convex, i.e., it can have multiple local optima and saddle points. The weights are therefore adjusted using an optimiser as described in Sect. 4.3.

From the description above, there are a number of different choices in designing the ANN, i.e., choosing its hyperparameters, including

- The number of nodes per hidden layer,
- The number of hidden layers,
- The choice of activation function,
- The choice of optimiser and its hyperparameters.

Increasing the number of layers and nodes increases the number of parameters in the system and increases the chances of overtraining the model. This can be avoided by the same techniques as discussed in Sects. 8.1.3 and 8.2. One option is to choose the correct parameters and functions via cross-validation techniques, as discussed in Sect. 8.1.3, in which several models are trained with different combinations of the number of nodes and layers. The trained models can then be compared to each other based on their performance on the validation set. This process could be expensive, especially if training lots of models. An alternative method is to use regularisation as demonstrated in Sect. 8.2.4. These methods involve adding a penalty to the cost function proportional to the weights' size, which encourages the parameters to stay small (hence reducing the complexity of the ANN). Another method to prevent overtraining is to use **early stopping** (Sect. 8.2.5), which stops the algorithm early to prevent the ANN from training too close to the noise in the data set. The choice of iteration to stop can also be decided by using cross-validation.

The activation functions depend on the application. For hidden layers, as discussed, common functions are the sigmoid or the tanh function. In deep neural networks the rectifier linear unit (ReLU) is the most popular choice. For the output layer the choice is determined by the type of problem. In binary classification the sigmoid function is used and in multi-class classification the softmax function. Then the loss function is the cross entropy loss. In regression the last layer is linear (i.e., no activation) and the loss is the mean squared function. For the optimiser and their hyperparameters see Sect. 4.3 for popular choices.

10.4.2 Recurrent Neural Networks

The machine learning models in the last sections have mainly concentrated on fixed-length input data. When including past observations to compute the functional form as in Eq. (10.15), a window length W or receptive field must be specified to determine how many past values to include. This is the case because all the regression models considered are designed to handle **tabular data**, i.e., datasets of fixed-size input vectors. Further, the algorithms generally do not assume any structure over the columns, i.e., in a structured dataset, the order should not matter.[3] Given the fixed length, one cannot efficiently model dependencies that require a specific order of the columns, which often is the case for sequential data due to autocorrelation. Further, if one chooses a large W, one needs a lot of data to be able to model dependencies that exist at very different time scales.

However, when dealing with time series and other sequential data, e.g., DNA sequences, video analysis, sound patterns and language, it may make sense to be less restrictive on the length of input values to model both long and short-term dependencies. Instead of specifying the length of the input, i.e., the receptive field that should be considered, the model needs to learn the relevant length.[4]

Recall the architecture of a feed-forward neural network (cf. Fig. 10.16). The network consists of fully-connected layers, and every node is connected to every node of the next layer. The structure of the network can be represented by a directed acyclic graph. Recall that in NARX sequential data is added in the form of the lagged values of the load $\mathbf{L}$ and some external features $\mathbf{X}$. Despite inputs potentially being sequences of arbitrary length, the input $X_1, \ldots, X_N$ is required to be sequential of a fixed dimension n. As discussed before, this is because the fully-connected neural network can only compute the function $f(X_1, X_2, \ldots, X_N)$ on these fixed-length inputs. However how could a more flexible $f(X_1, X_2, \ldots, X_N)$ be calculated for variable values of N?

For that the inputs can be calculated **recurrently** by feeding the vector sequentially and passing in each step, not only the current value of X, X_k but also the value of the activation of the prior step, Z_{t-1}, i.e.:

$$Z_t = h(Z_{t-1}, X_t), \text{ for } t = 1, 2, \ldots, N \tag{10.28}$$

The final prediction is then the output of the final calculation:

$$f(X_1, X_2, \ldots, X_N) = Z_N \tag{10.29}$$

Here, X_t can be a vector $\{L_t, X_{1,t}, \ldots, X_{n,t}\}$ of the load L and n features, e.g., weather or calendar variables that belong to the time step t considered.

[3] Hence, the *set* in dataset.

[4] Note learning about lengths of input was partially seen with the exponential smoothing method described in Sect. 9.2 which discounted older observations by determining a decay factors which needed to be learnt.

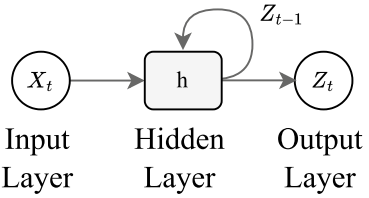


Fig. 10.17 A graph structure with recurrent connections of a recurrent neural network

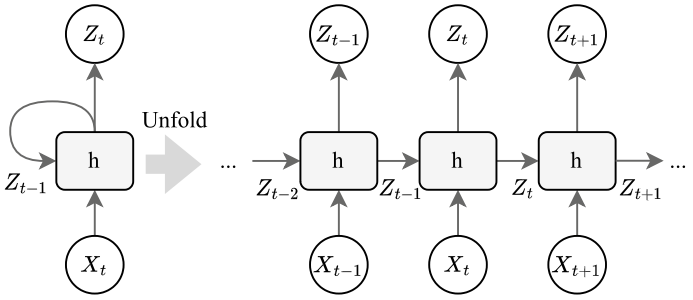


Fig. 10.18 An unfolded recurrent neural network

Recall, that h is an artificial neuron that applies an activation function introducing nonlinearities. Then Z is the activation, sometimes referred to as the intermediate **hidden state**.

Neural networks that consist of such **recurrent connections** are called **recurrent neural networks (RNN)** and they are designed to capture the dynamics of sequences more appropriately than regular feed-forward neural networks. Figure 10.17 shows the structure of such an RNN with the recurrent connection shown as a loop.

This activation over time can be thought of as multiple copies of the same network, each passing an activation to a successor. This makes RNNs "deep" neural networks, even if technically only one layer is modeled.[5] RNNs can be thought of as feed-forward neural networks where each layer's parameters (both conventional and recurrent) are shared across time steps. While the standard connections in a feed-forward neural network are applied synchronously to propagate each layer's activations to the subsequent layer at the same time step, the recurrent connections are propagating the activation also to the nodes in the same layer but over time. This can be visualised as in Fig. 10.18 by providing an unfolded view of the activations over time.

To understand how the RNN neuron, henceforth referred to as a cell, computes the new activation based on the old activation and a new value, see Fig. 10.19. It visualises this activation within the cell by showing that the input X_t and the activation of the

[5] However, note more layers can also be explicitly modelled in an RNN.

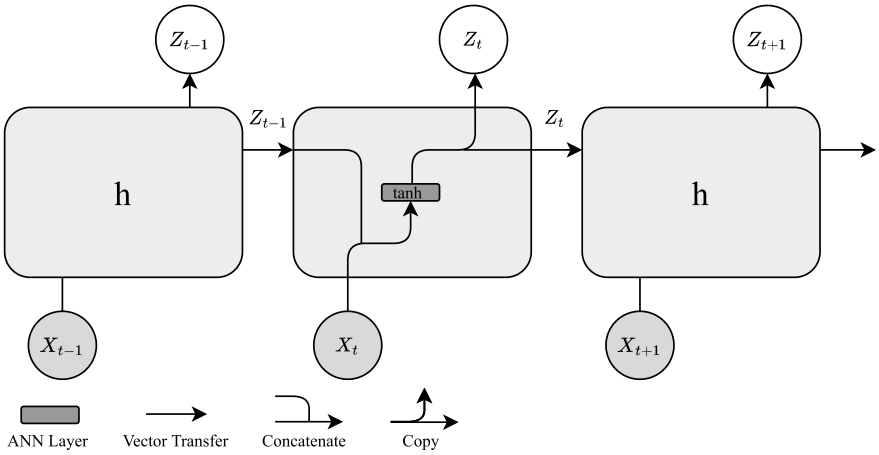


Fig. 10.19 A visualisation of the internals of an RNN cell with the tanh activation function

prior time step Z_{t-1} are concatenated and then passed to the activation activation function g, which in the context of standard RNN is often the tanh function. So the activation can be written as:

$$\mathbf{Z_t} = g\left(\boldsymbol{\beta}^T[\mathbf{X}_t, \mathbf{Z_{t-1}}]\right) = tanh\left(\boldsymbol{\beta}^T[\mathbf{X}_t, \mathbf{Z_{t-1}}]\right) \tag{10.30}$$

Again $\boldsymbol{\beta}$ denotes a weight vector, g the activation function and we use $[\bullet]$ to denote the concatenation of the vectors $\mathbf{X_t}$, the feature vector, and $\mathbf{Z_{t-1}}$ the activation of the prior step t.

One can see from Eq. (10.30), that the recursive call of the activation function can lead to **vanishing and exploding activations** and more importantly their gradients when computing the loss function and its gradients in model training, i.e., finding the optimal weights of the network. For instance, consider only the first three steps, this leads to

$$\mathbf{Z_3} = g\left(\boldsymbol{\beta}^T\left[\mathbf{X}_3, g\left(\boldsymbol{\beta}^T\left[\mathbf{X}_2, g\left(\boldsymbol{\beta}^T\left[\mathbf{X}_1, \mathbf{Z_0}\right]\right)\right]\right)\right]\right) \tag{10.31}$$

Here, repeated multiplication with the weights can lead to very small or very large values. While this can be alleviated by tricks such as gradient clipping, standard RNN tends to be unstable to train when longer dependencies are modelled. RNNs are typically only successful in modelling short-term dependencies. Hence, they have not proven practical for load forecasting where longer dependencies like weekly or even yearly seasonal patterns are typical. Therefore, in this section hyperparameters of the standard RNN model are not further discussed, and the introduction to RNN serves only as the background for more modern variants such as LSTM and GRU, which have been popular for sequence modelling and have proved successful for

load forecasting. In particular, their training is more stable than the standard RNN. These more modern techniques will be explored in more detail in Sect. 10.5.

10.5 Deep Learning

The algorithms so far in this chapter are considered classical machine learning algorithms. This section introduces neural network architectures that are considered **deep neural networks** or as part of the subfield of machine learning called **deep learning**. While the notion of "deep" neural networks has been touched upon in the context of RNNs, where "deep" meant deep in time, it has been found that RNNs in their standard form are not able to model long-term dependencies in time. In this way they are more similar to a standard feed-forward neural network. This chapter introduces adaptations to RNNs that make them capable of modelling more long-term dependencies, hence can be considered neural networks with many layers, i.e., deep.

It should be noted that there is no clear definition of when an artificial neural network is considered "deep". Recall the architectural graph from ANN in Fig. 10.16. The number of weights increases exponentially with the number of new layers. One way to distinguish standard feed-forward networks from deep neural networks is that deep neural networks often have so many layers that fully connected layers are infeasible.

But why add many layers in the first place? This is due to a second way of distinguishing classical machine learning from deep learning. In classical machine learning, the modelling flow is to first hand-design features and then fit a model that maps from the features to the target. As discussed in Chap. 4, in classical statistical modelling the goal is to avoid the curse of dimensionality and only include variables that help to understand the process. However, in machine learning, we care about making the best possible prediction and in deep learning, the objective is to find suitable **feature embeddings** or **representations** automatically that can be used by a predictive model. When stacking several layers, lower layers learn more straightforward representations that are passed to subsequent layers that can use these simple features to model more abstract features to be used in the final prediction model. This process is often also referred to as **representation learning**.

While this novel modelling flow of automated representation learning is now the default in disciplines such as computer vision and language modelling, where manual feature engineering has been predominantly displaced, for time series and load forecasting, often manual features are still a suitable approach, especially in settings where one does not have an abundance of data available.

10.5.1 Modern Recurrent Neural Networks

Recall from the last section that RNNs have issues due to numerical instability, leading to them only being capable of modelling short-term dependencies. This section introduces the two most popular extensions of RNNs, namely **gated recurrent units (GRUs)** and **long short-term memory (LSTM)**. Recall from Sect. 10.4.2, in contrast to layers in feedforward networks, a layer in recurrent neural networks receives the input of the input layer $\mathbf{X}_t$, as well as the activation signal from the last time step of itself $\mathbf{Z}_{t-1}$, which was referred to as a hidden state.

LSTMs introduce a second hidden state, the **cell state**. Now the current state of a cell depends on the current value $\mathbf{X}_t$, as well as on the previous activation $\mathbf{Z}_{t-1}$ and the previous cell state $\mathbf{C}_{t-1}$. This cell state functions as a memory of the cell where the training determines how long- and short-term values should be memorised. To control how much of the input to forget, LSTMs introduce the **forget gate**, the **input gate** and the **output gate**. Figure 10.20 gives an overview of these parts of the LSTM. Note how the gates are essentially made up of ANN layers, i.e., weights and different activation functions.

Figure 10.21 gives an overview of each of these gates. Figure 10.21a shows the forget gate that governs how much to keep from the previous cell state and how much to add from the current input of the previous activation. The last activation and the input are concatenated and passed through the sigmoid function that brings it to between 0 and 1. With the pointwise multiplication, this means the closer values are to 0, the more the cell state "forgets". The closer the value is to 1, the more is kept. It's hence computed as:

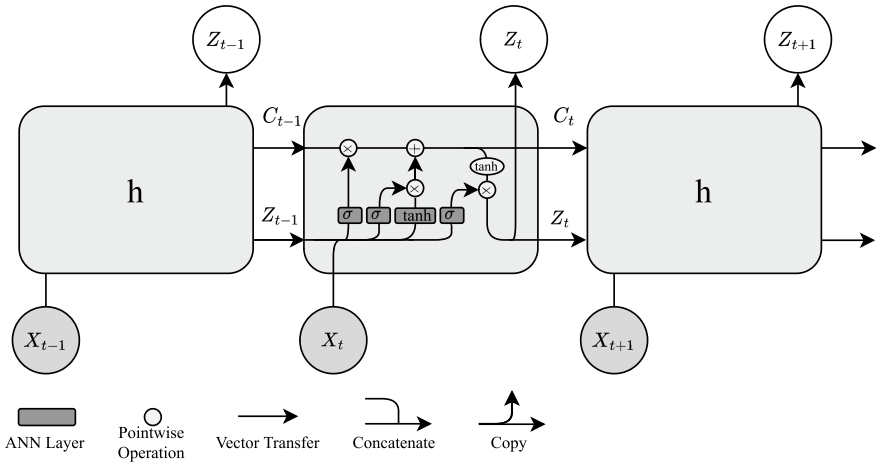


Fig. 10.20 An overview of an LSTM cell unrolled in time

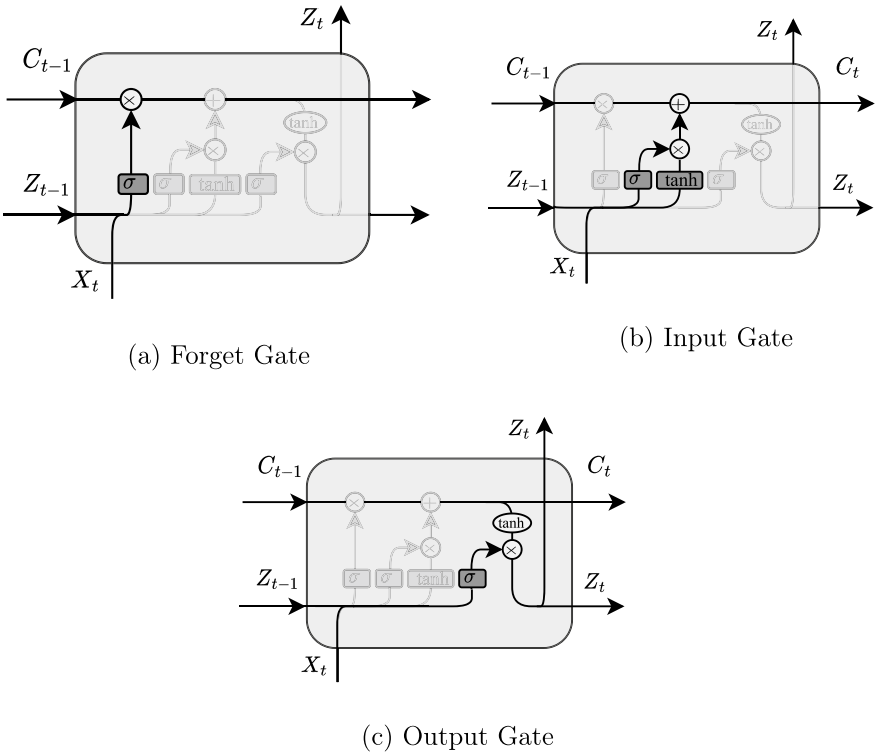


Fig. 10.21 The LSTM cell with the cell state and different gates highlighted

$$\mathbf{F_t} = \sigma\left(\beta_F^T[\mathbf{X}_t, \mathbf{Z_{t-1}}]\right) \tag{10.32}$$

Note that there is a weight vector $\boldsymbol{\beta}_F$ that is specific to this gate.

The next part, as shown in Fig. 10.21b, is called the input gate. It adds or subtracts from the current state. It computes input value $\mathbf{I}_t$ and a candidate value $\tilde{\mathbf{C}}_t$ as:

$$\mathbf{I_t} = \sigma\left(\beta_I^T[\mathbf{X}_t, \mathbf{Z_{t-1}}]\right)$$
$$\tilde{\mathbf{C}}_t = tanh\left(\beta_C^T[\mathbf{X}_t, \mathbf{Z_{t-1}}]\right)$$

Then the updated cell state $\mathbf{C}_t$ is computed by:

$$\mathbf{C}_t = \mathbf{F_t}\mathbf{C}_{t-1} + \mathbf{I_t}\tilde{\mathbf{C}}_\mathbf{t} \tag{10.33}$$

Again, note the weight vectors $\boldsymbol{\beta}_I$ and $\boldsymbol{\beta}_C$ that must be determined in the training process. Note, that by adding the activation to the values prior in time before feeding it into the tanh activation to compute the next activation, this is related to residual skip

connections described in Sect. 10.5.3, as the network can learn if there is something useful that should be added from the current input, or if the old cell state should be kept.

The final part is called the output gate that learns what to output from the cell state as well as the former hidden state and the current input as the next hidden state:

$$\mathbf{O_t} = \sigma\left(\beta_O^T[\mathbf{X}_t, \mathbf{Z_{t-1}}]\right)$$
$$\mathbf{Z_t} = \mathbf{O_t} * tanh\,(\mathbf{C}_t)$$

With these gates, LSTMs can be trained to be more stable than standard RNNs and have been popular in sequence modelling and also for time series and load forecasting. Over time, several variants have been proposed, like peephole connections that give each of the gates access to the current cell state and coupled forget and input gates that, instead of separately deciding what to forget and keep, make those decisions jointly. A full discussion of what parts are necessary or the most effective is not part of this book. For more information see [3, Chap. 10].

The most popular related architecture is the **Gated Recurrent Unit (GRU)** cell. The main idea is similar to that of LSTMs, as it similarly introduces gates to control how much to remember from previous states. However, it is a little bit simpler and has fewer parameters. It does not have a dedicated cell state and introduces the reset and update gates. Figure 10.22 gives an overview of the GRU cell. We omit a detailed walkthrough from this book as it is simar to the LSTM. Generally, it is simpler than the LSTM and is hence faster to train and less prone to overfitting (e.g., when used with time series). See [3, Chap. 10] for a description of GRU.

So far, only a single hidden layer has been discussed. In practice, multiple LSTM or GRU layers can be stacked on top of each other. However, this increases the

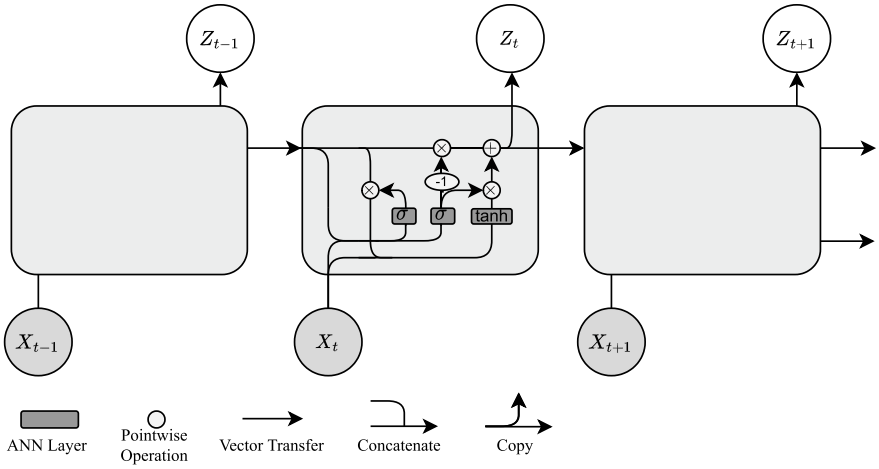


Fig. 10.22 An overview of a GRU cell unrolled in time

number of parameters drastically and may lead to overfitting. Several layers should be explored, when there is a lot of data available, e.g., when fitting a global model trained on the data of several households, buildings or other metered instances. So in terms of hyperparameters, the design decisions are similar to feedforward neural networks, namely the number of layers and the number of hidden units per layer. The activation functions are as introduced in the descriptions before. Further, an optimiser and its hyperparameters need to be chosen.

While more stable than RNN, LSTM and GRU remain difficult to train and may lead to overfitting for time series. For longer relationships, up to, e.g. hundreds of steps back—not uncommon, for instance, with weekly seasonality—both LSTM and GRU can get quite deep for practical applications. One hundred steps back in time can be interpreted as a standard feedforward network with 100 layers. Generally, LSTM and GRU are still slow to train, as they are not easy to parallelise as the states have to be computed sequentially.

10.5.2 Convolutional Neural Networks

The beginning of this section has motivated the idea that stacking many layers can enable learning of increasingly complex representations of the input data. Consider modelling a high-resolution time series, for instance, 1-minute load data, with a long receptive field. This leads to a large number of lagged values that need to be included in the model to capture both the short-term local patterns and long-term trends. With a regular fully-connected neural network, this would require connecting each input neuron with a large number of neurons in the next hidden layer. Each node in the hidden layer is, in turn, connected to each neuron in the next hidden layer (and so forth). In particular, where there is multi-dimensional input, for instance, in other domains like images or even videos, then even a few hidden fully-connected layers would be infeasible as the neural network would have an excessive number of parameters.

One of the main drivers of the recent surge of machine learning has been the success of **convolutional neural networks (CNN)** that cope with a large number of parameters by using a different architecture. For instance, to decide if an image contains a rabbit, a CNN can make use of the fact that it does not need to view the full image at once, but can instead view successively smaller parts of the image, as it does not matter where in the image the rabbit is. The architecture makes use of so-called **invariances**, namely **locality** and **translational invariance**. Standard CNNs make use of the successive stacking of **convolutional layers** and **pooling layers**, which will be explained below.

Convolutions can identify patterns in data points that are close together. In images, for instance, adjacent pixels are close as they are also close in the physical world that the image represents. Similarly, for many time series, neighbouring data points are also near, as certain behaviours may occur close in time. This locality can be exploited by convolutions. A convolution is a mathematical operation defined through two

functions (in the continuous case) or matrices and sequences (in the discrete case). This section will henceforth only consider the discrete case due to the inherent discrete time steps of the load data which is analysed in this book. Note also, that only 1-dimensional convolutions will be considered in this book due to the focus on univariate time series data.

Consider two sequences $\mathbf{X} = (X_0, X_1, \ldots, X_{n-1})$ and $\mathbf{K} = (K_{-p}, K_{-p+1}, \ldots, K_0, \ldots, K_p)$ of length n and $2p + 1$ respectively. The notation for index of K starting at $-p$ is because this simplifies the later calculations as will be shown.

The convolution $(\mathbf{X} * \mathbf{K})_n$ is calculated by reversing the order of one of the vectors (which notice will be easier to do with $\mathbf{K}$ due to the notation used above!) and taking a sliding dot product with the other sequence, $\mathbf{X}$. In other words the convolution creates a new sequence $\mathbf{Z} = (Z_0, Z_1, \ldots, Z_{n-1})$ defined as

$$Z_n = (\mathbf{X} * \mathbf{K})_n = \sum_{m=-p}^{p} X_{n-m} \cdot K_m. \tag{10.34}$$

Notice that in this summation it may require values which are outside the index of the defined sequences. In this case those values are simply set to zero (this is referred to as **zero padding**). For example

$$Z_0 = X_0 K_0 + X_1 K_{-1} + \cdots + X_{p-1} K_{-p} \tag{10.35}$$

While generally there is no constraint on the length of either sequences, i.e., both can be of the same length, in the context of neural networks, one is typically longer (here the input sequence $\mathbf{X}$) and one shorter (the so-called **filter** or **kernel**,[6] $\mathbf{K}$). Figure 10.23 shows schematically how the convolution function can be used to calculate a target sequence $\mathbf{Z}$. In summary, the steps to compute a convolution are as follows:

1. Reverse the kernel sequence $\mathbf{K}$, then
2. shift the kernel along the input sequence $\mathbf{X}$ one point at a time, and
3. at each step, calculate the dot product of the two aligned sequences, i.e. multiplying the aligned values and adding these products.

The resulting sequence $\mathbf{Z}$ is the convolution of the kernel and the input sequence. In the context of convolutional neural networks, the result may be referred to as **feature map**, or more generally, it is a **representation** of the input data in the context of **representation learning**.

Note that formally a convolution is defined over indices from negative infinity to positive infinity. In practice, one adds **padding** of zeros on both sides as illustrated

[6] Note, as this book combines topics from statistics, classical machine learning and deep learning, the term *kernel* has appeared in this book already in the context of kernel density estimation in Sect. 3.3 and in the context of support vector machines in Sect. 10.2. While in each of those contexts, it is related to the notion of a function, unfortunately, in each context the term kernel has different distinct meanings.

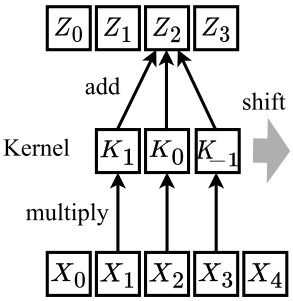


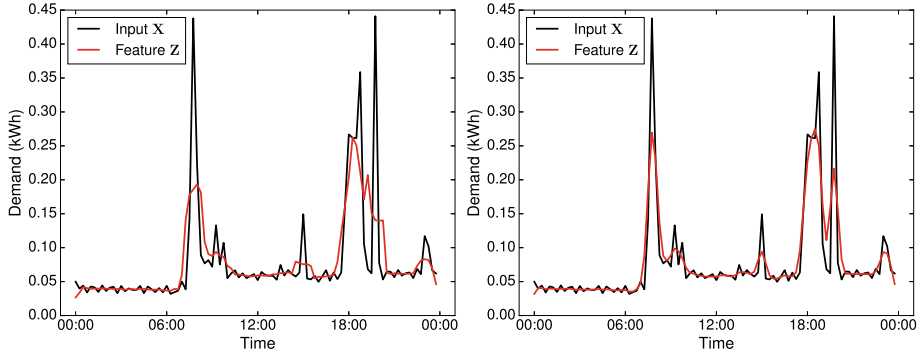
Fig. 10.23 Schematic of the convolution operation

above in equation (10.35). Then one can choose to keep only the part of the convolution that is non-zero or limit the results to just those values where the kernel completely overlaps with the input sequence.

To understand the effect of the convolution operation on typical input sequences, consider the example in Fig. 10.24. It shows a load profile at the household level as input $\mathbf{X}$ and the resulting feature map $\mathbf{Z}$. Figure 10.24a considers the effect of the application of a filter $\mathbf{K} = [0.2, 0.2, 0.2, 0.2, 0.2]$. Given that the weights add up to 1 and with the Definition (10.34), it becomes clear that this is simply a moving average of the values before and after the current value. It's essentially smoothing the profile. A related operation is shown in Fig. 10.24b. The kernel $\mathbf{K} = [0.05, 0.24, 0.40, 0.24, 0.05]$ represents a Gaussian distribution, i.e., the centre point is weighted more than the edges. This also leads to an average, but the shape of the original load profile is more strongly preserved. In image processing, this operation is often called a **Gaussian blur** and is considered a more natural average filter than simply using the equally weighted filter. Finally, Fig. 10.24c shows the result of a kernel designed to highlight variation between neighbouring data points. In the context of images, this would detect edges. In the context of load profiles, it highlights the sudden increases and decreases in load.

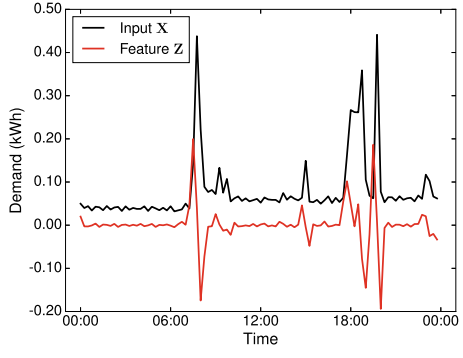
These filters are not defined manually in a convolutional neural network. Similarly to weights in a feed-forward neural network, the filters' values are determined by the optimiser in the training process. The output, i.e., the feature map, is then passed through an activation function so that it can be thought of as analogous to the activation in feed-forward neural networks. It can be regarded as the learned representation passed to subsequent layers. There are as many feature maps as filters after each layer. The filter can be thought of as a feature extractor as the optimisation process will enforce filters that specialise in finding specific recurring features, like the above-mentioned averaged profile or the highlighted edges, that are helpful for downstream layers. Filters in the first layers could learn basic shapes, such as edges or corners, while later layers can detect more complex compositional patterns.

The second operation commonly used in convolutional neural networks is **pooling**. 1D pooling is effectively down-sampling the input sequence using an aggregation function. This aggregation function can be the mean or, more commonly, the max function. Figures 10.25 and 10.26 show this schematically. In Fig. 10.25 the pooling



(a) Averaging the input with filter $\mathbf{K} = [0.2, 0.2, 0.2, 0.2, 0.2]$

(b) Averaging with Gaussian filter $\mathbf{K} = [0.05, 0.24, 0.40, 0.24, 0.05]$



(c) Edge filter $\mathbf{K} = [0.5, 0.0, -0.50]$

Fig. 10.24 The effects of applying different kernels to a load profile at the household-level

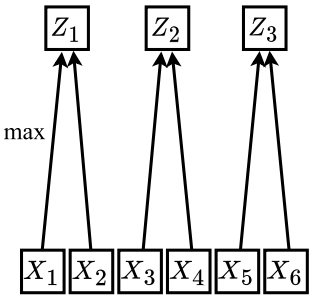


Fig. 10.25 Max pooling operation with pool size 2 and stride 2

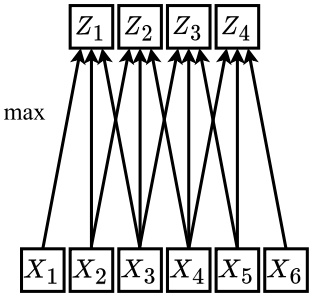
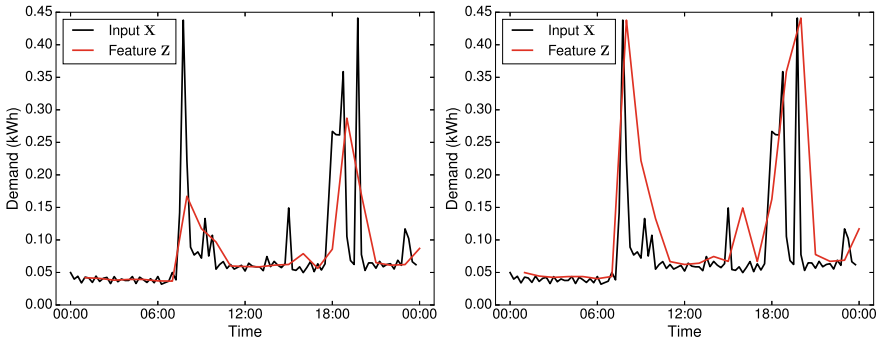


Fig. 10.26 Max pooling operation with pool size 3 and stride 1

factor or **pool size** is 2, i.e., two values are averaged. Hence, the final sequence is half the length of the input sequence. In Fig. 10.26, the pooling factor is 3, but instead of shifting the pooling operation by the pool size, it is only shifted by one step, the so-called **stride**.

Similar to above, consider the example in Fig. 10.27. It shows the same load profile as before as input $\mathbf{X}$ and the result of applying the pooling operation with both a pool size and stride of 4. On the left, it shows mean pooling. This operation is typically done when downsampling a load profile, here from a 15 min to 1 h resolution. Each point is the average of the current and prior 3 values. For a profile at the household level, this smoothes the profile and the distinct peaks that are related to high-power appliances that are only used briefly, like a kettle or hair drier. The right figure shows the max pooling operation. It more strongly preserves the peaks of each of the considered intervals compared to the averaging pooling. Note, that each of the sequences is now one-fourth of the length of the original sequence.

A convolutional layer consists of passing the resulting feature maps through an activation function. These building blocks, **convolutional layers** and **max pooling layers** are the essential parts of CNNs. Figure 10.28 shows how they can be stacked for sequences as input, in the same manner as in more common 2D, 3D and 4D architectures, as they are used, for instance, in image and object recognition. After stacking several convolutional and max pooling layers, a CNN typically flattens and concatenates the last layer's activation and feeds it into a fully-connected neural network that makes the final prediction as described in Sect. 10.4.1 using the feature representations extracted by the convolution and pooling layers.



(a) Mean pooling operation.

(b) Max pooling operation.

Fig. 10.27 The effects of applying different pooling functions to a load profile at the household level (here with pooling size 4 and stride 4)

The trainable parameters of CNNs are, therefore, the filters of the convolutional network layers and the weights of the fully connected layer. Note that the pooling layers are having no trainable parameters but are merely downsampling the output of the convolution layers. As the convolutional layers essentially function as a feature extractor for the fully connected network, it is possible that filters trained on one dataset are used for a completely new dataset without retraining but only training the fully connected layers. This is referred to as **transfer learning** or **fine tuning** (see also discussion in Sect. 13.4). The flattened output of the convolutional layers can further be concatenated with additional features denoted as $\mathbf{X}$ to condition the forecast on more external covariates where convolutional operations are not useful. For instance, [4] show that for residential load forecasting, it can improve the forecast to condition it in this way on calendar-based variables and the weather forecast.

CNNs have several hyperparameters and architectural choices. First of all, the number of convolutional and pooling layers. For convolutional layers, the number of filters and the filter size, i.e., the length of the kernel, are the important hyperparameters. It is common practice to choose odd filter sizes, as this makes implementation easier. Often 3 or 5 are reasonable choices. ReLU is most commonly used as an activation function in the convolution layers. For the pooling layers, the max function is the most common choice. Here, only the pool size needs to be chosen. In 2D, a pooling size of 2 reduces feature maps in both dimensions, e.g., to a quarter of the input size, which is, therefore, a reasonable choice. Similarly, for 1D time series, a size of 4 can also be a suitable initial choice. Then, finally, the structure of fully connected layers, the number of nodes per layer and their activation functions need to be chosen analogously to fully-connected neural networks. However, ReLU is a reasonable default. Again, as in other neural network models, an optimiser and its hyperparameters need to be chosen (see discussion in Sect. 4.3) as well as other parameters affecting training like the batch size and the maximum number of epochs to train.

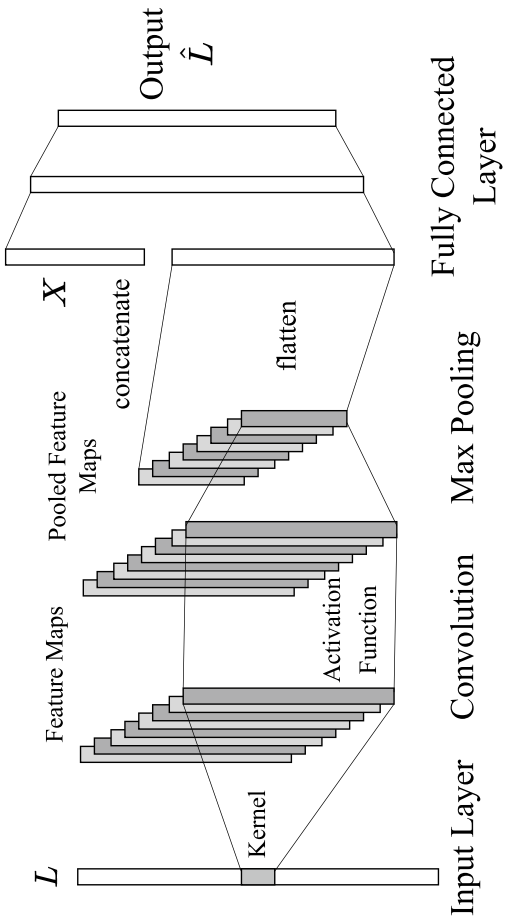


Fig. 10.28 The structure of a convolutional neural network for time series data

CNNs, in general, have several strengths over recurrent neural networks. They can model long-term dependencies better than LSTM, as 100 steps back do not make the model "deeper" and thus they avoid the numerical issues discussed earlier. Further, they are much faster to train, as the calculation of filters is trivially parallelisable since one filter does not depend on the others. Also, as discussed, in CNN, parts of the architecture can be reused for new, but similar tasks. This process of transfer learning never really worked as well for LSTM and GRU. However, the transfer of a **pre-trained** model to a new task has proven an effective strategy in practice, that can improve generalisation and drastically decrease training time for a new task, i.e., effectively saving energy and, therefore costs and even CO_2 emissions (cf. discussion on sustainable AI [5]).

However, in this standard form, CNNs have several problems in the context of time series. First, with many filters and several layers, a model can still be comparatively large and have many parameters that can overfit with time series, especially for large receptive fields. Hence, some more modern building blocks that made the training of very large CNN architectures possible have also been introduced to time series and will be discussed in the next section. Another problem with time series is the convolution operation itself. As one can see from the definition and Fig. 10.23, the convolution operation includes future values when calculating the dot product of the filter. For time series forecasting, this means that future values can leak from the test period into the training data, whereas this data is supposed to be unknown at the time of the forecast, and can produce an overly optimisatic prediction. For these reasons, it is therefore recommended to use adjusted versions of CNNs for load forecasting, as will be introduced in the next chapter.

10.5.3 Temporal Convolutional Networks

This section discusses adjustments to convolutional neural networks that have proven effective for working with time series, namely **causal convolutions**, **dilated convolutions**, **residual skip connections** and $\mathbf{1 \times 1}$ **convolutions**. While those have been features of the WaveNet architecture [6] that has been introduced as a generative model for handling raw audio data, neural network architectures based on these features for more general time series tasks have since been referred to as **temporal convolutional networks (TCN)** [7]. In the following, we describe the most important building blocks.

The most important adjustment is made to avoid the possibility of leaking future data into the training data (see Sect. 13.6.4 on data leakage). While one could add zeros as padding instead of future values, there is a better solution: **causal convolutions**. For that, the convolution operation is simply shifted to only consider prior values when calculating the sliding dot product with the flipped kernel (cf. Sect. 10.5.2). Figure 10.29 shows this schematically (compare it to regular convolutions in Fig. 10.23). To formally define the operation, consider input sequence

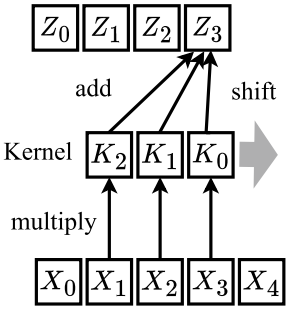


Fig. 10.29 Causal convolution operation

$\mathbf{X} = (X_0, X_1, \ldots, X_{n-1})$ and kernel $\mathbf{K} = (K_0, K_1, \ldots, K_{k-1})$ of length n and k respectively. Then, we can then define causal convolution as:

$$Z_n = (\mathbf{X} *_c \mathbf{K})_n = \sum_{m=0}^{k-1} X_{n-m} \cdot K_m. \tag{10.36}$$

A second problem that was discussed before is that a large receptive field, i.e., a large window of past values to include in the model, may lead to unnecessary many parameters as it requires too many filters, and it may be unnecessarily slow, as many convolutions may need to be computed. In traditional modelling, one addresses this by manually deciding to feed only relevant past values, e.g., only of the last day, the same day of a week ago, the same day two weeks ago, or similar. However, in deep learning, we would still want to pass a considerable amount of past values and have the model learn relevant features, i.e., internal representations, automatically. To achieve that, TCNs make use of **dilated convolutions** that solve this by introducing a **dilation factor** d that essentially adds steps between calculations of convolution operations. With this, it is still possible to cover a large receptive field by stacking several layers of dilated causal convolutions. Figure 10.30 compares regular causal convolutions with dilated convolutions, highlighting the operations that are necessary to cover the same receptive field with full causal convolutions (left, with filter size $k = 4$) and causal dilated convolutions (right, filter size $k = 2$ and dilations $d = [1, 2, 4]$). The same receptive field can be covered with a smaller filter size (i.e., fewer parameters) and much fewer operations (i.e., much faster training). The dilated causal convolution operation is then defined for the same input sequence $\mathbf{X} = (X_0, X_1, \ldots, X_{n-1})$ and kernel $\mathbf{K} = (K_0, K_1, \ldots, K_{k-1})$ as:

$$Z_n = (\mathbf{X} *_d \mathbf{K})_n = \sum_{m=0}^{k-1} X_{n-d \cdot m} \cdot K_m. \tag{10.37}$$

One important idea to improve CNNs that is common in TCNs are **residual skip connections** that were popularised with ResNet [8]. Prior to these, there were limits to stacking more layers since training becomes unstable and accuracy reduces as

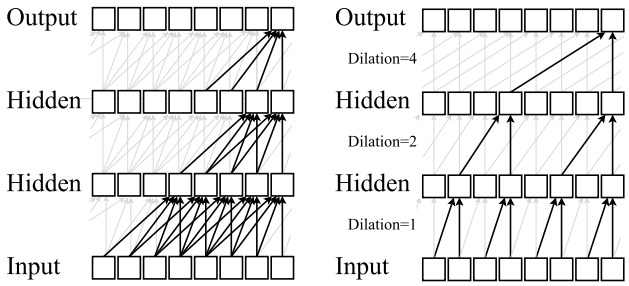


Fig. 10.30 Comparison of standard and dilated causal convolutions

more layers are added. However, in theory, adding more layers should not degrade performance, as ideally, if more layers would lead to worse performance, it would be preferable for the layer to simply learn the identity mapping, hence, not change performance. Skip connections allow for layers to essentially be set to the identity mapping, i.e., skipped, by adding the activation of previous layers with activations of successive layers. For dense layers, this can be achieved by concatenating those activations. More commonly, this is achieved by addition. So essentially, skip connections allow the model to choose whether layers add value to the outcome or not, thus improving results in large neural network architectures. Recall that this is similar to how the input gate works together with the cell state of prior time steps in LSTM (see Sect. 10.5.1).

Another idea that is common in modern TCNs is **1×1 convolutions** that were popularised with the Inception model [9]. A 1×1 convolution helps reduce the dimensionality along the filters. When applied to only one feature map (or input), it simply scales each pixel by a constant weight. This would by itself not be useful. However, when applied to several filters, it essentially learns a linear projection of the filter maps, thus reducing the number of filters. This reduces the number of parameters, whilst retaining some feature-related information through the learned weights.

TCNs have similar hyperparameters to CNNs generally (see Sect. 10.5.2). However, when using dilated convolutions, two possible hyperparameters can be used to increase the receptive field: the dilation factor d and the filter size s. The most important decision is the architecture, i.e., how the building blocks are connected. Figure 10.31 shows the TCN architecture as introduced in [7]. One TCN residual block contains two dilated causal convolution blocks, each followed by weight normalisation, a ReLU activation and a dropout layer (see Sect. 8.2 on these regularisation techniques). These two blocks are bypassed by a residual skip connection with a 1×1 convolution. If the block is in the first layer, it receives the input sequence, else it receives the activation of the prior block $i - 1$, Z_{i-1}, and passes its activation Z_i to the next block. Note, that in the literature and in libraries, details in the implementation of what is referred to as TCN may vary. An architecture based on WaveNet has been used for residential load forecasting in [10]. In [11], TCNs are used for

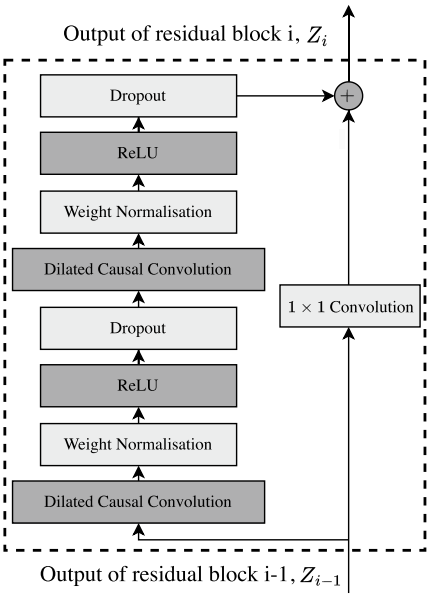


Fig. 10.31 The structure of a TCN residual block as introduced in [7]

load forecasting at the system scale. [12] use only the causal dilated convolutions as part of a CNN architecture for fitting the parameters of different probabilistic models for residential load forecasting. Note, that modern TCNs don't make use of pooling layers that are popular in image and object recognitions. Instead, dilation and 1×1 convolutions are used to decrease the number of parameters.

Compared to LSTMs and GRUs, TCNs are more efficient to train (see discussion in the last section on CNNs). Compared to standard CNNs, TCNs solve, most importantly, the issue of causality. But several ideas like dilation, residual skip connections and 1×1 convolutions have been shown to improve regular CNNs empirically, as they make training more stable, enable more efficient training and require fewer parameters. This can be beneficial for time series, where deep machine learning models are generally prone to overfit in settings without a lot of training data.

10.5.4 Outlook

As the last section has shown, with more modern deep architectures, it becomes less and less clear what is working in what situations. The field of deep learning is moving very rapidly with the successes in image and language modelling and some of these models are being utilised within the field of time series using more and more specific architectures. Many of the building blocks for such models have been introduced in the previous sections.

One example, N-BEATS [13], is a model based on fully connected residual networks, and has been successful as part of the M5 time series forecasting competition,[7] placing second. It includes an interpretable version that enforces individual stacks to learn the trend and seasonality independently. Another popular time series model is DeepAR [14], an RNN architecture that has been introduced for probabilistic intermittent demand forecasting. It fits a global model across different products and predicts one step ahead based on previous step values and some covariates (e.g., weather). It uses a Gaussian or Negative Binomial likelihood function, and its parameters are predicted by the neural network.

A whole different approach, not using the aforementioned blocks, is **transformer models** [15]. They have surpassed all other approaches in text processing in most tasks and have had initial successes working with image tasks. They may be well suited for time series as well, as compared to CNNs they are truly sequential, i.e., no perceptive field needs to be determined. This allows them to handle different length inputs, which is only possible with CNNs and TCNs using zero padding. Like RNNs, transformers are designed to process sequential input data. Some consider them to be a version of recurrent neural networks, however, unlike RNNs, transformers process the entire input data all at once. They use the so-called **self-attention** mechanism to provide **context** for any position in the input sequence. By not having to process one step at a time, it allows for much better parallelisation than RNNs and therefore reduces training times considerably. However, as transformers have not made it into the load forecasting literature (yet), they are not covered in more detail in this book.

This leads to a more general question. Given the many possibilities with deep models, it is unclear where to start. Time series forecasting has for a long time been approached by only statistical methods, as discussed in Chap. 9. Only recently, machine learning models have shown to be successful in certain situations (see the M4 and M5 forecasting competitions [16, 17]). As discussed in more detail in Chap. 12, start simple first! Compared to images and text, many time-series forecasting problems, have lower data availability. So when making predictions for only one instance, like one building, household or substation, and only with the data of that instance (see discussion of local and global modelling in Sect. 13.4), statistical and traditional time series models perform well. They also perform well, when the data is of low resolution (e.g., daily, weekly, or yearly time series), and when covariates like seasonalities and other external influences are well understood. Machine learning and deep learning models tend to perform better when fitting models across multiple time series, like one model for multiple buildings or households or hierarchical models (see Sect. 13.4 on this topic). In probabilistic forecasting, they perform well when densities are complex (e.g., multi-modal). Further, they can be useful when fitting models to processes with complex, non-linear external influences. However, one important finding of the M5 competition was that combinations of statistical and machine learning models can reach state-of-the-art results with the advantage of remaining at least partly interpretable, combining the advantages of both and hence are particularly useful for real-world applications.

[7] https://mofc.unic.ac.cy/history-of-competitions/.

10.6 Feature Importance and Explainable Machine Learning

This chapter introduced several machine-learning models for load forecasting. As discussed, in machine learning the objective is to learn representations for input data to improve the forecast automatically. This comes, however, at the cost of understanding the relationship between the input data and the prediction. As explainable machine learning is a research domain that has many different approaches, and none have shown to be dominant in the load forecasting practice, it will not be covered in depth in this book.

However, this section will briefly discuss the following methods:

- Feature importance of tree-based methods (model specific),
- permutation importance (model agnostic),
- SHAP Values (model agnostic).

As introduced in Sect. 10.3, tree-based forecasting models have **feature importance methods** built in. These models attempt to determine the most relevant features for the internal representation of the data, i.e., fitting the model. Different measures can be used to assess the feature importance, most commonly, Gini importance or the mean decrease in impurity (MDI). These can be output with the model to give some indicator for the feature importance and to explain the model. Note, however, that these are biased towards high cardinality features. Also, they are computed on the training set in the model fitting phase, and may, therefore, not reflect the ability of the feature to be useful to make predictions that generalise to the test set and to the application. So they should merely be seen as an indicator. They work because with tree-based models only a sample of the variables are used to construct each tree (Sect. 10.3). That means a sample of trees do not utilise one of the input variables. This means the performance of using a particular variable can be compared to not utilising that variable. In short, the importance of that feature can be assessed.

Another popular method is **permutation importance**. It was introduced by Breiman in the context of random forests [18] (See Sect. 10.3.2 for more details on random forests). The idea is to randomly permute each feature's values to analyse how it affects the final prediction outcome. This idea is simple and computationally cheap to implement. It can be applied to any fitted model and is not restricted to tree-based methods. While being popular due to the above reasons, they are generally not advised when the dataset contains strongly correlated features, as it may otherwise be biased in favour of correlated features.

Permutation importance and feature importance can be combined to help identify all relevant variables in a input dataset. The **Boruta** algorithm [19] adds duplications of some of the input variables for a random forest model but permutes them. Hence these permuted variables should have no relevance to the dependent variables and its feature importance can be compared to the other (non-permuted) variables to identify which ones have an importance lower than the random inputs. Those with a higher

importance can be viewed as therefore being more relevant to the supervised learning model performance.

Finally, there are **SHAP values** (SHAP stands for SHapley Additive exPlanation). The theory is based on cooperative game theory. A detailed discussion of how they work is not part of this book but see [20] for a more detailed discussion on SHAP values. The output is also a feature importance score. They work well for correlated features, as interactions of variables are also analysed. However, they are expensive to calculate as one step includes building combinations of each of the features. This is, therefore, infeasible for a large number of inputs and hence, often only approximations are calculated.

10.7 Questions

As in the last chapter, for the questions which require using real demand data, try using some of the data as listed in Appendix D.4. Again, ideally choose data with at least a year of hourly or half hourly data and split it into training, validation and testing with a $3:1:1$ ratio.

1. Explain the difference between the activation functions and the loss function in a neural network. How do they relate? Explain how each of them is chosen in the process of modelling a feed-forward neural network for a specific task.
2. The Exponential Linear Unit (ELU) is another activation function that is a strong alternative to ReLU. It is defined as:

$$\text{ELU(z)} = \begin{cases} z, & \text{for } z < 0 \\ \alpha(\text{e}^z - 1), & \text{for } z \leq 0 \end{cases}$$

   Plot the function and its derivative. Discuss possible strengths and weaknesses compared to other activation functions discussed in this chapter.
3. Take an example load profile for a day that has some variation over the day, i.e., some distinct peaks. Use a convolution implementation of a library such as scipy or numpy in Python to compute some more manually selected kernels as was done in Fig. 10.24 and observe their resulting influence on the feature map. Try a kernel $\mathbf{K} = [-1.0, 2.0, -1.0]$. Before implementing, try to predict what the result will look like.
4. Using a neural network library such as Tensorflow[8] or PyTorch[9] implement a simple NARX, i.e., a fully-connected neural network that accepts the last two weeks of a half-hourly load profile as input and outputs a prediction for the next day. How many neurons does the input layer have? How many the output layer? Add one fully-connected hidden layer with ReLU activation. Use the appropriate

[8] See https://www.tensorflow.org/.

[9] See https://pytorch.org/.

activation function for the output layer. Use the library to output the number of weights (or trainable parameters) each network architecture has and visualise it as a function of the number of layers. How does the number of parameters scale with the number of layers? Train your network with different hidden layers (e.g. one and five) and visualise the train and validation error over the number of epochs trained. Observe the different training times needed. Do you observe overfitting for the deeper network compared to the more shallow one? Add dropout of 10%, 20% and 50% to the hidden layers and observe if that changes the progress of the training and validation loss. Try another regularisation method from the ones that have been discussed in Sect. 8.2.

References

1. A.J. Smola, B. Schölkopf, A tutorial on support vector regression. Stat. Comput. **14**, 199–222 (2004)
2. J.H. Friedman, Greedy function approximation: a gradient boosting machine. Ann. Stat. 1189–1232 (2001)
3. I. Goodfellow, Y. Bengio, A. Courville, Deep Learning (MIT Press, 2016). http://www.deeplearningbook.org
4. A. Elvers, M. Voß, S. Albayrak, Short-term probabilistic load forecasting at low aggregation levels using convolutional neural networks, in *2019 IEEE Milan Powertech* (IEEE, 2019), pp. 1–6
5. L.H. Kaack, P.L. Donti, E. Strubell, G. Kamiya, F. Creutzig, D. Rolnick, Aligning artificial intelligence with climate change mitigation. Nat. Clim. Chang. 1–10 (2022)
6. A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, Wavenet: a generative model for raw audio (2016). arXiv:1609.03499
7. S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling (2018). arXiv:1803.01271
8. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778
9. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015), pp. 1–9
10. M. Voss, C. Bender-Saebelkampf, S. Albayrak, Residential short-term load forecasting using convolutional neural networks, in *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)* (2018), pp. 1–6
11. P. Lara-Benítez, M. Carranza-García, J.M. Luna-Romera, J.C. Riquelme, Temporal convolutional networks applied to energy-related time series forecasting. Appl. Sci. **10**(7), 2322 (2020)
12. M. Arpogaus, M. Voss, B. Sick, M. Nigge-Uricher, O. Dürr, Short-term density forecasting of low-voltage load using Bernstein-polynomial normalizing flows. IEEE Trans. Smart Grid (2023). pp. 1–1. https://doi.org/10.1109/TSG.2023.3254890
13. B.N. Oreshkin, D. Carpov, N. Chapados, Y. Bengio, N-beats: neural basis expansion analysis for interpretable time series forecasting (2019). arXiv:1905.10437
14. David Salinas, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Deepar: probabilistic forecasting with autoregressive recurrent networks. Int. J. Forecast. **36**(3), 1181–1191 (2020)
15. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need. Adv. Neural Inf. Process. Syst. **30** (2017)

16. Spyros Makridakis, Evangelos Spiliotis, Vassilios Assimakopoulos, The m4 competition: results, findings, conclusion and way forward. Int. J. Forecast. **34**, 802–808 (2018)
17. S Makridakis, E Spiliotis, V Assimakopoulos, The m5 accuracy competition: results, findings and conclusions. Int. J. Forecast (2020)
18. Leo Breiman, Random forests. Mach. Learn. **45**(1), 5–32 (2001)
19. M.B. Kursa, W.R. Rudnicki, Feature selection with the Boruta package **36**, 1–13 (2010)
20. S.M. Lundberg, S.I. Lee, A unified approach to interpreting model predictions. Adv. Neural Inf. Process. Syst. **30** (2017)

Chapter 11
Probabilistic Forecast Methods

The previous two chapters were concerned with point forecasts which only produce a single estimate for each time step in the forecast horizon, i.e. one value L_t for each of the time periods $t = N + 1, N + 2, \ldots, N + k$ (assuming a forecast horizon of length k steps ahead starting at forecast origin N). Point estimates are limited in their description of the future demand, especially when the underlying data has a large degree of uncertainty. A more detailed picture of the possible values of the demand can be produced by estimating the *distribution* of the demand for each period in the forecast horizon. Forecasts which estimate the spread of the distribution are often called **probabilistic forecasts**. That is the subject of this chapter.

11.1 The Different Forms of Probabilistic Forecasts

As introduced in Sect. 5.2 and Fig. 5.4, there are three core forms of probabilistic forecasts which will be explored in this book: quantile forecasts, density forecasts and ensemble forecasts (not to be confused with ensemble machine learning models such as random forest in Sect. 10.3.2). These can be grouped into two core categories: **univariate** (quantile and density) and **multivariate** (the ensemble forecasts). To understand these types, consider the scenario of trying to estimate the distribution of the data for the time steps $t = N + 1, N + 2, \ldots, N + k$.

For a univariate forecast, the aim is to estimate the distribution of the demand at each time step. In other words, estimate a total of k univariate distributions. The distributions at different time steps are independent of each other, meaning that the spread of the variable at one time step is not influenced by information about the demand at other time steps. Another way to say this is there are no *inter-dependencies* modelled. The univariate Gaussian, as introduced in Sect. 3.1 is the most famous example of a univariate distribution function. There are two main forms for estimates of univariate distributions, either a full continuous density function, or discrete values (quantiles)

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_11

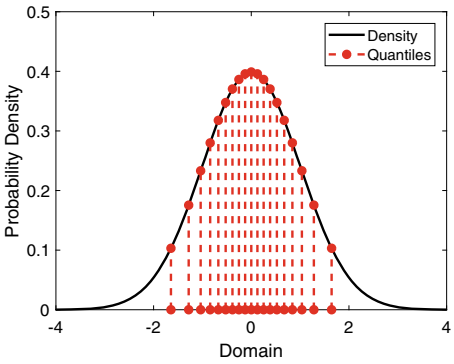


Fig. 11.1 Example of both Gaussian distribution as described by its density function (solid line) and 20-quantiles (red dotted lines)

defining uniformly spaced levels of equal probability.[1] These two forms are illustrated in Fig. 11.1 for a standard Gaussian distribution. The density estimate is often preferable, as it describes the entire distribution, but requires either knowing/assuming that the distribution is from a particular parametric family (e.g. Gaussian in this case), or requires training relatively expensive methods, such as Kernel Density Estimates which will be described in Sect. 11.5. The quantile estimates (described in more detail in Sect. 3.2) in Fig. 11.1, show the 5, 10, . . . , 95% quantiles for the Gaussian density, and are clearly less descriptive of the distribution, however they are a lot less expensive to compute and don't rely on assuming a specific distribution of the data. Since a univariate distribution has to be estimated for each of the k time steps in the forecast horizon this reduction in computational cost can be particularly advantageous.

For multivariate forecasts the task is instead to estimate a single multivariate distribution for all k demand variables in the forecast horizon (see Sect. 3.3 for more on multivariate distributions). The advantage of multivariate distributions is that they take into account the *inter-dependencies* over the entire forecast time horizon. To illustrate this, consider the example of household demand. This is mainly determined by the occupants behaviour. If a person gets into work late then they will likely get back from work later, hence their demand shift in the morning will correspond to a shift in the evening. In other words, there is an **interdependency** between the demand in the morning and the demand in the evening due to the link between these two activities. A multivariate forecast can therefore be sampled to produce demand profile scenarios which include these correlations. Thus more complicated and realistic interdependent behaviours can be simulated and utilised to optimise applications such as storage control (Sect. 15.1).

Similar to the univariate case, the full multivariate density can be estimated but it is typically more complicated and difficult to model accurately. The methods are often more computationally expensive, there is fewer packages/resources for fitting them, and there are very few standard parametric multivariate distribution functions which can be used to fit to the data. Instead finite samples from the distribution are

[1] Actually the quantiles do not have to be uniformly spaced but it can often be simpler and more useful to do this.

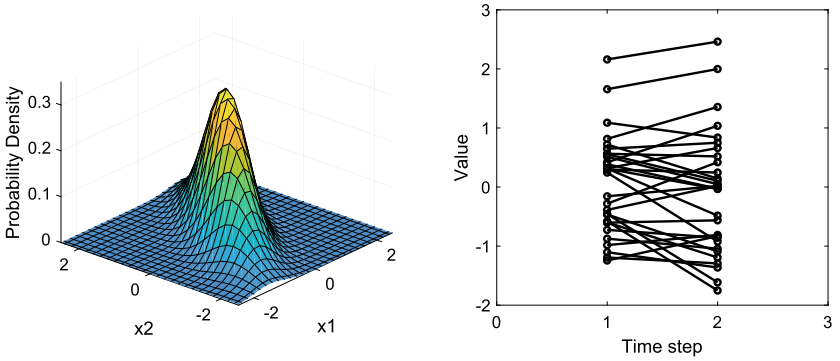


Fig. 11.2 A bivariate Gaussian (left) and 30 ensembles from the distribution (right)

often estimated instead. These *ensembles* are realisations or 'representatives' from the distribution. To illustrate this consider a basic case with $k = 2$ where the estimates at two consecutive time steps $t = 1, 2$ are jointly described by a bivariate Gaussian distribution, shown in the left of Fig. 11.2. Drawing 30 random samples from this distribution gives the time-dependent correlated bivariate ensembles on the right. Notice in the language of distributions introduced in Sect. 3.3 that the multivariate probabilistic forecast is a joint distribution and the univariate probabilistic forecasts are marginal distributions of the full joint distribution.

11.2 Estimating Future Distributions

As discussed in the previous section, the aim of a probabilistic forecast is to estimate the future distribution of the demand whether at a single time step (univariate) or multiple (multivariate). To estimate the uncertainty requires accurately modelling the variation. There are a few standard practical approaches, which will be outlined in the section, and are the basis for many of the techniques in the following sections.

The first approach tries to model the distribution directly by training on the observations. As with most point forecasts these models use the historical data to capture the variation and will typically make assumption about how the past distribution will relate to the future demand. The parameteric models (Sect. 11.3), kernel density estimation (Sect. 11.5) and the quantile regression (Sect. 11.4) all model the distribution in this way.

The aim is to train the parameters or hyperparameters of a distribution model directly (e.g. the Gaussian model) or use a model which will estimate the distribution (e.g. quantiles). The advantage of these approaches is that as long as the right model is used, and they are trained on sufficient data from the target distribution, then they can accurately capture the uncertainty. For example, if we are modeling demand for 2pm and we know that the historic 2pm data all come from the same distribution then this

data can be used to estimate the true distribution. Unfortunately, it is often not known for certain which data comes from the same distribution and so certain assumptions will need to be made based on the analysis of the data. Another drawback of this approach is that its accuracy is correlated with the amount of available data. Small amounts of data will mean a potentially inaccurate estimate.

The second type of model doesn't model the variation directly but instead inserts variation into the model either through adjusting the input variables and/or the model parameters. For example, assume there is a demand model which is dependent on temperature alone. Then the variation in the demand can be modelled by inserting different values of the temperature into the model. Usually these are formed by tweaks on an individual estimate of the temperature and simulates the sensitivity of the demand to the temperature.

This approach is used in numerical weather prediction to produce forecast ensembles/scenarios. Small deviations are applied to the most likely state of the atmosphere and the numerical weather prediction models are reapplied to the adjusted states to produce a range of weather scenarios. Analysis of closeness of the final ensembles can indicate confidence in the future weather states, and widely ranging ensembles may mean there the future weather is highly uncertain.

Alternatively small adjustments can be applied to the model parameters. This accounts for mis-specifications in the model and can generate other likely future states. Multiple adjustments can therefore produce a range of outputs allowing for an estimate of the future distribution. The difficultly with both of these adjustment approaches is that the correct deviations have to be applied to the inputs/parameters in order to produce an accurate distribution estimate. This can be aided in the input case by randomly sampling from the historical observations, or from estimating a distribution from which you can sample.

Another drawback of this model is that the demand variation is not being simulated directly but instead is estimating the sensitivity of the model to the inputs or parameters. Consider the temperature example above. The demand may change with the temperature but in fact it is the variation in the demand for a fixed temperature which is of primary interest (assuming the temperature can be accurately forecast). The key is to add adjustments to the temperature so that it captures this variation. Once again cross-validation is one approach which can be used to determine an appropriate adjustment to the inputs/parameters.

The following sections will mainly focus on the first approach for producing probabilistic forecasts and train the models directly on the historical observations.

11.2.1 Notation

In the following subsections a few probabilistic forecast methodologies are introduced for at least one of each of the three types introduced in Sect. 11.1: quantile, density and ensemble forecast. For the next sections it is worth considering the following notation and conditions.

1. As before consider the demand is represented by the time series $L_1, L_2, \ldots$, where L_t is the demand at time step t.
2. Without loss of generality suppose the aim is to forecast the demand k-steps ahead for the time stamps $t = N+1, N+2, \ldots, N+k$.
3. For univariate probabilistic forecasts: denote the true distributions as CDFs, $F_1(L_{N+1}|\mathbf{Z}), F_2(L_{N+2}|\mathbf{Z}), \ldots, F_k(L_{N+k}|\mathbf{Z})$, one function for each time step in the forecast horizon, i.e. F_t is the univariate distribution of the demand at time step t. Each forecast is conditional on prior information $\mathbf{Z}$ which represents the set of all required dependent variables such as weather, historical demand etc. which determine the future demand. The corresponding CDF forecasts, for each time step $t \in \{1, 2, \ldots, k\}$, are denoted $\hat{F}_t(L_{N+t}|\mathbf{Z})$. For simplicity the $\mathbf{Z}$ may not be included in the notation.
4. For the multivariate probabilistic forecasts the true distribution can be represented by a single CDF, $F_{t=1,\ldots,k}(\mathbf{L}|\mathbf{Z})$ describing the distribution of the multivariate random variable $\mathbf{L} = (L_{N+1}, L_{N+2}, \ldots, L_{N+k})^T$. The prior information $\mathbf{Z}$ contains all dependent variables and the historical loads up to time step N. Often the $\mathbf{Z}$ will not be included for clarity.
5. The mth ensemble of an ensemble forecast will often be denote as $\hat{\mathbf{L}}^{(m)} = (\hat{L}^{(m)}_{N+1}, \hat{L}^{(m)}_{N+2}, \ldots, \hat{L}^{(m)}_{N+k})^T$.

11.3 Parametric Models

Parametric distribution models are desirable as they can give a full description of the spread of the data usually using only a few parameters. This section begins by discussing parametric models via a simple example of a univariate distribution (Sect. 11.3.1). Individual univariate parametric models are usually too inflexible to model the distributions accurately, but families of simple univariate distributions can be "mixed" to estimate much more general shapes and will be introduced in Sect. 11.3.2.

11.3.1 Simple Univariate Distributions

Some simple univariate distributions have already been introduced in Sect. 3.1. The most common being the Gaussian (or Normal) distribution, but the lognormal, and the gamma distribution were also presented. A range of distributions can be modelled using these functions in addition to other similar ones. The advantage of such models is that they only require training a small number of parameters to fully estimate the distribution. However, the restriction to a specific functional form means simple parametric models cannot estimate more complex distributions. For example, the distribution functions mentioned above are all **unimodal** which means they describe distributions with a single modal (maximum) value. It would be impossible to model

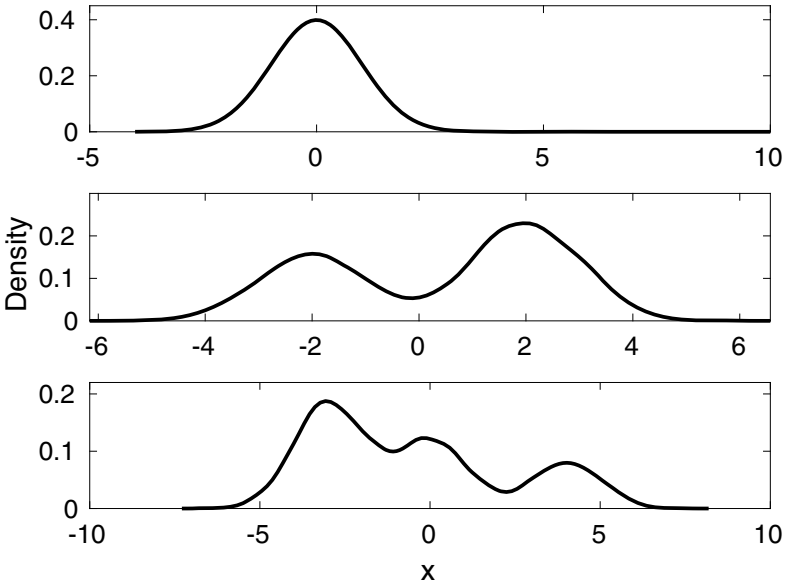


Fig. 11.3 Examples of unimodal (top), bimodal (middle) and trimodal (bottom) univariate distributions

multimodal distributions (distributions with multiple distinct maximum values). An example of a unimodal, bimodal and trimodal distribution are shown in Fig. 11.3.

Although univariate models are unlikely to produce the most accurate univariate probabilistic forecasts they can be useful as benchmark models to compare to more sophisticated approaches described later in this chapter. Further since they are described by relatively few parameters they may be easier to train than non-parametric models. Training parametric models requires estimating each individual parameter which describes the chosen distribution family. For example, a Gaussian will require estimates for the mean and standard deviation, whereas the gamma distribution requires estimating the shape and scale parameters. In the case of the Gaussian distribution the mean and standard deviation can be found by maximum likelihood estimation (Sect. 8.2.1) and these values turn out to simply be the sample mean and sample standard deviation (Sect. 3.5) respectively. To ensure the best possible estimate is produced requires carefully selecting the most appropriate input data to train the parameters (in contrast to data driven machine learning techniques which will learn from all the data). The data can be identified by the analysis methods outlined in Chap. 6. For example, suppose some hourly data is discovered to have strong daily periodicity then it may be appropriate to train 24 different models, each one using only the data from a specific hour of the day.

Parametric models also exist for multivariate models. In particular there is a multivariate version of the Gaussian distribution. As mentioned in Sect. 11.1 these parametric models can be used to produce ensemble probabilistic forecasts over the

h-steps ahead (by estimating a h-dimensional parametric multivariate distribution). Unfortunately, there are much fewer well defined multivariate distributions which can accurately capture a wide variety of probabilistic forecast behaviours. This makes them less suitable compared to more versatile methods which will be introduced in Sects. 11.6 and 11.7 which can also capture interdependencies across time steps in the forecast horizon.

11.3.2 Mixture Models

More versatile univariate distribution can be modelled by combining *mixtures* of the simple parametric distributions discussed in Sect. 11.3.1. The general form of a PDF for a **finite mixture model** of a random variable $\mathbf{x} \in \mathbb{R}^p$ is

$$f(\mathbf{x}) = \sum_{k=1}^{K} \pi_k g_k(\mathbf{x}, \theta_k), \tag{11.131}$$

where $g_k(\mathbf{x})$ are PDF's usually from a single family (e.g. Gaussian's) with their own corresponding parameter's θ_k (e.g. mean and standard deviation for a Gaussian). The π_k are weights which satisfy $\sum_{k=1}^{K} \pi_k = 1$, and are often called **mixing probabilities**. Mixture models are often used for clustering, and in this case each PDF defines a distribution of points from one of the clusters, and the weights signify what proportion of the observations are in each cluster.

The most common mixture models for continuous variables use Gaussian components, i.e.

$$g_k(\mathbf{x}) = \frac{1}{(2\pi)^{N/2} det(\mathbf{\Sigma}_k)^{1/2}} exp\left((\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right), \tag{11.132}$$

with covariance $\Sigma_k \in \mathbb{R}^{p \times p}$, and mean vector $\boldsymbol{\mu}_k \in \mathbb{R}^p$. This is called a Gaussian mixture model (GMM). A simple example of a GMM ($p = 1$) with three clusters is shown in Fig. 11.4 with mixture probabilities of 0.5, 0.25 and 0.25, means of 1, 3, 6 and all the same standard deviation of 1. It is easy to see that more complicated distributions can be estimated by adding more groups/clusters.

Although GMMs have a lot more parameters to fit to the observations then a single Gaussian model they can be solved relatively efficiently via an iterative process called the **expectation-maximisation algorithm** (EM) which finds an optimal estimate[2] for the maximum likelihood function (See Sect. 8.2.1).

Consider observations, $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N} \in \mathbb{R}^p$. Without going into the details of the EM-algorithm, the process iterates between an expectation step (E-step), which

[2] Note this is more than likely a locally optimal rather than a globally optimal estimate.

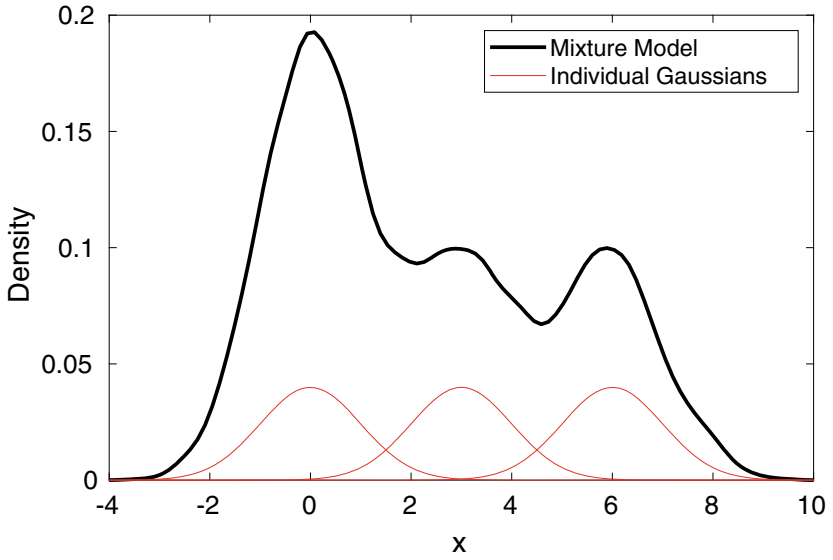


Fig. 11.4 Example of a three component Gaussian mixture model. Also shown are scaled version of the individual Gaussian components (in red) to show their positions and how they contribute to the overall distribution of the GMM

calculates the expectation of the log-likelihood function with current estimates of the parameters, and the maximisation (M-step) which updates the parameters which maximises the current expected log-likelihood function. For a GMM this translates to the following steps (calculated for each iteration):

1. Calculate the posterior probability τ_{ik} that each observations $\mathbf{x}_i$ belongs to each group $k = 1, \ldots, K$,
$$\tau_{ik} = \frac{\pi_k g_k(\mathbf{x}_i, \theta_k)}{\sum_{k=1}^{K} \pi_k g_k(\mathbf{x}_i, \theta_k)}. \tag{11.133}$$
This is the E-step.
2. Update the mixing probabilities $\pi_k^{new} = \sum_{i=1}^{N} \tau_{k,i}$, for each component $k = 1, \ldots, K$.
3. Update the mean for each component $k = 1, \ldots, K$,
$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} \tau_{k,i} \mathbf{x}_i}{\sum_{i=1}^{N} \tau_{k,i}}. \tag{11.134}$$
Note this is a weighted average, weighted based on the membership probabilities.

4. Update the Covariance matrix for each component $k = 1, \ldots, K$,

$$\boldsymbol{\Sigma}_k = \frac{\sum_{i=1}^{N} \tau_{k,i}(\mathbf{x}_i - \boldsymbol{\mu}_k)(\mathbf{x}_i - \boldsymbol{\mu}_k)^T}{\sum_{i=1}^{N} \tau_{k,i}}. \tag{11.135}$$

I.e. a weighted version of the sample covariance.

The number of groups, K, is a hyperparameter that must be chosen. Although this could be picked during cross-validation, a likelihood function framing means that information criteria (as introduced in Sect. 8.2.2) can also be used to find the most appropriate number of clusters. For different sized clusters calculate the BIC (or AIC). Plotting the BIC against the number of clusters can be used to find the point where increasing the number of clusters shows diminishing returns in terms of the drop in the BIC. This point is the "elbow" point of the plot (and hence why this heuristic is called the"elbow method") and indicates one choice for a suitable number of clusters. "Suitable" here is a relatively subjective term since there may be several other reasons why different numbers may be more appropriate or useful.

An example of the method is illustrated in Fig. 11.5. Here, the optimal number of clusters is around four since the tangential lines intersect around this value. Tangential lines are often used to make it easier to identify the elbow and hence the number of clusters.

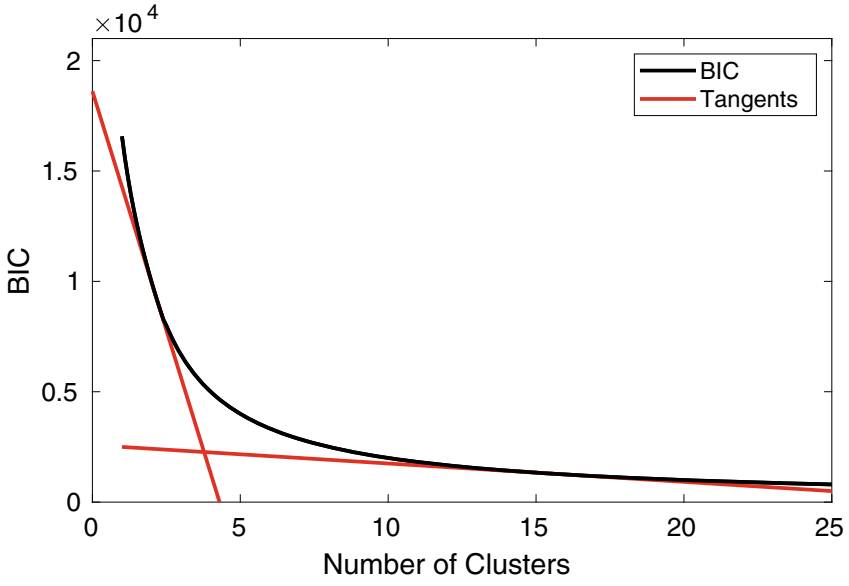


Fig. 11.5 Example of Bayesian information criteria for different numbers of clusters in a GMM. Also shown are tangents to the curves to demonstrate the 'elbow plot' method for determining the 'optimal' number of clusters

Fitting a GMM is an easy way to estimate a distribution *if* all the data comes from the same distribution. For a time series this means that the data used for training forms a stationary series. Unfortunately this is unlikely to be the case in general. Different hours of the day may have different distributions and the time series may be dependent on weather, time of year, or a whole host of other variables. Hence, even though the EM-algorithm allows for relatively quick training of the GMM, there may be insufficient data to train several mixture models accurately.

11.4 Quantile Regression and Estimation

The majority of models used for probabilistic load forecasting are nonparametric and are popular because they allow more flexibility in what distributions are being modelled. One of the simplest and most common ways to generate univariate probabilistic forecasts is quantile regression, the subject of this section. One of the advantages of the method is that it is a simple adaption of standard least squares regression.

Consider estimating the q quantiles (See Sect. 3.2 for introduction to quantiles) for the time steps $t = N + 1, N + 2, \ldots, N + k$. Popular choices are deciles (10-quantiles) or demi-deciles (20-quartiles) so that the distribution is split into 10 or 20 areas of equal probability respectively.

Consider the historical time series $L_1, L_2, \ldots, L_N$. Recall from Sect. 9.3, the aim in standard linear regression is to find the parameters $\boldsymbol{\beta}$ of some forecast model $f_t(\mathbf{Z}, \boldsymbol{\beta})$ by minimising the least squares difference with the observations. In mathematical terms this can be written

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta} \in \mathcal{B}} \left(\sum_{t=1}^{N} (L_t - f_t(\mathbf{Z}, \boldsymbol{\beta}))^2 \right). \tag{11.136}$$

Here $\mathcal{B}$ represents the set of feasible values the parameters can take, this is often the multi-dimensional real space $\mathbb{R}^p$, where p is the number of parameters for the chosen forecast model. Once the parameters are found the model can then be used to produce forecast values using new inputs.

For quantile regression the principle is identical, except now instead of minimising a least squares cost function, for each quantile $\tau \in \{1, 2, \ldots, q\}$ a set of parameters, $\hat{\boldsymbol{\beta}}_\tau$ must be found which minimise the difference between the model and the observations according to the quantile loss function, i.e.

$$\hat{\boldsymbol{\beta}}_\tau = \arg\min_{\boldsymbol{\beta} \in \mathcal{B}} \left(\min \sum_{t=1}^{N} c_\tau(L_t, f_t(\mathbf{Z}, \boldsymbol{\beta})) \right), \tag{11.137}$$

where the cost function $c_\tau(x, y)$ is defined by

$$c_\tau(x, y) = \begin{cases} \tau(x - y) & x \geq y \\ (1 - \tau)(y - x) & x < y \end{cases},$$

This is repeated for all quantiles. Recall this is the same pinball loss score introduced in Chap. 7. The process of quantile regression is slightly more complicated than for least squares regression as the cost function isn't differentiable. However the problem is easily reformulated as a linear programming problem and can be solved very quickly. Quantile regression is only applicable for models, $f_t(\mathbf{Z}, \beta)$, which are linear combination of the parameters. This still allows a lot of versatility in the types of relationships that can be modelled.

To illustrate the process consider a very simple example. Generate 400 points from a Gaussian distribution (See Sect. 3.1) with mean $\mu = 2$ and standard deviation $\sigma = 3$ to represent a time series of 400 points. Here the y-axis values are the random points and the order in time is simply the order in which they were sampled. The time series is shown in black in Fig. 11.6. Now consider a simple linear model of the form $f_t(\boldsymbol{\beta}) = at + b$, (i.e. the parameters $\boldsymbol{\beta} = (a, b)^T$) where there is no other inputs $\mathbf{Z}$ since there is no dependencies in this particular model. A quantile regression for this linear model is applied to the time series as in Eq. (11.137) for each deciles (or 10-quantiles). These are shown in Fig. 11.6 as the red dashed lines. Notice in theory, in the limit of increasing numbers of samples, the final quantiles should be flat horizontal lines, and should describe quantiles for a Gaussian distribution with mean

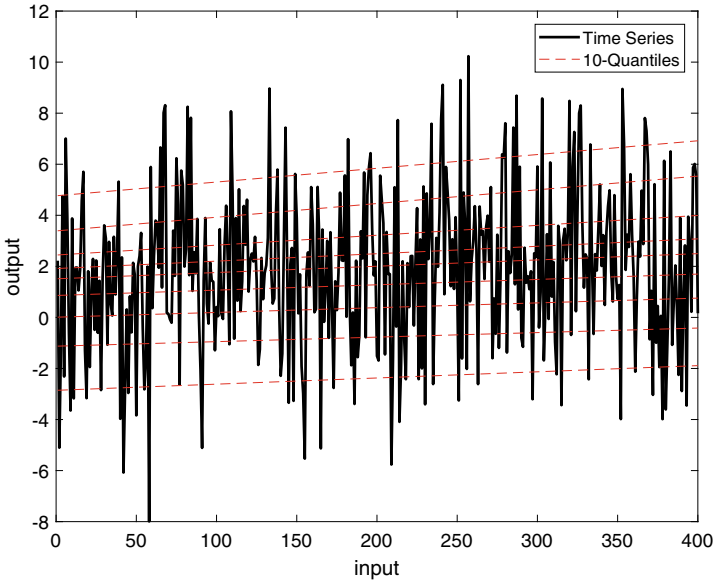


Fig. 11.6 Random time series (black) and the 10-quantiles generated from a quantile regression applied to the simple linear model $a + bt$

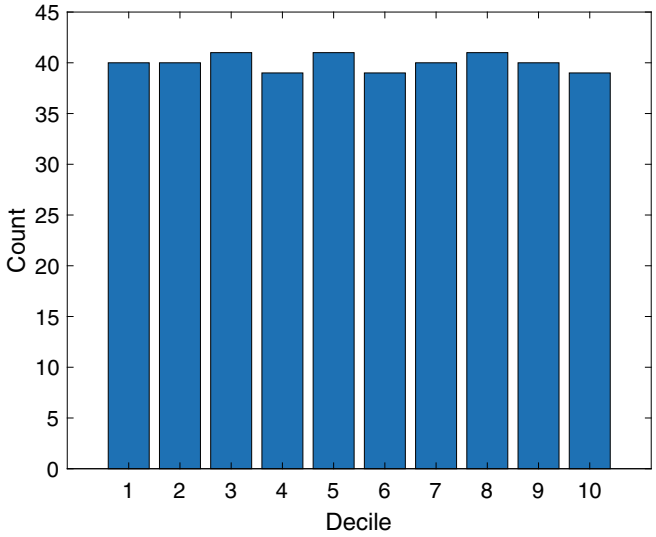


Fig. 11.7 Plot of the probability integral transform for the example in the text. This shows the count of observations in each decile as defined by the quantile regression on the linear model to the data

$\mu = 2$ and standard deviation $\sigma = 3$. However, in this case there is a slight gradient since there is only a relatively small amount of data and skews in the sampling can have a large effect on the model fit.

Recall in Chap. 7 that the Probability Integral Transform (PIT) can be used to assess the calibration of a probabilistic forecast. The quantile regression lines should split the data into equal probabilities of observations which would mean 40 (400/10) observations are expected between each of the consecutive deciles. This is shown to be the case in the PIT in Fig. 11.7. Notice in some quantiles there is actually 39 or 41 observations due to the relatively small number of samples. Having a uniform PIT on the training set should be expected when an appropriate model is chosen. The true assessment of the model is, as always, determined by evaluating it on an unseen test set rather than the training set. In addition, for a probabilistic forecast both calibration and sharpness are important properties and therefore the proper scoring functions introduced in Sect. 7.2 should be used to evaluate forecasts rather than the PIT alone.

Finally, it should be noted that since each quantile is trained independently, sometimes the quantiles may cross over with each other which would obviously be inconsistent since say the 5% quantile may end up higher than the 10% quantile. To prevent this, it is valid to reorder each quantile at each time step to ensure the p percentile is lower than the q percentile, when $p < q$.

11.5 Kernel Density Estimation Methods

As shown in Sect. 3.4 kernel density estimation (KDE) could be viewed as a smooth version of a histogram. However, instead of adding discrete counts in different buckets, a continuous distribution can be estimated by adding kernel functions at the positions where observations are made. Consider observations X_j, for $j = 1, \ldots, N$ of a random variable X, then the KDE for the probability density function is defined as

$$\hat{F}(X) = \frac{1}{Nh} \sum_{j=1}^{N} K\left(\frac{X - X_j}{h}\right), \tag{11.138}$$

where h is the bandwidth, a smoothing parameter for the estimate, and $K()$ is some kernel function. A popular example of the kernel function is the so-called Gaussian kernel defined as

$$K(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x\right). \tag{11.139}$$

The chosen kernel is often less important than the proper training of the bandwidth. Also note that the choice of kernel function has no relationship to the true distribution, i.e. a Gaussian kernel does not mean the data is distributed as a Gaussian. The importance of the bandwidth is illustrated in Fig. 11.8. The plot shows a comparison between the histogram of the 200 observations (left) versus the KDE of the same observations but for three different bandwidths (right). Selecting a bandwidth too small and the KDE will overfit the observations, too large and the KDE will underfit and have a higher bias (recall Sect. 8.1.2 on bias-variance tradeoff). Although

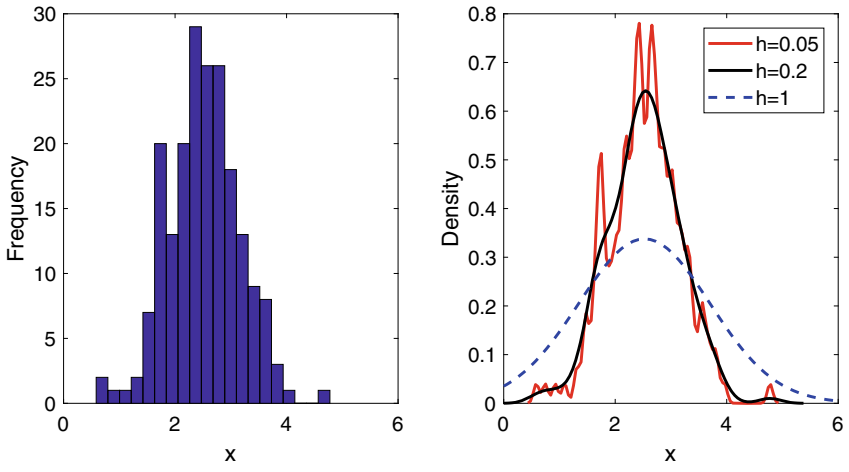


Fig. 11.8 Examples of estimating a distribution from 200 observations using **a** a histogram with 20 equally spaced bins and **b** a kernel density estimate with different bandwidths

there are rules of thumb for choosing the bandwidth, these are often based on strong assumptions of the underlying distribution (such as being Gaussian) and hence are too restrictive for the purposes of load forecasting. Instead the bandwidth can be chosen using cross validation by minimising the fit (often defined according to minimising a probabilistic scoring function such as the CRPS, see Chap. 7) between the estimate and the observations in the validation set. This could be through a search of the hyperparameter space e.g. grid search (Sect. 8.2.3). Although there is only one parameter for the simplest form of KDE, it can be a computationally expensive process.

The KDE can be easily adapted to probabilistic time series forecasts. Consider again historical observations L_t for $t = 1, \ldots, N$ for some random variable which are assumed to come from the same distribution with the aim being to generate k-step ahead density forecasts for each time step $t = N + 1, \ldots, N + k$ in the forecast horizon. In this simplified case the most basic kernel density estimate can be defined as

$$\hat{F}_i(L_{N+i}) = \frac{1}{Nh} \sum_{t=1}^{N} K\left(\frac{L - L_t}{h}\right), \tag{11.140}$$

for each time step in the horizon $i = 1, \ldots, k$, and bandwidth h. In other words, the distribution is assumed to be the same at each time step. This is clearly unrealistic for several reasons. For one, it is likely that older data is less relevant than more recent information. In addition, the KDE estimate in Eq. (11.140) is also independent of any other inputs, e.g. temperature or time of the day/week. To rectify these shortcomings modified versions of the simple KDE estimate are available.

To reduce the influence of older points a simple decay factor, λ with $0 < \lambda \leq 1$, can be introduced. This reduces the contribution of older data to the overall distribution function. One possible implementation is

$$\hat{F}_i(L_{N+i}) = \sum_{t=1}^{N} w_t K\left(\frac{L - L_t}{h}\right), \tag{11.141}$$

where the exponential decay weight $w_t = \frac{\lambda^{N-t}}{h \sum_{l=1}^{N} \lambda^{N-l}}$. In this case both the decay factor λ and the bandwidth must be optimised (again, often by cross-validation). Other weightings are of course possible, as will be seen with the conditional KDE form below. If more historical data is likely to be relevant then a slower decay, e.g linear, may be of interest. The only restriction is that the weights should sum to one to ensure the final function is still a well-defined probability distribution.

Another simple modification is to train only on specific historical points. For example, load data often has simple seasonalities (such as daily or weekly) of integer period s. In this case, the density can be estimated using

$$\hat{F}_i(L_{N+i}) = \frac{1}{N_s h} \sum_{t \in \mathcal{I}_i} K\left(\frac{L - L_t}{h}\right), \tag{11.142}$$

where N_s is the number of elements of the set $\mathcal{I}_i = \{j | N + i - j = sk$ for some integer $k\}$. $\mathcal{I}_i$ is every index which is a constant multiple of the period s, so in the case of hourly data with daily seasonality, to predict the $i = 2$ time period (e.g. 2AM if the forecast origin is at midnight) the historical data used to construct the KDE estimate would only use the data from the 2AM on each of the historical days. Recall the periodicities and seasonalities can be found using the methods presented in Chap. 5.

Another popular update to the simple KDE is conditional kernel density estimation. This estimates the distribution of the variable L_i, *conditional* on some dependent variables, say T, S. We can now utilise the pairs of independent-dependent observations as (T_t, S_t, L_t) and define the conditional distribution of $\hat{F}_i(L_i|T, S)$ as

$$\hat{F}_i(L_{N+i}|T, S) = \sum_{t=1}^{N} \frac{K((T_t - T)/h_T)K((S_t - S)/h_S)}{\sum_{l=1}^{n} K((T_l - T)/h_T)K((S_l - S)/h_S)} K\left(\frac{L - L_t}{h}\right), \tag{11.143}$$

were h_T, h_S are the bandwidths for the distributions representing T and S respectively and now must be found together with the dependent series bandwidth h. This is, as before, simply a weighted sum like in Eq. (11.141), but where the weights are kernel based functions of the dependent variables. As usual, popular choices of the independent variables are weather variables, but also period of the week. This can also be extended or simplified to take into account less or more variables respectively. However each bandwidth significantly increases the computational cost of training the models which can be impractical beyond two conditional independent variables.

Often to help accelerate the optimisation, the variables are normalised (e.g. to $[0, 1]$) in order to reduce the search space (see Sect. 6.1.3). The forecast can be rescaled after the training is complete. As mentioned in Sect. 3.4 there are options for the different kernels, and different ones can be tested, although often the choice has minimal impact on the accuracy of the forecasts [1].

The different modifications presented here can obviously be combined to create other models. For example the conditional kernel density form shown in Eq. (11.143) can be extended to include a decay factor like in Eq. (11.141) or restrictions can be applied on the inputs like in Eq. (11.142). As with many KDE methods, the drawback is that each modification often increases the training complexity and computational cost.

11.6 Ensemble Methods

This section introduces **ensemble forecasts**, by which we mean a set of point forecasts from the same forecast origin, estimating each time step with the same forecast horizon (of length h time steps). The point forecasts are samples of equal probability from a h-dimensional multivariate distribution representing the joint distribution over the forecast horizon (see Sect. 3.3 for more on joint distributions). In other words,

each ensemble represents an equally likely load trajectory. The methods described in this section produce these ensembles without needing to produce the full joint distribution.

11.6.1 Residual Bootstrap Ensembles (Homoscedasticity)

This section describes a method for generating ensemble forecast which can be viewed as realisations from a full multivariate probabilistic forecast in the case where the time series is assumed to have fixed variance. To begin consider a 1-step ahead point forecast model, for example, this could be the exponential smoothing model in Sect. 9.2 or the ARIMA models in Sect. 9.4. Denote this as $f(L_t|\mathbf{Z}, \boldsymbol{\beta})$ which may use previous historical data as well as any explanatory inputs (all described by the set of variables $\mathbf{Z}$) to create an estimate, $\hat{L}_{t+1|t}$, for the true value, L_{t+1}, at $t+1$. The $\boldsymbol{\beta}$ are the parameters for the model. The next time step ahead can be forecast by iteratively applying the model and including the forecast from the previous time step as a pseudo-historical input to the model. Hence for the next time step

$$\hat{L}_{t+2|t} = f(\hat{L}_{t+1|t}|L_t, \mathbf{Z}, \beta) = f(f(L_t|\mathbf{Z}, \beta)|L_t, \mathbf{Z}, \beta). \tag{11.144}$$

The process can obviously be repeated to produce k-step ahead forecasts. Now, recall that there is an error process describing the difference between the observations and the 1-step ahead forecasts described by the residual $\epsilon_{t+1} = L_{t+1} - \hat{L}_{t+1|t}$. By including the small deviations, described by the residual series, into the forecast, different trajectories can be created which represent different, but equally likely outcomes.

To describe the algorithm in more detail, consider a k-step ahead forecast generated from a one step ahead model, e.g. $\hat{L}_{t+1} = f(L_t|\mathbf{Z}, \boldsymbol{\beta})$. Assume the forecast origin is at time step $t = N$. Hence the aim is to produce ensemble forecasts which cover the period $N+1, \ldots, N+k$. The residual series, $\epsilon_t = L_{t+1} - \hat{L}_{t+1}$ (which is calculated for the entire training set), is assumed to have a fixed variance (the series is said to have **homoscedasticity**) and are uncorrelated with each other. Using this residual series the process of generating a new ensemble for a k-step ahead forecast, using a 1-step ahead forecast model f, is relatively simple. For each ensemble, b, the procedure is as follows:

1. Randomly sample with replacement (this is called a bootstrap sample) a residual, $\hat{e}_1^{(b)}$, from the set of all residuals, $\{\epsilon_1, \epsilon_2, \ldots, \epsilon_N\}$.
2. Add this residual to the current 1-step ahead forecast value $\tilde{L}_{N+1|N}$ to produce a new value $\hat{L}_{N+1|N}^{(b)} = \tilde{L}_{N+1|N} + \hat{e}_1^{(b)}$.
3. Include $\hat{L}_{N+1|N}^{(b)}$ in the forecast model to generate an estimate for the next time step, $\tilde{L}_{N+2|N+1}^{(b)} = f(\hat{L}_{N+1|N}^{(b)}|L_N, \mathbf{Z}, \beta)$.

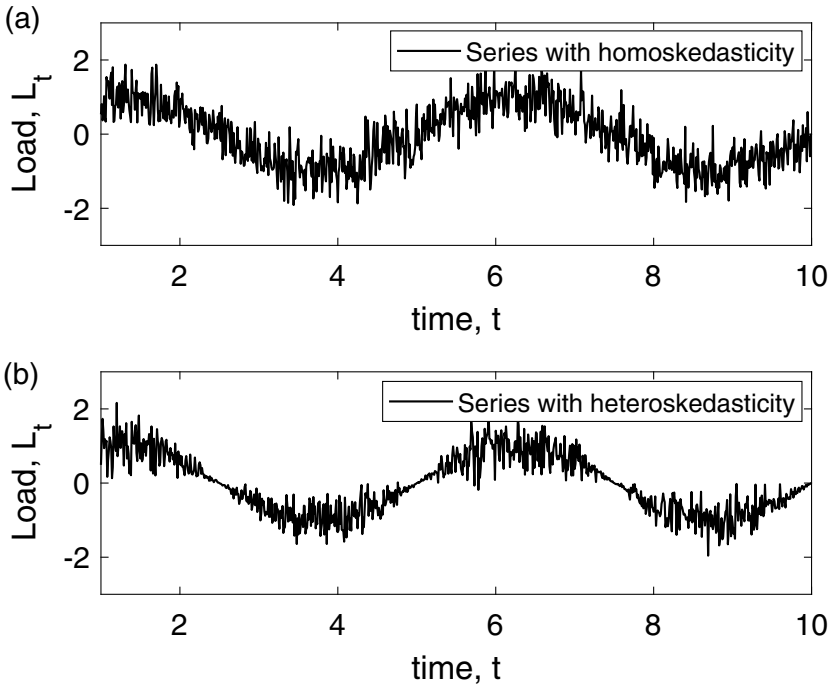


Fig. 11.9 Example of a simple periodic time series with homoscedasticity (top) and heteroskedasticity (bottom)

4. Update this value using another bootstrap sample from the residual series to give $\hat{L}^{(b)}_{N+2|N+1} = \tilde{L}^{(b)}_{N+2|N+1} + \hat{e}^{(b)}_2$.
5. Continue this procedure until the kth step is reached.
6. The final series, $\hat{L}^{(b)}_{N+1|N}, \hat{L}^{(b)}_{N+2|N+1}, \ldots, \hat{L}^{(b)}_{N+k|N+k-1}$ is the bth bootstrap ensemble.

This is also known as a **residual bootstrap forecast**. The process can be repeated to produce as many ensembles as desired. The more ensembles generated, the more load diversity is captured. Generating more samples increases the computational cost, but since each ensemble is independent of the others they can be generated in parallel. Notice that this method strongly assumes that the 1-step ahead errors in the future will be similar to the past 1-step ahead errors. An example of a simple periodic series with homoscedasticity is shown in Fig. 11.9a.

If instead of sampling from the actual residuals you sample from an assumed or fitted distribution then the method can be referred to as a Monte Carlo forecast. For example, it is often assumed that residuals are Gaussian distributed with zero mean and therefore instead of sampling from the set of residuals, the values can be sampled from a Gaussian distribution trained on the residuals. An example of an ensemble forecast generated from the Monte Carlo simulations, for 100 ensembles is shown

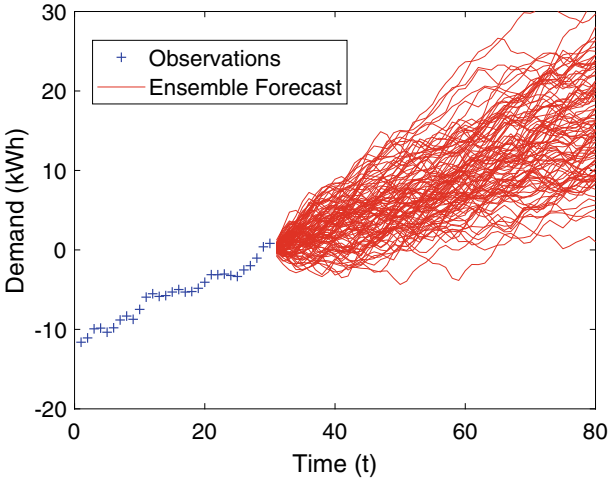


Fig. 11.10 Example of an Monte Carlo derived ensemble 50-step ahead forecast with 100 ensembles for a simple ARIMA model

in Fig. 11.10 for a simple ARIMA(4, 1, 1) model. Notice that the errors get wider (have larger variation) with forecast horizon length. This is due to the accumulation of the errors from one step to the next. This intuitively makes sense as the uncertainty should increase the further ahead the prediction.

Notice that the forecasts at each time step can be used to estimate a univariate estimate. This can be done by either fitting quantiles or a density estimate to the collection of ensemble points at each time step.

11.6.2 Residual Bootstrap Ensembles (Heteroskedasticity)

An advantage of the bootstrap method described in Sect. 11.6.1, is that a multivariate forecast can be generated with minimal computational cost since only the original point forecast model needs to be trained. Further, if the model contains autoregressive features (as ARIMA and exponential smoothing do) then the ensembles also retain the interdependencies of the time series. A drawback to the method is the strong assumption of homoscedasticity for the series of residuals. In fact, it is likely that periods of high demand will also have larger variability. A time series where the variance changes in time is said to have **heteroskedasticity**.

An example of a simple periodic series with heteroskedasticity is shown in Fig. 11.9b in which the largest variation in the demand coincides with the largest amplitude of the periods. When time series are heteroscedastic the variability can be incorporated using so-called GARCH-type models which can extend the bootstrapping method described in Sect. 11.6.1. An outline of these methods are given here,

but since they are relatively complicated, the details are ommited. The interested reader is referred to some further reading given in Appendix D.

Consider a model for the load time series with some mean process (e.g. any of the point forecasts presented in Chaps. 9 and 10) and as before let ϵ_t be the 1-step ahead error/residual terms. Now assume that the standard deviation σ_t also varies with time t. It is helpful to write the residual in the form

$$\epsilon_t = \sigma_t Z_t, \tag{11.145}$$

where $\sigma_t > 0$ is a time-dependent conditional standard deviation, and the $(Z_t)_{t_\epsilon \mathbb{Z}}$ is a random variable which is stationary, independent in time and has the conditions $\mathbb{E}(Z_t) = 0$ with $\mathbb{V}ar(Z_t) = 1$. Splitting the data into the standard deviation this way helps to simplify the actual variation into the magnitude of variation, represented by the standard deviation, and the random component which is now stationary due to the scaling. Once the components of Eq. (11.145) are found it is easy to apply an adapted form of the bootstrap method by taking random samples from the distribution of Z_t, and then rescale with the modelled standard deviation σ_{N+k} at the time step being forecast.

To do this, first a model must be chosen for the standard deviation. In economic models standard choices are ARCH and GARCH models. The ARCH and GARCH models are essentially variance counterparts to the AR and ARMA models of the point forecasts introduced in Sect. 9.4. The GARCH(p, q) model is of the form

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2, \tag{11.146}$$

where the q is the lag residual terms (called the ARCH term) and p is the lag variance terms (this is called the GARCH term). The ARCH model only has the q lags with the residual terms and no variance terms. These are often coupled with the corresponding AR, or ARIMA model for the mean process and are solved with ordinary least squares or maximum likelihood methods.

ARCH and GARCH are specific forms of the variance which are often suitable for financial time series applications. In fact standard deviation can be modelled much more generally and these will be referred to as GARCH-type methods. In load forecasting, the variation in the demand is often larger for time periods when the demand is typically higher (however, of course, for each new time series the patterns in the variance should be analysed before choosing a model). For this reason it is often suitable to choose a standard deviation model which is similar to the point forecast model chosen.

For ARCH and GARCH problems the procedures are relatively well established and hence there are many packages for automatically selecting the order and coefficients for the model. In the more general case a simple procedure can be followed which is adapted from [2]:

1. Since the components of the residuals as given in Eq. (11.145) are not known, instead consider the absolute or squared residuals $|\epsilon_t|$ or ϵ_t^2. The former will be the focus as it better captures the heavy tails of energy demand. However, the same procedure is applicable for both forms.
2. Since $\mathbb{E}(|\epsilon_t|) = \sigma_t \mathbb{E}(|Z_t|))$, fitting a model to $|\epsilon_t|$ is equivalent to fitting a model to a scaled version of the standard deviation, $C\sigma_t$, for some constant $C > 0$ since Z_t is a stationary variable (and hence has constant expectation). The fit is usually achieved using ordinary least squares (Sect. 8.2).
3. The constant C must now be estimated by considering the normalised residuals, $\epsilon_t/|\epsilon_t| = \epsilon_t/C\sigma_t = Z_t/C$, i.e. a scaled version of Z_t. Since Z_t has variance equal to one, the scaling, C, can be be estimated by calculating the sample standard deviation, α, of the normalised residuals which tells us that $C = 1/\sqrt{\alpha}$.

For this method notice that there is no assumption on the underlying distribution (only on its variance). An updated version of the bootstrap forecast in Sect. 11.6.1 now updates the 1-step ahead forecast by adding a sample from the distribution of Z (sample from the empirical distribution for Z formed from the standardized residuals ϵ_t/σ_t) which has then been scaled by the σ_t model at the current time. Note it is assumed that the errors ϵ_t and standardised residuals are uncorrelated in time to allow them to be randomly sampled for the bootstrap. As usual residuals should be checked (in this case the standardised residuals) to ensure that they satisfy the assumptions about correlation, fixed variance and have zero mean. If these assumptions do not hold then further updates can be applied (as in Sect. 7.5). A specific example of the above GARCH-type model will be given in the LV case study in Sect. 14.2.

11.7 Copula Models for Multivariate Forecasts

This section introduces **Copulas**, another popular method for generating multivariate probabilistic forecasts, widely used in quantitative finance, but now used extensively in energy forecasts. Here only the basics will be described. Suggested further reading can be found in Appendix D.

A copula is simply a function, C, from an N-dimensional unit box to a 1-dimensional unit box, i.e. $C : [0, 1]^N \longrightarrow [0, 1]$, and describes a cumulative distribution function on variables $U_1, \ldots, U_N$ where each variable U_i is uniformly distribution over $[0, 1]$, i.e. $C(u_1, u_2, \ldots, u_N) = P(U_1 \leq u_1, U_2 \leq u_2, \ldots, U_N \leq u_N)$. In other words a copula models the joint distribution on variables $U_1, \ldots, U_N$ with uniform marginal distributions (See Sect. 3.3 for more details on joint and marginal distributions). A copula is focused on the correlation/inter-dependence structure of the variables. An advantage of these methods is the wide range of different possible copula functions which can be used to model the inter-dependence.

The power of copulas lie in the fact that, due to *Sklar's theorem*, any multivariate distributions can be modelled using only the marginal distributions and a copula. Consider a random variable $\mathbf{X} = (X_1, X_2, \ldots, X_N)^T \in \mathbb{R}^N$ with joint distribution

$F_{X_1,\ldots,X_N}(x_1,\ldots,x_N)$ and marginal CDFs denoted by $F_1(x_1),\ldots,F_N(x_N)$, then by Sklar's theorem, there exists a copula C such that

$$F_{X_1,\ldots,X_N}(x_1,\ldots,x_N) = C(F_1(x_1),\ldots,F_n(x_N)). \tag{11.147}$$

Note that for any random variable X with CDF F, is uniformly distributed when transformed by its CDF, i.e. $F_X(X)$ is uniformly distributed. Recall this is just the probability integral transform (PIT) described in Chap. 7.

The process in Eq. (11.147) is reversible which means once a copula model has been trained on observations then multivariate samples can be easily produced with the relevant dependency structure. First samples $u_1, u_2, \ldots, u_N$ are generated from the copula distribution and then each variable is transformed into the original random variable space using the inverse CDF of the marginal $F_i^{-1}(u_i)$ for each corresponding component of the sample point.

Suppose that $\mathbf{X} = (X_1, X_2, \ldots, X_N)^T \in \mathbb{R}^N$ has a multivariate Gaussian distribution, then the corresponding copula is defined purely by the correlation matrix since this explains the entire dependency structure. This also means that each correlation matrix defines a specific Gaussian copula. Of course just because a multivariate random variable has a Gaussian copula doesn't mean that it follows a multivariate Gaussian distribution since the marginals need not be Gaussian. If the correlation matrix is the identity then the copula is called the **independence copula** defined by

$$C_0(u_1, u_2, \ldots, u_N) = u_1 \ldots u_N, \tag{11.148}$$

where each component is independent of the other components. In general, there is no simple analytic form for a **Gaussian copula** but it can be expressed as

$$C_R^{Gauss}(u_1, u_2, \ldots, u_N) = \Psi_R(\Psi^{-1}(u_1), \ldots, \Psi^{-1}(u_N)), \tag{11.149}$$

where Ψ is the univariate CDF of the standard Gaussian (mean zero and unit standard deviation) and Ψ_R is the multivariate Gaussian with zero mean and correlation matrix R.

Another family of popular copula's are the **Archimedean copulas**, which have explicit formula's and can represent multivariate distributions using only one parameter, θ. They have the general form

$$C(u_1, \ldots, u_N; \theta) = \psi^{-1}\left(\psi(u_1; \theta) + \cdots + \psi(u_N; \theta); \theta\right) \tag{11.150}$$

where $\psi : [0, 1] \times \Theta \to [0, \infty)$ is a continuous, strictly decreasing, convex function such that $\psi(1; \theta) = 0$. For example one popular Archimedean copula is the Gumbel Copula, which is defined by

$$C_{Gum}(u_1, u_2, \ldots, u_N|\theta) = \exp\left[-\left((-\log(u_1))^\theta + \cdots + (-\log(u_N))^\theta\right)^{1/\theta}\right]. \tag{11.151}$$

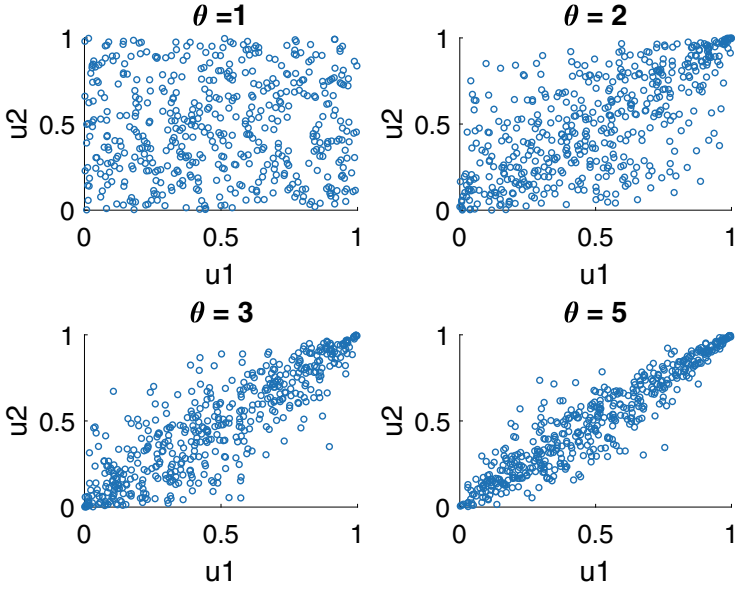


Fig. 11.11 Example of samples from Gumbel copulas with different values of θ

Notice that this becomes the independence copula when $\theta = 1$. Examples of samples from a bivariate Gumbel copula with different values of θ are shown in Fig. 11.11. Note that the Gumbel can never represent negative correlation.

It is clear that different copula's are useful for different dependency structures and that not all copulas are useful for all types of data. For example, Gumbel copula's shouldn't be used with data with negative correlations. How to choose and fit a copula will be briefly considered later in this section.

The value of the Pearson's correlation coefficient depends on the marginals and the copula, which means random variables will have different Pearson values when transformed using the marginal CDFs. A more convenient measure for the correlation structure for a copula are Rank correlation coefficients which only depend on the *rank* of the data (see Sect. 3.5). Since the rank of data is unchanged by the application of a monotonically increasing function, it means the rank correlation coefficients won't change when the marginal CDFs are applied. An example of a rank correlation coefficient was given in Sect. 3.5: the Spearman's rank correlation coefficient.

To better understand how copula's work and how they can be used to generate new samples, consider a simple bivariate distribution for the random variables $(X1, X2)$, whose dependency structure is described by a Gaussian copula with covariance

$$R = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

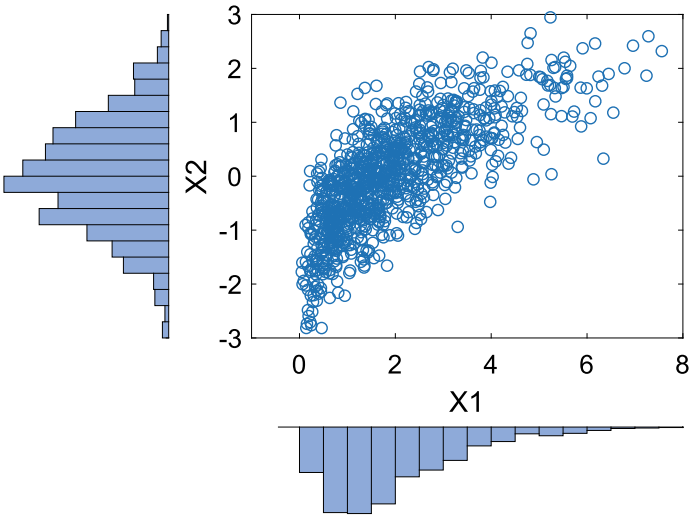


Fig. 11.12 Observations from a bivariate distribution with Gaussian Copula, with Gamma ($X1$) and Gaussian ($X2$) marginal distributions (shown as the histograms)

i.e. it has linear Pearson correlation $\rho = 0.8$. Also suppose that the marginal of the first variable, $X1$, has a distribution described by a Gamma distribution, where

$$Gamma(X, \alpha, \beta) = \frac{1}{\beta^{\alpha}\Gamma(\alpha)} \int_0^X t^{\alpha-1} e^{-t/\beta} dt \tag{11.152}$$

where $\alpha = 2$ is the shape parameter (determines the shape of the distribution), and $\beta = 1$ is the scale parameter (determines how spread the distribution is). $\Gamma(.)$ is the so-called gamma function. The second variable, $X2$, is described by the standard Gaussian distribution (mean zero and unit standard deviation). An example of 1000 observations from this distribution is shown in Fig. 11.12. On the horizontal and vertical of the plots are marginal histograms of each variable, which shows the Gamma and Gaussian distributions respectively.

The distribution after applying the CDF of each marginal to its respective variable is shown in Fig. 11.13. The marginals are now described by uniform distributions, as expected. Notice in this example the sample pearson correlation between $X1$ and $X2$ is $\rho = 0.767$ but between $U1$ and $U2$ is $\rho = 0.785$ and is not preserved by the transformation (although they are close in this example). In contrast the Spearman's correlation between $X1$ and $X2$, and between $U1$ and $U2$ are both $r = 0.787$ as expected. In practice the objective would be to fit a copula to this transformed data. In this case assume it is known that the copula is a Gaussian and therefore the aim is to find the Pearson correlation coefficient ρ. In fact in the [3] bivariate case the Spearman's coefficient r is related to the linear correlation via

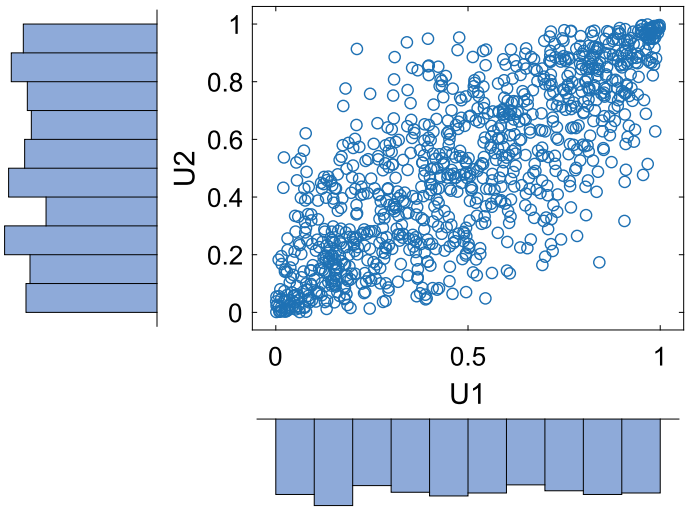


Fig. 11.13 Distribution of transformed observations where the marginals of their respective components have been applied. The marginals are now uniform as shown by the histograms

$$\rho = 2 \sin\left(r \frac{\pi}{6}\right), \tag{11.153}$$

and hence the invariant value of the Spearman's correlation can be used to estimate the Pearson correlation and gives $\rho = 0.801$ (note it isn't exactly the 0.8 used to generate the data due to numerical deviations in the sample, the larger the sample the closer we would expect the sample value to be to the original parameter).

Given the copula, new samples can be generated and then transformed to the original space (with the same linear correlation) by using the inverse CDF of the marginals. An example of 1000 new points generated using the copula (and transformed using the inverse CDFs) is shown in Fig. 11.14. Notice how the final distribution successfully resembles the original distribution in Fig. 11.12.

A similar process can be used to generate ensemble demand forecasts. In this case, consider a demand time series $L_1, L_2, \ldots$, where the aim is to generate a day-ahead multivariate forecast with, say, forecast origin $t = N$. Further for simplicity, suppose the data is hourly and hence a 24-step ahead forecast is being considered. The aim here will be to generate a multivariate distribution for the day, $F_{N+1,\ldots,N+24}(L_{N+1}, \ldots, L_{N+24})$, and hence model the inter-dependencies between different times of the day. It is assumed that CDFs for the marginals at different times of the day are already known, i.e. $F_1(L_{N+1}), \ldots, F_{24}(L_{N+24})$ are known. These could be estimated, e.g. by the univariate probabilistic models described earlier in this chapter. In this case a copula can be used to model the intra-day dependency structure by training on the daily profiles transformed by the marginals.

There is a range of different copula models, only some of which are mentioned above. The questions remain on how to train and choose the copulas on the data.

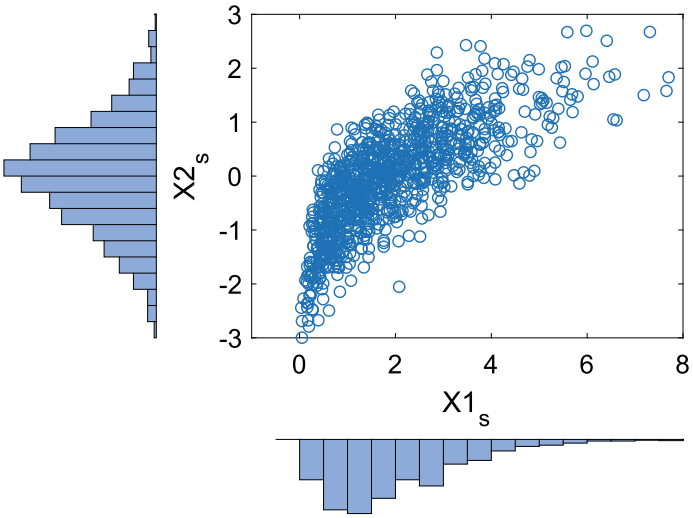


Fig. 11.14 Samples from the copula model fitted to the observations

In theory, for parametric models for the copulas and marginals, maximum likelihood estimation (see Sect. 8.2) could be used to fit a copula, however this can be complicated for high dimensional problems as there are lots of parameters to train. Instead the models can be estimated using a two step process called the *Pseudo-Maximum Likelihood*, where first the marginals are estimated and a reduced form of the maximum likelihood, given by

$$\sum_{k=1}^{n} \log \left[c\{\hat{F}_1(X_{1,k}), \ldots, \hat{F}_N(X_{N,k}) | \boldsymbol{\theta}_C\}) \right], \tag{11.154}$$

is maximised. Here c is the copula density corresponding to the Copula CDF, C, and $\boldsymbol{\theta}_C$ represents the parameters for the copula model. The maximisation above can still be complicated especially for higher dimensional problems, depending on the copula model considered. One approach has already been suggested for the simple Gaussian copula example given above. The correlation matrix can be estimated by using the Spearman's correlation coefficients for each pair of variables and this can either be used as final correlation matrix or as an initial guess in the pseudo-maximum likelihood optimisation in Eq. (11.154).

Choosing the correct copula's depends on many factors and a detailed investigation is beyond the scope of this book. Further reading is suggested in Appendix D. In summary, the choice depends on the type of correlation being modelled as well as the dependencies within the tails/extremes of the distribution. One possible approach for choosing an appropriate copula(s) model can be based on comparison on a validation set as described in Sect. 8.1.3.

11.8 Questions

For the questions which require using real demand data, try using some of the data as listed in Appendix D.4. Preferably choose data with at least a year of hourly or half hourly data. In all the cases using this data, split into training, validation and testing in a 60, 20, 20% split (Sect. 8.1.3).

1. Sample 20 points from a 5-dimensional Gaussian distribution. Make sure that some of the variables are more correlated than others by manipulating the correlation between them in the covariance matrix. You can fix the variance of all the variables to one to make the model simpler. Now consider that each dimension of the Gaussian is a different time step in a time series of five points. Plot each sample to create samples like in Fig. 11.2. What can you see between the variables which are highly correlated? What if you change the variance for different variables, how does this change the ensemble plot?
2. Generate a quantile regression. Take your linear forecast model you generated in Sect. 9.7. Now fit to the training data a quantile regression for percentiles of $10, 20, 30, \ldots, 90$ using inbuilt packages such as quantreg in R.[3] Apply to the test set, and count how many values lie between each set of quantiles. Plot the probability integral transform. What shape is it? Is there a bias in the model? Is it under or over dispersed? What adjustments to the quantiles could help produce a uniform PIT?
3. To demand data with daily or weekly periodicity fit a kernel density estimate for each time step from each period in the seasonal cycle. For example, if the data is half hourly with daily seasonality then train 48 models for each half hour of the day. Fit the model by performing a grid search for the bandwidth. With the final model, apply it to the test set. Generate quantiles for the estimate, and thus calculate the PIT for the same percentiles as the previous question. Is the PIT uniform, overdispersed or underdispersed?

References

1. J. Jeon, J.W. Taylor, Using conditional kernel density estimation for wind power density forecasting. J. Amer. Stat. Ass. **107**(497), 66–79 (2012)
2. S. Haben, G. Giasemidis, F. Ziel, S. Arora, Short term load forecasting and the effect of temperature at the low voltage level. Int. J. Forecast. **35**, 1469–1484 (2019)
3. T.C. Headrick, A note on the relationship between the pearson product-moment and the spearman rank-based coefficients of correlation. Open J. Stat. **6**, 1025–1027 (2016)

[3] https://cran.r-project.org/web/packages/quantreg/index.html.

Chapter 12
Load Forecast Process

The previous chapters have discussed all of the aspects of developing and testing a short term load forecast. This includes

- How to analyse data to identify patterns and relationships.
- How to identify, extract and select features to include in the forecast model.
- How to split data for training, validation and testing.
- The different types of forecasts and their features.
- Popular statistical and machine learning models for point and probabilistic forecasting.
- How to select error measures and scores to assess your forecast accuracy.

However, what are the steps required for actually producing a forecast? In Sect. 12.1 the general steps in developing a forecast experiment are discussed, and in Sect. 12.2 some of the criteria for choosing among the plethora of forecast models introduced in Chaps. 9–11 are given.

12.1 Core-Steps for Forecast Development

The following are the main steps in developing a forecast model for your chosen application. They are written in the approximate order they should be applied but many steps can be repeated or ignored depending on the circumstances. For example, sometimes further data analysis may be required if new data becomes available, or the initial model reveals other relationships to be checked. The process can be repeated when looking to refine the forecasts but ideally this should only be tested on data that has not been seen or used previously to ensure no bias or cheating (even unconscious) is included in the experiment.

1. **Understand the problem**: Presumably you are creating the forecasts for a specific application or purpose (such as those in Chap. 15). In which case it is worth fully understanding what the objectives are and what would be a good measure

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_12

of success. Without solid aims it is easy to lose focus, and produce sub optimal results. Often the objectives will be determined by business objectives, in which case it is essential to translate these aims into a well-defined problem where you will know that a suitable result has been achieved. Proper development of the problem framing early on can influence or determine the choice of error measure, model, or even the data analysis later in the modelling process.

2. **Access the Core Data**: Although it's likely the data is available in some format (otherwise the project itself may not be possible in the first place), it is important to perform some form of data audit early in the study to ensure you have the minimum required data to tackle the problem. Your understanding of what data may be needed will likely change as you investigate the data further but a quick check of what and how much data is available is essential to prevent wasting time on an impossible task and allows time to collect data you discover may be vitally needed.
3. **Initial Data Checks**: Now you have the data, a deeper check of the quality and usability is required. You can start to understand some of the relationships and features of the data at this point but the main objective is to understand the amount of cleaning and preprocessing that is required, and whether you have sufficient data to continue. This is a good point to check for missing data, anomalous values, outliers etc. (Sect. 6.1).
4. **Data Splitting**: Once you understand the complexity, quality and quantity of the data, you can determine the split of the dataset into Training, Validation and Testing data (Sect. 8.1.3). This may be determined by the length of any seasonalities, or the number of parameters (more parameters need more training data). This choice is important to determine the right bias-variance trade-off (Sect. 8.1.2) so your models don't over and under fit the data. The split also prevents the researcher utilising information which would not be available at the time of producing the prediction (also known as *data leakage*). This would be unrealistic and create unscientific results since in real-world scenarios you would be testing on unseen data.
5. **Data Cleaning**: Here anomalous values and outliers should be removed or replaced with other values. If there is sufficient data then it may be better to remove the values so as not to introduce biases from any imputation methods used (Sect. 6.1.2). If the missing or removed values are not imputed you must make sure to adjust your model so it can handle missing cases or you should reduce the data so no missing instances are included (although this reduces the amount of data available for training/testing). If time permits it may be worth including tests of models with and without cleaning to see if it affects the model performance.
6. **Visualisation and Data Analysis**: Next is the deep dive into the data, arguably the most important aspect of developing forecasting models. This will help you understand the relationships and patterns in the data, the types of relationships, the important exogenous variables, perhaps even the models which may be most suitable. This step will likely iterate with the data cleaning step since you won't necessarily know what an outlier is without doing some preliminary analysis, and

similarly you can't complete your analysis until the data is cleaned. Visualisation techniques and feature identification methods are discussed in Sect. 6.2.

7. **Further Pre-processing**: Given the data analysis further preprocessing may be required. For example, perhaps the data must be normalised or scaled to help with training the data. Alternatively a transformation may have to be applied to change the data distribution to one which is more appropriate for the model being used, e.g. to be Normally distributed so that a linear regression model can be used (Sect. 6.1.3).
8. **Initial Model and Error Choices**: Given the analysis, choose and derive some models. You can always add further models later on, in particular once you've finally tested the models on the test set and discovered limitations or possible improvements. It is also important at this stage to choose an appropriate benchmark (or benchmarks) with which to compare your model (Sect. 8.1.1). Further criteria for choosing the initial models are given in Sect. 12.2. In addition, at this stage a suitable error measure must be chosen. When tackling a specific application, the performance within the actual application is the true test of the usefulness/effectiveness of the forecast model. However it may not be practical to test all models (and their adaptions) within the application due to high computational costs. In these cases a computationally cheaper error measure which correlates with the application performance is more appropriate. Different error measures for point and probabilistic forecasts are discussed in Chap. 7.
9. **Training and Model Selection**: Using the data split (determined in a previous step) train the data on the training dataset (Sect. 8.2). Utilise the validation set to compare accuracy of the models and determine the optimal hyperparameters within each family of models (Sect. 8.2.3), including any weights for regularisation methods (Sects. 8.2.4 and 8.2.5). For some models such as ARIMA (Sect. 9.4) a model can be chosen without using the validation as a hold-out set. In these cases you can use Information Criteria (Sect. 8.2.2) to choose the hyperparameters on the combined training and validation set. These final chosen models (including the benchmarks) are taken through to the testing phase.
   Note if you are considering rolling forecasts (Sect. 5.2) or forecasts that are updated at regular intervals then you will have to apply a rolling window over the validation set.
10. **Apply the models to the test set**: Retrain the models on the combined validation and training datasets (This should improve the fit of the data, see Data augmentation in Sect. 8.2.5). Again, if you are considering rolling forecasts (Sect. 5.2) or forecasts that are updated at regular intervals then you will have to apply a rolling window over the test set.
11. **Evaluation**: Now you must evaluate the final models in a variety of ways to understand where they under (or over) perform. The most basic assessment is to ask which model performs best according to the chosen metric or measure? Are there different periods of the day or week which have different accuracy for different models? How does the accuracy change with horizon? How do the models rank compared to the chosen benchmark(s)? What were the common

features of the models that perform best? Or worst? Finally, consider the residuals and their distributions. Are there any remaining features or biases? (Sect. 7.5).

12. **Model Corrections and Extensions**: Forecast corrections (Sect. 7.5) should be considered (with the correction ideally trained on the validation set, not the test set) where possible. However, a simple improvement is to combine the models that you've already produced (Sect. 13.1). This has been shown to be a very effective way to utilise the diversity across the models to improve the overall accuracy.
13. **Evaluation with the Application**: If the forecasts are used within an application then a true test of their usefulness is within the application or an *in silico* model for the application (for example for controlling a storage device, Sect. 15.1) rather than the error measure. If an error measure has been appropriately chosen the application performance will correlate with scores for the accuracy. This should be confirmed and if there is inconsistencies they should be further investigated.
14. **Next Steps**: Looking at your analysis of the results there could be new inputs which could be appropriate (e.g. different weather variables), or different ways of using the same inputs (e.g. more lagged values from the temperature time series or combining different inputs to create a new variable). The process should now be repeated (ideally on new, unseen data) to test further updates or adaptions. This process can be repeated until a sufficient forecasting accuracy or application performance has been achieved.

The steps in producing a forecast are outlined in the diagram in Fig. 12.1. The procedure can be seen in terms of two components. The data collection and analysis is the first part which describe how the data is mined, analysed and wrangled to get it ready for testing. It also is used to understand the features to use in the models. The second stage is the model selection, training and testing.

12.2 Which Forecast Model to Choose?

In the last few chapters a wide variety of methods were introduced from the different types listed in Sect. 5.3. There are point and probabilistic forecasts, those suited more to direct forecasts than rolling, and a mix of statistical and machine learning methods. There are no hard and fast rules to determine which are the most appropriate models to use, and the choices will depend on the application and context. However, there are some general principles which can be followed to help narrow down the model choice:

1. **Computational costs**: short term load forecasts require at least daily updating. For this reasons, models which are computationally quick to train, but are less accurate, may be preferable to more accurate but computationally expensive models. If the model is taking too long to run then it may be worth trying one which uses less processing power, memory, or time to run.

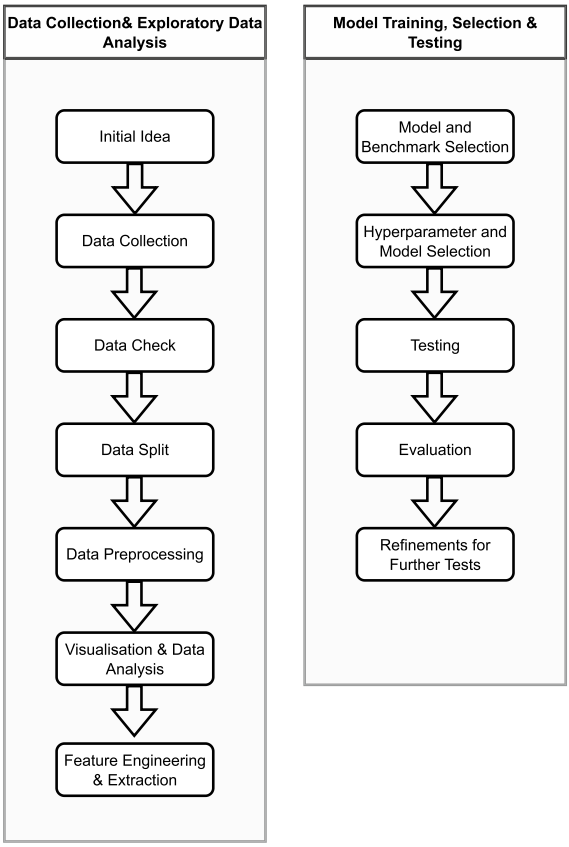


Fig. 12.1 Forecasting procedure as outlined in this section

2. The **type of relationship** believed to exist between the predictor and the dependent variable: if the relationships are not clear, or appear to be relatively complex, then machine learning techniques may be preferable (see Chap. 10). Are the relationships between explanatory variables linear or nonlinear? If linear then simple statistical models such as linear regression and ARIMA may be suitable (Sects. 9.3 and 9.4). If nonlinear then perhaps GAMs (Sect. 9.6) or neural network models (Sect. 10.4) should be considered.
3. The **type of features** (Sect. 6.2): For example, if lagged components of the data are important then an autoregressive model may be the most appropriate (see Sect. 9.4). If only exogenous variables like weather or econometric variables are important, maybe a cross-sectional forecast model is more appropriate than a time series one. In these cases, tree-based models (Sect. 10.3) and simple feed-forward neural networks (Sect. 10.4) could be applied.

4. **Number of features**: If a lot of features are required to train an accurate model then you risk overtraining. Not only are regularisation techniques likely to be required (Sect. 8.2.5), but the larger number of model inputs will mean more computational cost. Simpler models like linear regression can be quickly trained and they can be utilised within weighted regularisation methods such as LASSO (Sect. 8.2.4) to help with variable selection.
5. The **amount of data available for training**: many machine learning techniques require large amounts of data to create accurate forecast models, whereas some simpler statistical models can be trained with relatively little data.
6. **Number of time series**: If you are forecasting many time series (e.g. forecasts for smart meters from thousands of households) then it may not be practical to generate a forecast model for each time series. Instead you can train a single model over all (or a selection) of the time series. This is called *global modelling*, in contrast to the individual training (local modelling). This is discussed in a little more detail in Sect. 13.4.
7. **Interpretability**: many models, including support vector regression and most statistical methods, are easier to understand than others, such as neural networks, in terms of relating the outputs to the original inputs. This means sources of forecast errors can be more easily detected and fixed, and relationships more easily understood. This also means that further development of the method can also be applied, since weakness in the model are more easily identified. The trade-off is that more interpretable models can often have lower performance.
8. **Forecast judgment**: As a forecaster gains more experience they may come to better understand which methods tend to work well and which ones do not. Unfortunately, this can only come with time and practice.

In the long run the only true test of a methods accuracy is to implement it. When creating forecasts a good adage to remember, attributed to statistician George Box (co-creator of the Box-Jenkins method for ARIMAX models, Sect. 9.4), is "All models are wrong, but some are useful".

However, at first, try models of different classes first to see what works well. For instance, if LASSO regression does not work well as the variables have highly non-linear relationships, ridge regression is also likely to not perform well. Similarly, if the data is insufficient to train a complex LSTM model, suffers from overfitting and needs a lot effort to tune the regularisation parameters in order to outperform much simpler models, a CNN will likely be similarly hard to train for the specific problem.

Chapter 13
Advanced and Additional Topics

This chapter will give brief introduction to a range of more advanced topics which are beyond the primary aims of this book. The further reading section in Appendix D will give some references to other texts which will investigate some of them in more depth.

13.1 Combining Forecasts

In Sect. 10.3.2 it was shown that by combining several weaker models a stronger more accurate model could be produced. This principle can be extended to any selection of forecast models, even ones considered to be quite inaccurate. For example, an average could be taken of the outputs from a linear regression model, an ARIMAX model, and a random forest model to produce a new forecast. Each individual model will have its own strengths and weakness's. By combining them, the idea is to mollify the weakness's of the different models to produce an overall more accurate model. Although this may seem surprising (it could be reasoned that errors would accumulate), combining models has been shown time and time again to be an easy, but effective, way to produce a forecast which is more accurate then any individual model.

Ideally the aim should be to combine models, which are as different to each other as possible to capture different features of the forecast (i.e. models with autoregressive components vs. those with few autoregressive components). Differences can be achieved by having models with different assumptions (e.g., combining a machine learning and a statistical model), models that have different features (e.g., combining a model that uses weather information and one that does not) or different data (e.g., models that are trained on different parts of the available data).

Even when the amount of diversity is limited the combined forecast is often an improvement. Further, it may even be beneficial to retain the least accurate models in any combination as they may model different features of the system that other, more

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_13

accurate, models, may not. For example, one model could better capture weather features, another time of day effects, and another may better estimate the peaks in the demand. The time series forecasting competitions, the *M competitions*,[1] have consistently shown that even compared to high performing singular methods, some of the most accurate models are those which combine traditional machine learning algorithms and statistical models.

The optimal way to combine forecasts is still an active research area but below some simple methods for combining multiple forecasts are introduced for both point and probabilistic forecasts.

There are a number of different ways to combine different point forecast models $f_1, f_2, \ldots, f_n$ but one of the simplest and most popular is to take a linear weighted average

$$\hat{f}(x) = \sum_{i=1}^{n} w_i f_i(x). \tag{13.1}$$

with $\sum_{i=1}^{n} w_i = 1$. Unless there is good reason, often a good initial combination forecast is to use equal weights for the forecasts, i.e. $w_i = \frac{1}{n}$ for $i = 1, \ldots, n$. However, if there is sufficient data for testing the different combinations, and/or there is good evidence that a particular forecast may be more accurate on average than the others, then it may be worthwhile to train or create weights which are tailored towards the most accurate forecast. The optimial weights can be found by testing a range of values over a validation set, however this becomes more complex for larger sets of methods and requires enough data to properly train and validate the weightings. Alternatively, the accuracy of the model (e.g. assessed via the RMSE) could be used to give relative weights to the different forecasts with higher weights for more accurate forecasts. Some other methods are listed in the further reading section in Appendix D.2.

As with point forecasts, individual probabilistic forecasts can be combined together. The evidence is increasing that these combined forecasts are better than the individual methods. This topic is very much an active research area (e.g. see [1]) and therefore much of it is beyond the scope of this book but there are some simple, easy-to-apply techniques that have shown to be effective in certain situations.

Analogous to the weighted averaging presented for point forecasts above, a similar method can be applied to quantile forecasts by averaging the same quantiles from each method. For example, consider forecast models $f_1^\tau, f_2^\tau, \ldots, f_n^\tau$ for the same quantile τ, then create a combined quantile forecast given by

$$\hat{f}^\tau = \sum_{i=1}^{n} w_i f_i^\tau, \tag{13.2}$$

[1] https://mofc.unic.ac.cy/history-of-competitions/.

The same weights should be used for each quantile and are found by minimising a probabilistic scoring function such as CRPS via cross validation (See Chap. 7). As with point forecast model combination the weights should sum to one.

Further literature on combining forecasts is given in Appendix D.

13.2 Hierarchical Forecasting

Electricity networks are naturally hierarchical since traditionally electricity is generated and then transmitted to lower levels of the network. The network is becoming more complicated especially due to the increased installation of distributed generation sources like wind turbine and solar photovoltaic farms.

Matching supply and demand must be achieved throughout the network rather than simply at the aggregate level. Hierarchical forecasting refers to generating forecasts at different levels of the hierarchy. This is demonstrated in Fig. 13.1. In load forecasting this could include forecasting the demand for an individual household, the aggregated demand at the secondary or primary substation, and also at the national level. Of course you can also include all the intermediate levels inbetween.

A useful property of hierarchical forecasts is to ensure that they are *coherent*. In other words aggregations of forecasts at lower levels equal the forecasts of the aggregated level. This is useful to anticipate or co-ordinate multiple applications (e.g. flexibility services such as batteries) at different levels of the hierarchy.

Note, that the aggregation of the demand monitored at the lower level is unlikely to match the demand at the higher level because not all loads are monitored (for example street furniture). Further, there is often switching on the network where demand is rerouted to other networks. Switching on a substation is illustrated in Fig. 13.2. Demand from a nearby network can be rerouted to another substation when there is, for example, a temporary fault. Hence the demand on a network may have a shift to a different regime of behaviour (see Sect. 13.6). Finally, there are electrical losses on a network since some energy is lost in the distribution process. This is another reason the aggregation of demand at a substation is unlikely to match the aggregation of the downstream connected loads. By extension, the forecasts of the demand at the substation are unlikely to match the aggregation of the forecasts of the individual loads (even in the unlikely situation of a perfect forecast!).

To simplify the following discussion lets assume that there is no switching, there is minimal loses and we happen to have access to all the major downstream loads connected to the substation. To make this more precise, let D_t be the demand at time t for a substation and let $L_t^{(k)}$ be the downstream load from connection $k \in \{1, \ldots, K\}$ at the same time. In this scenario the following holds

$$D_t = \sum_{k=1}^{K} L_t^{(k)}. \tag{13.3}$$

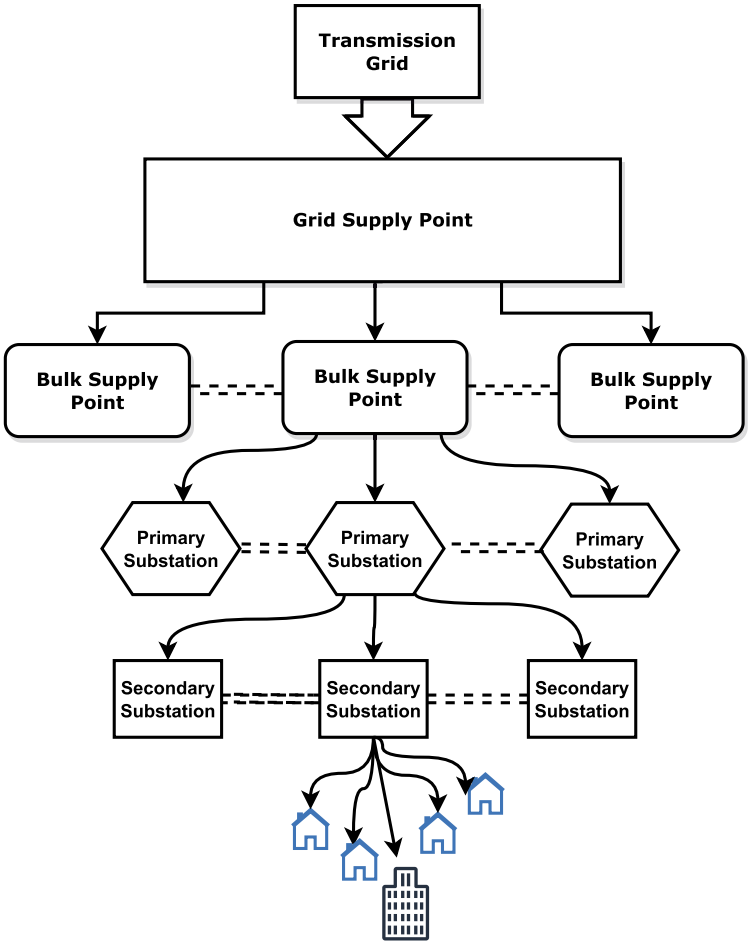


Fig. 13.1 Electricity network hierarchy showing multiple levels from Transmission level down to individual households and buildings. Dotted lines show potential linking between load points at the same levels, i.e. between bulk supply points

Now assume that a forecast is produced for each time series, which we denote by using a hat, e.g. $\hat{D}_t$ is the forecast estimate at time t for the substation demand, and similarly $\hat{L}_t^{(k)}$ is the forecasts for the downstream demands. However, in general, due to forecast errors and biases it is unlikely that the forecasts match, i.e. the following

$$\hat{D}_t \neq \sum_{k=1}^{K} \hat{L}_t^{(k)}. \tag{13.4}$$

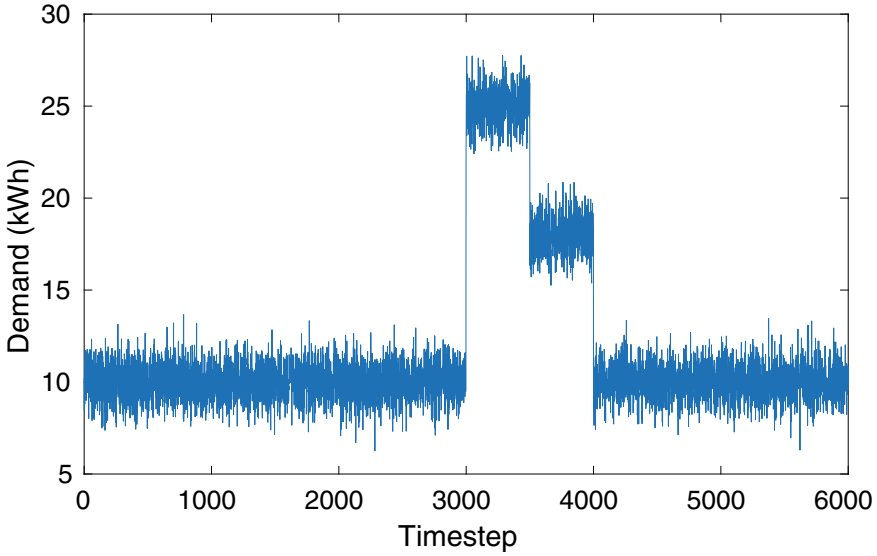


Fig. 13.2 Illustration of the effect of demand switching on the total demand on a distribution substation

A simple way to produce coherent forecasts would be to only produce the forecasts for the downstream levels $\hat{L}_t^{(k)}$ and estimate the other levels through aggregation. The issue with this approach is that the aggregate forecast is likely to be less accurate than a direct forecast of D_t. This is because the lower aggregate series are more volatile and therefore less easy to accurately forecast. Further if errors are correlated in the downstream forecasts, they may accumulate when aggregated. An alternative is to forecast the substation and then split the demand at the lower levels. However, given the complex behaviour across the downstream demands it is likely these forecasts will be inaccurate since it is not obvious how to disaggregate the demand, especially if it changes depending on time of day, time of year and on special days (Christmas, New Year etc.).

In fact let $\hat{\mathbf{L}}_t = (\hat{D}_t, \hat{L}_t^{(1)}, \ldots, \hat{L}_t^{(K)})^T$ be the vector of forecasts for each time series in the hierarchy. A coherent forecast, $\tilde{\mathbf{L}}_t$ can be written generally as

$$\tilde{\mathbf{L}}_t = \mathbf{S}\mathbf{G}\hat{\mathbf{L}}_t \tag{13.5}$$

where $\mathbf{S} \in \mathbb{R}^{5\times 4}$ is sum matrix which sums lower levels up to the higher levels of the hierarchy and $\mathbf{G} \in \mathbb{R}^{4\times 5}$ for $M > 0$ is a matrix that maps the forecasts to the bottom level of the hierarchy, and depends on the method deployed.

In the special case of the simple coherent approach which simply sums the lowest level forecasts this gives the matrices

$$\mathbf{S} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$\mathbf{G} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

G drives the different approaches, but a manual choice is likely to be suboptimal, i.e. not produce the most accurate set of coherent forecasts. There are many ways to define optimal but one such method was given by the authors in [2] which minimises the total variance of the forecasts. The details are beyond the scope of this book (see [2] or [3] for complete details) but the final choice is given by

$$\mathbf{G} = (\mathbf{S}^T\mathbf{W}^{-1}\mathbf{S})^{-1}\mathbf{S}^T\mathbf{W}^{-1}, \tag{13.6}$$

where $\mathbf{W} = \mathbb{V}ar(\hat{\mathbf{L}}_t - \mathbf{L}_t)$ is the covariance of the baseline forecast errors.

This example has only shown one level in a hierarchy but of course this can be extended to have multiple levels. For example in distribution networks this could consist of residential smart meters at the lowest level, which aggregate up to the secondary substations, several of which aggregate to primary substations (see Fig. 13.1).

Coherency can also be applied to probabilistic forecasts but this is much more complicated and beyond the scope of this book. This area has a lot of interest, and more new and novel results can be expected. Further reading on this topic is cited in the Appendix D.2.

13.3 Household Level Forecasts

Forecasts at the household level (or aggregations of only a few households) have specific features and challenges. In this section the focus is on the specific problem of trying to measure the errors for point forecasts of household data.

The unique problems occur because data at this level is particularly spikey and hence very sensitive to small changes of the occupants within the household. To illustrate this consider a household with a single occupant who, every weekday, wakes up, leaves, and returns homes at roughly the same time each day. When the occupant comes home they turn on various appliances, lights, heating, TV, cooker etc. Although there are often similarities from week-to-week, there is still significant irregularity. Figure 13.3 shows daily demand for three consecutive Monday's for a real household overlayed over each other. This example shows volatility, especially

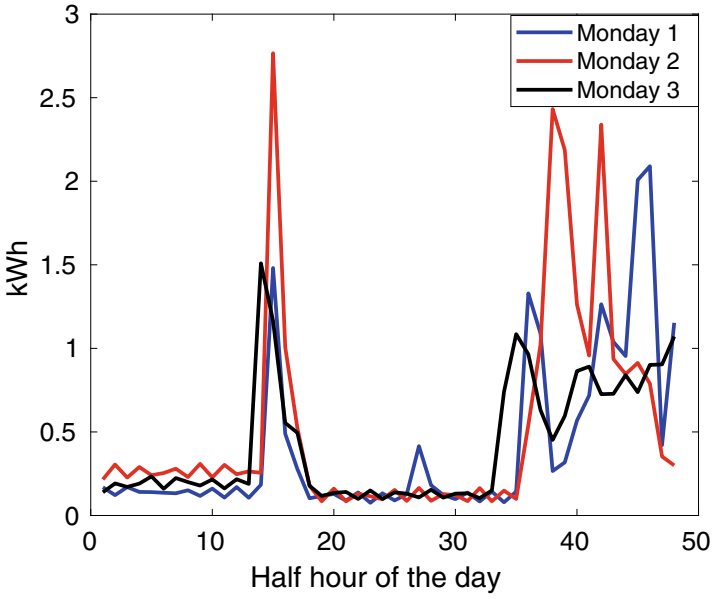


Fig. 13.3 Overlay of three consecutive Monday's demand profiles from the same household. Constructed using data from the CER Smart Metering Project—Electricity Customer Behaviour Trial, 2009–2010 [4]

in the evenings (in fact the demand is actually much more regular than many other households, and highlights how irregular such demand can be).

Due to natural variation in behaviour, and other unexpected events, demand profiles are likely to change even for the most regular consumers. For example, unexpected traffic in the morning, or a missed alarm will result in the occupant arriving late to work. In turn, the late start may mean the occupant now decides to work late and thus arrive home later than usual. An illustration of such a profile is shown in Fig. 13.4, for this the original "typical" profile is shifted slightly.

This unique feature does not typically occur at aggregations of over 10 households. The individual demands and their irregularities smooth out and the data is no longer spikey (This is shown in Fig. 1.2 in Sect. 1.2). This relationship is emphasised in the case study in Sect. 14.2, which shows the power law relationship between aggregation size (size of feeder equating to more households) and relative error (Fig. 14.8). The scaling law shows it becomes increasingly difficult to forecast accurately at low aggregations relative to higher aggregations.

The "spikeyness" of household level forecasts also produces a specific problem in terms of measuring the errors of household level forecasts. Consider comparing the two forecasts as illustrated in Fig. 13.5. One forecast is a simple flat forecast (made say from taking the average half hourly demand from the actual daily demand), the second forecast is a simple shift on the actuals (this could be viewed as a seasonal persistence forecast (Sect. 9.1) where the profile is relatively regular but with small

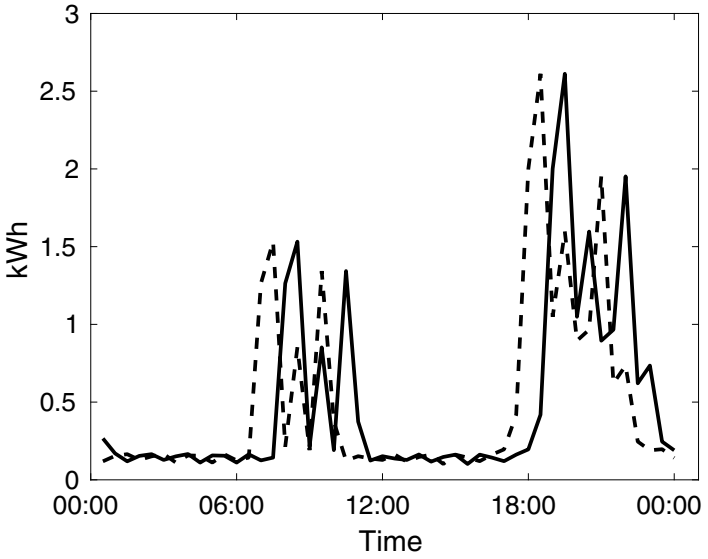


Fig. 13.4 Daily demand for one profile, together with the shifted profile. Constructed using data from the CER Smart Metering Project—Electricity Customer Behaviour Trial, 2009–2010 [4]

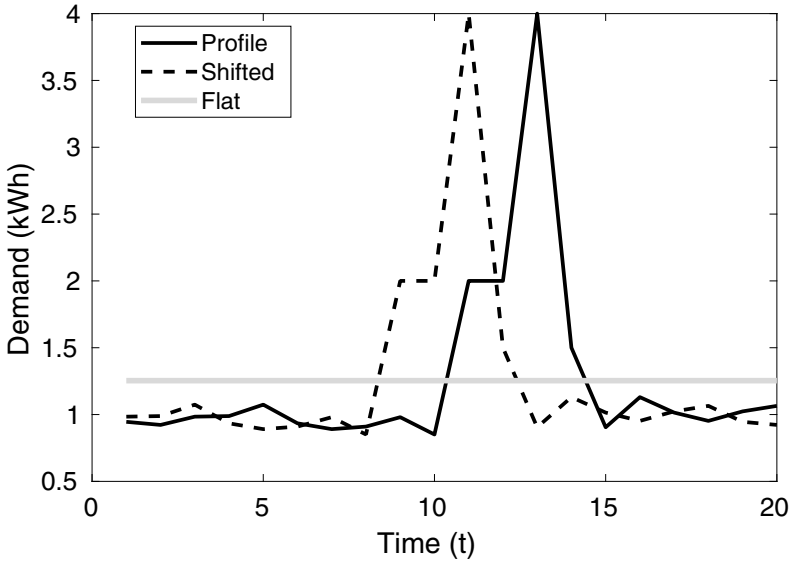


Fig. 13.5 Demonstration of the double-penalty effect. The actual demand (bold line) is compared to a shifted version (dashed line), and a flat estimate (grey line) which is just the uniform profile formed from the average of the actual

shifts in demand). The second forecast is subjectively quite good, and potentially useful. It correctly estimates the peak magnitude and is only slightly misaligned with the peak timing. For an application like storage control for peak reduction (Sect. 15.1), this second forecast can help inform the appropriate charging schedule for a storage device since the peak is correctly anticipated albeit at a adjusted time. This means a battery can be prepared with a full charge and ready to discharge when the peak does finally appear (this presumes there is some sort of monitoring of the demand which identifies the appearance of the peak of course).

However, in the situation described in Fig. 13.5, despite being a useless forecast providing no information about the peaks, the RMSE error for the flat estimate (0.71) is smaller than that for the peaky forecast (0.91). The reason for this is the so-called **double-penalty effect**. For any pointwise error metric like RMSE the peaky forecast will be penalised twice: once for estimating a peak that didn't occur, and a second time for missing the peak that did occur. In contrast, the flat forecast is only penalised once.

There are two main options to deal with this situation:

1. Develop new error measures that reduce the double penalty effect and produce a more representative and smaller score for forecasts that more closely describe the peak magnitude and timing differences.
2. Consider probabilistic forecasts. Since the estimates now estimate the spread of data, variable timing of peaks will be more accurately captured. In particular, multivariate probabilistic forecasts (Sect. 11.6) will capture the uncertainty and the interdependencies in the data, and if properly calibrated will include ensembles which represent the wide range of possible household profiles.

Probabilistic methods are the most desirable since they have proper scoring functions (Sect. 7.2) with consistent interpretation. This means they can be assessed in an objective way. The drawback is the computational costs, and the large amount of data required to train a probabilistic forecast. Techniques for probabilistic forecasts have already been introduced in Chap. 11 so the rest of this section discusses alternative error measures for point forecasts.

One set of options are so-called **time series matching algorithms**. These are popular techniques used in areas such as speech recognition to show how close individual signals are. One such technique, dynamic time warping, has already been introduced in Sect. 10.1. This creates two new time series by stretching them to find the closest match. Then the original point wise forecast error measures can be used to measure the difference.

There is a number of drawbacks with this technique. Firstly in the standard DTW approach there is no penalty or limit on how much a time series can be stretched to enable a match of the features. Secondly, the ordering is fixed. If there are multiple peaks at similar periods but in a different order then dynamic time warping will match one peak but not the other. Since energy demand will likely allow different orders of appliances being used (TV on then the Oven, or Oven on then the TV) this may be too restrictive for household demand forecasts.

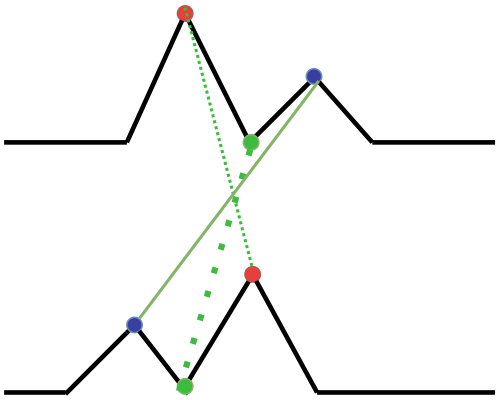


Fig. 13.6 Illustration of the matching between two time series performed by the adjustment error approach

An alternative method, as developed by the authors is the so-called **adjusted error measure** [5]. This allows a restricted permutation of the time series around each point. This can be solved relatively efficiently for small numbers of time series using an assignment algorithm called the Hungarian method. A basic illustration of the matching is shown in Fig. 13.6. The adjusted error measure matches a time series like dynamic time warping but allows reordering of peaks (within a limited area). The drawback to the adjusted error measure is that there is no penalty on permuting the time series within the limited area and the size of this permutation length must be chosen before hand.

In summary, no matter measure, point forecasts for households will require some subjective choices in order to deal with the spikey nature of the demand.

13.4 Global Verses Local Modeling

Traditionally in time series forecasting, a model's parameters are fitted based on historical data from the same instance one wants to forecast. As discussed in Chap. 2 in the context of load forecasting this instance can, for example, be a building, a household, a substation or another point of interest in the grid. So, if one trains a model for a specific building, the forecast model parameters are estimated using this building's available historical data. This was the same initial approach that was taken with the advent of machine learning models.

However, as discussed in Chap. 10, machine learning models, especially deep models such as LSTMs and CNNs tend to overfit when there is insufficient data. To mitigate against this a new possible strategy was developed for when implementing more complex machine learning models referred to as **global modelling**. In this approach the model is trained on data of multiple diverse instances, e.g. different buildings, simultaneously. Note that the instance that the forecast is made for, may or may not be part of the training data. The global modelling approach makes use

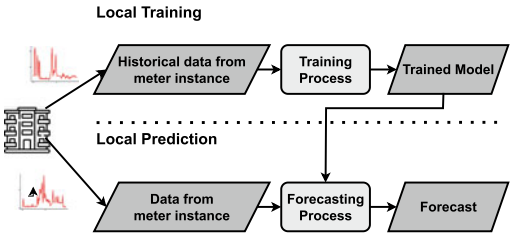


Fig. 13.7 Traditional local modelling process

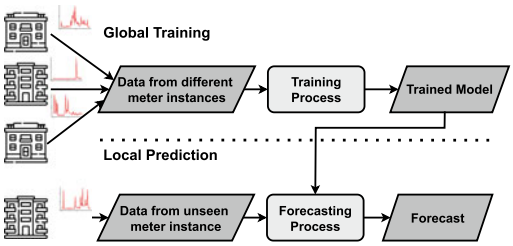


Fig. 13.8 A global modelling process for deep learning models

of the fact that deep machine learning models can learn general enough feature representations, which often generalise also to other instances.

The main benefit of global modelling is that having more data is often advantageous to deep learning models and avoids overfitting. Further, learning a model for each instance can be highly impractical. For example, the aim in many countries is to ensure most homes have smart meters and, if the intention is to provide smart services, such as storage control or home management systems, it may be impractical to train an individual model on each home. Instead global models may be more feasible. The traditional approach of fitting the model on the same instance is now referred to as **local modelling**. Figures 13.7 and 13.8 illustrate these two approaches to load forecasting.

However, if there are too many different instances in the data, there are typically diminishing returns, i.e., adding more data to the dataset does not lead to improvements. In fact, performance can even degrade as more data is added. This could especially be the case if many load profiles are added that are too diverse and if the amount of data is too large relative to the capacity of the model used. To mitigate this problem, alternative hybrid strategies have been proposed. A typical approach is to initially cluster the data to find groups of similar instances and then to train a global model on each cluster. At inference time for a particular instance, one needs to determine what cluster a specific new instance would belong, e.g. in k-means this would be through comparing the profile to each clusters representative profile, for finite mixture models (Sect. 11.3.2) this would be via calculating its membership posterior probability and then assigning to the cluster with the highest value. The prediction is then made by applying the trained model from this cluster to the particular instance. Figure 13.9 illustrates this process. The process is sometimes referred to as **pooling**.

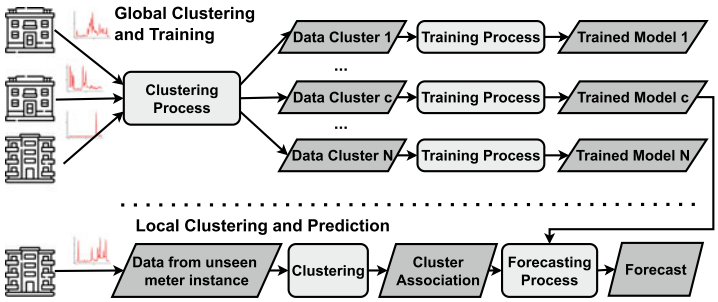


Fig. 13.9 A "pooling" approach to global modelling

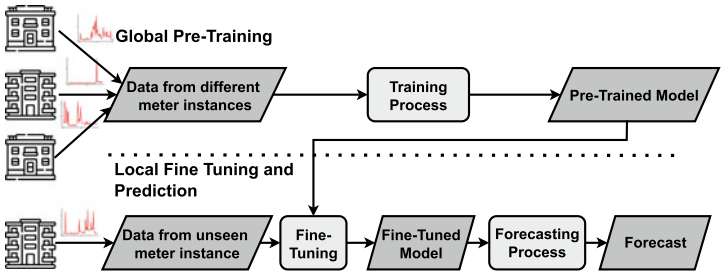


Fig. 13.10 Process of fine-tuning a pre-trained model

As briefly discussed with CNNs in Sect. 10.5.2, the lower layers of an ANN or the filters in a CNN have usually learned feature representations that are often general enough to also work as a feature extractor for other similar tasks. In other words, one can re-use these pre-trained parts of the neural network and use it for another related task. This general idea of reusing parts of an already trained ANN is called **transfer learning**. Figure 13.10 illustrates this approach to forecasting. This is the common procedure when working with state-of-the-art image or language models, where such models are **pre-trained** on very large image or text datasets to learn first some general features of images or text before being trained on the dataset for a specific task.

Local modelling is often not an effective strategy for deep machine learning models unless there is a lot of data available for the instance, or unless the resolution is sufficiently high. Hence, any of the global strategies or variations thereof should be explored. This has the benefit of improving generalisation and avoiding overfitting, but is also, in practice, a much more computationally effective strategy, as typically training a global neural network on multiple instances is more computationally effective than training multiple local models individually. This saves cost for computing, energy and hence unnecessary CO_2 emissions from the compute (see, for instance, [6] on the carbon emissions of machine learning training). Even if the model is transferred to a local computer (e.g. a building energy management system), there are no

real data privacy concerns in the context of models trained on smart meters. When the model was trained on multiple instances, no method is known to re-create the instances used to train the model.

13.5 Forecast Evaluation: Statistical Significance

It is often the case that several methods are performing very similarly but in fact there is no statistical significance between them. This can be a desirable situation as it means you may be able to choose a forecast model with other useful properties (say it has high interpretability, or low computational cost) without sacrificing accuracy. It is also important to rule out that a forecast is performing well by chance alone.

To tell if two time series forecasts are significantly different requires a statistical test. One of the most popular methods is the **Diebold-Mariano Test**. As with many statistical test, it begins with a null hypothesis, denoted H_0, which in the case of load forecasting states that "the two time series forecasts are equally accurate on average". The aim of the test is to see if the null hypothesis can be rejected, i.e. trying to show that the forecasts are in fact not equally accurate.

Suppose one forecast produces one-step ahead forecast errors $e_1, e_2, \ldots, e_N$, and the second forecast produces errors denoted by $\epsilon_1, \epsilon_2, \ldots, \epsilon_N$. Consider the loss differential series given by

$$d_t = g(e_t) - g(\epsilon_t), \tag{13.7}$$

where g is a loss function, usually defined as $g(x) = x^2$, or $g(x) = |x|$. Note that two forecasts have the same accuracy if the expected value of the differential loss is zero. Hence the null hypothesis can be reframed as $\mathbb{E}(d_t) = 0 \ \forall t$. The main assumption for the Diebold-Mariano test is that *the loss series is stationary*.

To perform the test requires producing a test statistic, i.e. a value derived from the observations. This statistic should follow a particular distribution under the null hypothesis. If the actual calculated value is found to be very unlikely it is evidence for rejecting the original hypothesis. "Unlikeliness" is determined by a significance level α and a p-value. The p-value is the probability of obtaining a value as extreme as the observed value, assuming the null hypothesis holds. The significance level is a value determined by the user before the experiment/test-statistic is derived and sets the threshold for rejecting the null hypothesis.

An example for a hypothesis test shown in Fig. 13.11 for a standard normal distribution, i.e. $Pr(x < X|H_0) \sim N(0, 1)$ which represents the distribution of points assuming the null hypothesis is true. Suppose in this example, it is undesirable for the test statistic, T, to lie in the extremes of the distribution. If it does occur in the extremes, the null hypothesis can be rejected. This is a **two-tail test**.[2] Note then that this means the p-value is given by $p = 2\min\{Pr(x > T|H_0), Pr(x < T|H_o)\}$. Suppose the significance level is chosen as $\alpha = 5\%$, which is represented as the shaded

[2] A one tail test will naturally only consider one of the tails.

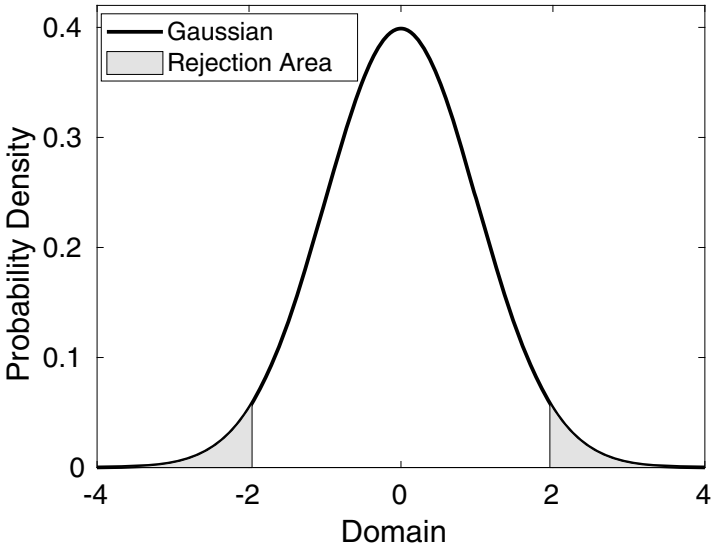


Fig. 13.11 Illustration of hypothesis testing with assumed standard Gaussian distribution. Shaded are the areas which represent the 5% extreme values which determine a rejection of the null hypothesis

part in Fig. 13.11 with a probability of 0.025 in each tail (i.e. total of $\alpha = 0.05$). In other words, if the statistic lies in the shaded area (i.e. the quantiles $z_{\alpha/2}, z_{1-\alpha/2}$ determined by $\alpha/2$ and $1 - \alpha/2$) the null hypothesis is rejected, i.e. if $|T| > z_{\alpha/2}$. If it is not in the tails, then the null hypothesis is not rejected. Note that since we are considering a standard normal distribution $z_{1-\alpha/2} = 1.96$ and $z_{\alpha/2} = -1.96$ as it corresponds to the 97.5th percentile value.

Returning to our example of comparing time series forecasts. The Diebold-Mariano statistic [7] is defined as

$$DM = \frac{\bar{d}}{\sqrt{\left(\sum_{n=-(h-1)}^{h-1} \gamma_n\right)/N}}, \tag{13.8}$$

where $\bar{d} = \frac{1}{N}\sum_{n=1}^{N} d_n$ is the sample mean of the loss differentials and γ_n is the autocorrelation of the loss differentials at lag n (Sect. 3.5). The h used in the denominator should be large enough to include any significant lags in the autocovariance and can be checked through the autocorrelation plots (Sect. 3.5). Assuming the loss differential time series is stationary then the DM statistic follows a standard normal distribution (Sect. 3.1).[3] A two-tailed test can thus be performed using the same

[3] The derivation of this result is beyond the scope of this book, but is a result of the central limit theorem, a powerful theorem which essentially says that the distribution of sample means is Gaussian.

distribution as shown in Fig. 13.11. Rejecting the null hypothesis in this case means that the two forecasts are not of similar accuracy.

For small samples the DM statistic can result in the null hypothesis being rejected too often. To account for this there is adjustments which can be applied such as the Harvey, Leybourne and Newbold test [8]. This adjusted statistic is as follows

$$HLN = DM\left(\frac{(N+1-2h+(h(h-1)/N))}{N}\right)^{1/2}. \tag{13.9}$$

Instead of the standard normal, this corrected statistic must be compared to a Student-t distribution with N-1 degrees of freedom. A Student-t distribution is similar to a Gaussian/normal distribution (Sect. 3.1) but with heavier tails, i.e. the extremes of the distribution converge to zero slower than a Gaussian distribution. Thus for this type of distribution very large/small values are more likely to occur.

It should be noted that the DM tests can be applied to more general error functions, including multivariate forecasts, [9]. Further reading in this area in provided in Appendix D.2.

13.6 Other Pitfalls and Challenges

There are other common difficulties that can be encountered when developing short term forecasts. A few of them are outlined here.

13.6.1 Collinearity and Confounding Variables

Correlation is an extremely useful property when producing forecast models, but it can also create complications. As it is commonly known "correlation doesn't equal causation". However, this does not mean the variable can not be used to produce accurate forecasts. It does mean care should be taken when assuming and interpreting the relationship between these variables. A variable with a spurious correlation may not be initially useful for the model but since the relationship is not causal it may not generalise well, and can create inaccuracies in your model at a later point.

The challenges extend beyond the independent and dependent variables. When two of the independent/predictor variables are highly (linearly) correlated they are called **collinear** and can create difficulties in interpreting their effect on the dependent variable. It can also increase the chances of overfitting. Adding a collinear variable may not add much accuracy to the forecast since much of the effect has already been captured by the correlated variable already included in the model. As an example, consider ambient temperature and wind chill. Wind chill is a combination of temperature and wind speed and estimates how the temperature feels to

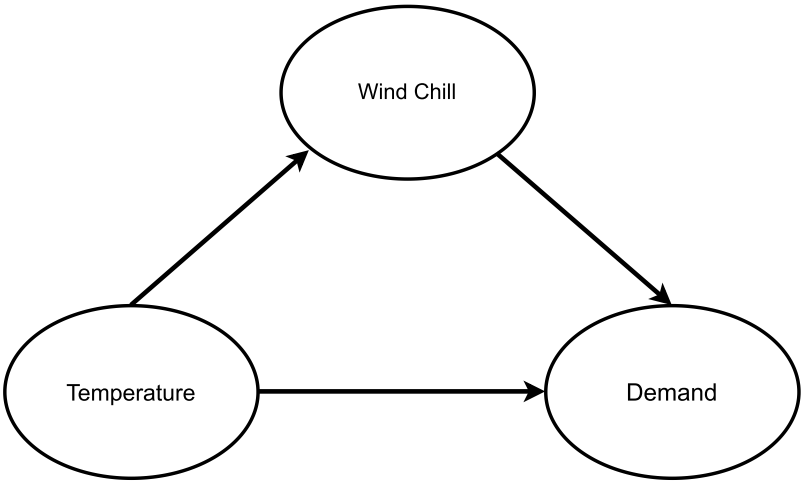


Fig. 13.12 Causal diagram showing the relationship between temperature, demand and wind chill

a human. Thus these two variables are often strongly correlated to each other. The causal relationships between them are shown in Fig. 13.12.

Collinearity can also effect the sensitivity of the model coefficients to whether those variables are in the model or not [10]. For a linear regression it therefore effects the precision of the associated coefficients. There are tests to help identify collinearity effects, one of which is the variance inflation factor (VIF) (see, for example, the book by Ruppert and Matteson [10]). This measures how much the variance of an independent variable changes when the effects of the other independent variables are compared. In addition correlation functions can be used to identify which variables are highly correlated.

Ideally, it should be checked whether the collinear variables improve the forecast model accuracy when they are included versus when one, or both are removed. If interpretation and stability is not important then the most accurate forecast should be retained. However, note that if both are retained and there is a concept drift (Sect. 13.6.3) in only one of the variables this can reduce the accuracy on new unseen data. Regular model retraining can reduce the likelihood of this.

There are a few approaches which may be able to reduce the collinearity effects. The most obvious is to prune the variables amongst those which have the strongest correlation and or highest VIF, possibly keeping those which have the strongest causal link with the dependent variable (causality not a particularly easy task to prove of course). Model selection techniques, such as the information criteria, as presented in Sect. 8.2.2, are another possibility for finding an accurate model with the lowest required variables.

Of course this collinearity can extend across multiple variables, in which case it is referred to as **multicollinearity**.

A related concept is that of **confounding variables**. If a variable causally effects both the dependent and an independent variable it is a *confounder* for them both. In Fig. 13.12 temperature is a confounding variable since it causes changes in demand but also is a major determinant of the wind chill values. Not taking into account confounders can make it difficult to understand the causal relationship between variables, and may mean an effect or correlation is overestimated. As an example, consider a model of the net demand linked to the wind chill values. Suppose wind chill has no effect on demand, but temperature does. Since temperature is also related to wind chill (Sect. 6.2.6) then it could be perceived that a change in wind chill is correlated to the change in demand, whereas in fact this is spurious since it is the temperature which are driving the related changes. Regardless, there may still be some influence of the wind chill on the demand which can improve the accuracy of the forecast model, but this may not clear because it is confounded by the temperature variable. If you are interested in the effect of wind chill you need to isolate it by controlling for the temperature. One way to do this is to control for the temperature by learning the relationship for fixed (or binned) values of the temperature.[4] This is not easy as you have to reduced the overall data from which to train the model since you need to learn on subsamples of the overall data. For linear relationships, linear regression can also be used to identify confounding variables and the size of the effect. This is done by comparing the effect (i.e. the associated coefficient) on regressing two models for the independent variable against the dependent variable, but including the suspected confounding variable in one of the models. Large changes in the coefficient can suggest a confounding variable with the magnitude of this change indicating the size of the effect.[5]

If the confounding variables are not included in the models then there could be a loss of accuracy. However, including them means that interpreting the results can be difficult, as shown above. Again, this may not be a major concern if the focus is on performance rather than interpretation. However spurious correlations may decrease generalisability of a model. Cross validation methods can help with model and input selection (Sect. 8.1.3) and thus ensure that the model is still generalisable.

Ideally all confounding variables and those which have causal effects on the dependent variable will be included in a model, but this may be very difficult. One issue caused by not identifying confounding variables is that the assumed relationship between the independent and dependent variables may be tenuous, and thus sudden changes in the independent variable may mean the model no longer holds or becomes very inaccurate. For example, suppose a model has been created for the total demand at a local substation connected to five unmonitored households. If it is found that the monitored demand behaviour of another household (but on a different network) is very similar it could be included in the model to help increase the accuracy. However

[4] This is also known as stratification.

[5] This may not be straightforward. An interesting example of the complications in trying to interpret the effects from various variables with many, and possibly unknown, confounding variables is shown via the so-called 'Table 2 fallacy', see [11]. The example shows that even variables that are not confounded can still create difficulties with interpreting the associated effects.

if, unknown to the modellers, one of the five households moves house the correlation with the demand no longer holds and may produce very inaccurate results.

The case study in Sect. 14.2 suggests a real example where it is possible seasonality effects may be a confounding variable for the temperature.

13.6.2 Special Days and Events

The forecast models described throughout this book typically assume that the future load will look similar to the historical load. To do this, data analysis and visualisation methods are deployed to look for common patterns in the data. Therefore a forecast model which uses these features will, on average, be very accurate for typical days. Unfortunately it may not do well for less common days or events.

More often than not these untypical days will be holidays such as New Years, Bank Holidays, etc. I.e. special days where businesses may be closed, houses may be occupied in different ways, and there may be different behaviours than usual. These special days will vary depending on the country and culture, for instance predominantly Christian countries will include Christmas, predominantly Muslim countries will include Ramadan, and North American countries will have Thanksgiving, etc. On days like these, workplaces may be closed, and there may be other behavioural changes (e.g. extra home cooking).

Another cause for less typical energy usage days may be because of special events. This could be a World Cup final, the Olympics, or a Royal Wedding which causes many more households to watch TV in their own homes, or perhaps go to public areas to watch communally. Other special days can be created due to highly atypical weather events. For example, a heat wave, a storm or an usual cold snap.[6]

Each of these types of events can cause energy demand to be very different than from what regularly occurs in the homes, or over an aggregate of homes. If they are not taken into account then the forecast models are likely to be inaccurate on these special days.

Ideally, these special days should be treated within the model. This can be through dummy variables (Sect. 6.2.6), or a separate model could be created for them. The problem with these approaches is that these special events are often very rare. In other words, there is not usually much data to train the models to accurately forecast special day demand. Take for example Christmas day which occurs on 24th December each year. There is only one day per year in which to train on and further to this, each year Christmas will fall on a different day of the week. It could be Monday one year, and Tuesday on another etc. This means that each Christmas day may be slightly different depending on which day of the week it falls on. This further reduces the relevance of the data for training.

[6] For example, the 'Beast From the East' cold wave which occurred in February and March of 2018 in the UK.

There is not many robust ways to alleviate these problems. With so few examples it is difficult to learn the regular patterns which means heuristics or educated guesses must be made about what data can be used to inform the models. A common way to do this is to use other data as proxies for the demand on the special day. For example, it is reasonable to assume holiday dates (like Bank Holidays) are very similar to weekend dates, with occupants acting similar on both. In these models the special days can be assigned the dummy variables for the weekend (or say Sunday).

13.6.3 Concept Drift

Concept drift in machine learning refers to the change in the statistical properties of a dataset over time. Often these changes can occur without the modeller being aware, as they can be subtle and/or take place very quickly. This creates a lot of problems for forecasts since the distribution of the random variable is changing and may mean the distribution in the test set is not going to be the same as the training set.

There are many reasons that energy demand may change over time, including:

- Churn of customers: If the occupants of many households or businesses connected to a LV network change then the demand behaviours will likely change.
- Uptake of new technologies: Disruptive technologies such as heat pumps, electric vehicles and photovoltaics can have major effects on the demand and change the aggregated loads on the distribution networks.
- Atypical Events: Special events (See Sect. 13.6.2) can create unusual demand. This includes unusual weather and special sporting events (Olympics, World Cup etc.).
- Improving Energy Efficiency: New technologies can mean reductions in demand. For example, new LED lights and TVs are much more efficient then their predecessors.
- Change in Energy Prices: Volatile energy prices can mean households and businesses have to change their behaviours or appliances to better reduce their costs.

Many of the concept drifts in electricity time series will be temporary. Efficient versions of appliances may change the demand behaviour when they are first installed but the future demand will then be stable until another major change. This suggests a way to reduce the effects of permanent changes in time series: **adaptive methods** or tracking methods. These methods use a limited window of the most recent observations to ensure that models can react to changes in the demand. When a change in the distribution occurs, the model will eventually train to this new distribution. Of course, it may require an certain period of time to pass until model accuracy is restored since the training data may consist of a mix of data from before and after the distribution change.

The change in demand can be detected by so-called **change-point analysis**, which aims to identify the time point where the data before and after the change-point has different statistical properties. A related time series property to concept drift is **regime switching**. This is where the time series has finite number of states or "regimes",

which the time series may switch between. To generate forecasts in these cases means generating a forecast model for each regime. This has the challenge of predicting which regime the time series will be in as well as the demand within that regime. An example in energy demand may be a commercial building which has different uses across the year. For example, University halls of residence may be empty over the summer whereas the rest of the year they are occupied by students.

13.6.4 Unrealistic Modelling and Data Leakage

Another pitfall in forecast experiments is the unrealistic assumptions and modelling which are often deployed. Ideally, forecasts should be designed to replicate the real-world scenario for which they are being generated. However, in many cases, to simplify the analysis, or due to lack of resources, a forecast model used in a desktop study may be non-replicable or impractical to apply in reality.

The most common mistake is the use of data which would not actually be available in practice. For demand forecasting this is usually weather data. Weather is considered a strong driver of electricity demand due to heating or cooling needs. Therefore weather forecasts can be useful inputs for accurate load forecasts. Unfortunately, weather data, in particular weather forecast data, can be difficult to source. Instead many experiments resort to using weather observation data which of course would not be known in advance. However, it is often easier to collect observed data. Any forecast that uses data which would only be available after the forecast is generated is known as an *ex-post* forecast. Those which only use data which is available at the time the forecast is generated, are known as *ex-ante* forecasts. This can be viewed as a form of **data leakage**. This term refers to using data in machine learning experiments when they would not be available in practice. This is often the case when test data is inadvertently included in the training leading to an unrealistic performance on the test set. In time series forecasting, other forms of data leakage may occur if non-time series splits are utilised in the cross-validation, such as the shuffled split (see Sect. 8.1.3).

Another practical consideration when developing load forecasts is understanding what load data would be available for use in the forecasts. For example, household smart meter data is often only transmitted to the data platforms/exchanges, at most once a day. Any model waiting for such data to train a model, for example a storage control scheduling algorithm (Sect. 15.1), may not have timely access for training and transmission. In addition to the data not being available due to collection restrictions, the speed of an algorithm to train or generate a forecast may be insufficient to be able to use the new data in time for the required application.

There may also be external constraints that limit what data is available. For example, if you are utilising weather forecasts, most numerical weather prediction centres only reveal the updated weather forecasts at particular times of the day, e.g. midnight, 6AM, noon, and 6PM since it is often computationally infeasible to do global predictions more frequently. Therefore the weather inputs used for a rolling

load forecasts may not be the most recent depending on the time of day. In summary, communications limitations, regulatory restrictions, commercial considerations, and computational expenses can all have implications for what data should or could be used when training or developing a forecast. The main point is that as many realistic assumptions should be embedded in a forecast trial or experiment to ensure they mimic the real-world conditions of the application they will be applied in.

13.6.5 Forecast Feedback

In some cases the forecasts can effect the demand itself, which will now mean that the forecast itself is incorrect. This feedback is especially true if linked to electricity costs. For example, suppose that a commercial business sees a load forecast which identifies that a large peak will occur during a period when electricity prices are high. The business then may decide to make plans to change their behaviour, e.g. turning off non-essential equipment, or shifting their demand. This will reduce the predicted peak and hence reduce their costs. This means that the forecast is now technically incorrect a condition created by the forecast itself. This is completely valid, after all forecasts are there to try and support many applications (Chap. 15) with the objective in many cases to reduce the demand and costs for consumers.

The feedback effect not only reduces the accuracy of the forecast but it also changes the training data itself. Where the forecast has influenced the observed demand this data may not describe the normal behaviour or features of the time series. As another example, consider the battery storage device example in Sect. 15.1. The forecast is used to help plan the charging and discharging of the battery on a feeder, however this also changes the demand on the feeder. What was originally going to occur is now unknown unless the charging of the battery is also recorded. These adjustments therefore effectively change the regime of the data during those periods (See Sect. 13.6.3) and should therefore not be used in the training for the 'typical' behaviour of the time series.

The question is how to deal with such feedback for future training? In the case of a storage device, if the changes are known and recorded then the uninfluenced demand can be recovered. However, in the cases where the original underlying demand is not known (such as with demand side response, see Sect. 15.2) then one solution is to not train the normal demand on those periods where there has been interventions. This is only practical if there is sufficient data where no interventions have occurred, otherwise the final models may be inaccurate. Alternatively, it may be possible to learn the intervention effect itself by incorporating it into the model, possibly even as a dummy variable, (Sect. 6.2.6), that way more of the training data can be utilised.

13.7 Questions

For the questions which require using real demand data, try using some of the data as listed in Appendix D.4. Preferably choose data with at least a year of hourly or half hourly data. In all the cases using this data, split into training, validation and testing in a 60, 20, 20% split (Sect. 8.1.3).

1. Generate some day ahead point forecasts for some demand time series. Preferably a few benchmark models, a few statistical methods and a few machine learning models. On the test set compare the RMSE errors of the individual models. Now combine all the models and calculate the RMSE, does it improve compared to any or all of the individual models? Try different averages of the models, for example in one case sample the two best, or the two worst. Try mixing some of the best and worse. See if some of them give smaller RMSE than the other combinations? Take the two best forecasts and see if they are statistically significant using the Diebold-Mariano test.
2. Take a collection of at least 100 smart meter demand time series. Partition them into ten sets of ten and aggregate each set. Now create a forecast for each aggregate over the test set. Now aggregate all set of ten (so they are now an aggregate of 100 smart meters) and using the same forecast model (trained on the aggregated demand series of 100 smart meters) generate a forecast on the test period. Calculate the RMSE error for this forecast. Now aggregate the forecast of the ten sets and calculate the RMSE error there too. Compare these two values, which is more accurate? Which would you expect to be more accurate?
3. Can you think of some other forms of data leakage?
4. What else may cause concept drift? Can you think of changes in your home which would cause reasonably large changes in your usually demand behaviour? Can you think of other buildings which may have some dramatic changes in their behaviour?
5. Think of some special days where the energy demand in your house may be different? What are some other reasons for changes in typical behaviour? Which of these days may be similar to each other?

References

1. Y. Wang, N. Zhang, Y. Tan, T. Hong, D.S. Kirschen, C. Kang, Combining probabilistic load forecasts. IEEE Trans. Smart Grid **10**, 3664–3674 (2019)
2. S.L. Wickramasuriya, G. Athanasopoulos, R.J. Hyndman, Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization. J. Am. Stat. Assoc. **114**, 804–819 (2019)
3. R.J. Hyndman, G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd edn. (OTexts, Melbourne, Australia, 2018). https://Otexts.com/fpp2, Accessed on July 2020
4. Irish Social Science Data Archive, Commission for energy regulation (cer) smart metering project - electricity customer behaviour trial (2012)

5. S. Haben, J. Ward, D.V. Greetham, C. Singleton, P. Grindrod, A new error measure for forecasts of household-level, high resolution electrical energy consumption. Int. J. Forecast. **30**(2), 246–256 (2014). https://doi.org/10.1016/j.ijforecast.2013.08.002, https://www.sciencedirect.com/science/article/pii/S0169207013001386
6. J. Dodge, T. Prewitt, R. Tachet des Combes, E. Odmark, R. Schwartz, E. Strubell, A.S. Luccioni, N.A. Smith, N. DeCario, W. Buchanan, Measuring the carbon intensity of AI in cloud instances, in *2022 ACM Conference on Fairness, Accountability, and Transparency* (2022), pp. 1877–1894
7. Francis X. Diebold, Roberto S. Mariano, Comparing predictive accuracy. J. Bus. Econ. Stat. **13**, 253–263 (1995)
8. David Harvey, Stephen Leybourne, Paul Newbold, Testing the equality of prediction mean squared errors. Int. J. Forecast. **13**, 281–291 (1997)
9. F. Ziel, K. Berk, Multivariate forecasting evaluation: On sensitive and strictly proper scoring rules (2019)
10. D. Ruppert, D.S. Matteson, *Statistics and Data Analysis for Financial Engineering: with R examples*. Springer Texts in Statistics (2015)
11. Daniel Westreich, Sander Greenland, The table 2 fallacy: presenting and interpreting confounder and modifier coefficients. Am. J. Epidemiol. **177**(4), 292–298 (2013)

Chapter 14
Case Study: Low Voltage Demand Forecasts

This chapter demonstrates the practical implementation of short term (day-ahead) forecasts for the application of residential low voltage networks. It is split into two main parts: An in-depth examination of a short term forecasting case study of residential low voltage networks (Sect. 14.2); and a example python code demonstrating how to implement some of the methods and techniques in practice (Sect. 14.3).

The case studies serve to demonstrate how to:

- identify the main challenges when implementing short term forecasts.
- use the techniques from Chap. 6 to analyse the data, and identify important features.
- use the analysis to choose several forecast models (from those presented in Chaps. 9 and 11). This includes both point and probabilistic models.
- test, compare and evaluate the forecasts.

The chapter begins by a short discussion of how to design a forecast trial which will frame the case study that follows later.

14.1 Designing Forecast Trials

It is worth reiterating some of the core elements which should be considered prior to, and while, developing the forecasts. These elements are important to ensure that the model is designed appropriately with minimal bias in methodology, and to ensure that the results are properly tested. The full forecasting procedure is outlined in Chap. 12 and will be followed implicitly throughout. This chapter will focus on the following main considerations.

1. **Initial Experimental Design**: Before plotting the data it is worth sketching out an initial experimental design and audit the available data used to produce and test the forecasts. What type of data is being considered? Is it expected to have seasonalities? What is the resolution of the data, half hourly, every ten minutes?

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_14

Is there sufficient data to produce a informative result? If there is, what is an appropriate split of the data into training, validation and testing sets (see Sect. 8.2)? It is important to think about these questions before analysing the data to prevent introducing bias into the test. Further, once the data has been split, it is advisable to avoid analysing the test set prior to generating forecasts to avoid 'cheating' and seeing the true values before submitting the forecast. A final consideration is to decide on what error measure to use (see Chap. 7). An incorrectly chosen error metric can skew the results, and makes it difficult to evaluate and interpret the results.

2. **Visualisation and Data Analysis**: It is essential to try and learn as much as possible about the underlying features and relationships in the data. In Chap. 5 a number of tools were presented showing how to achieve a better understanding of the data. Simple time series plots can highlight large scale behaviours, scatter plots can identify strong relationships between variables, and autocorrelation plots can highlight periodicities and autoregressive behaviours in the data.
3. **Pre-processing**: A necessary component to the data analysis is data cleansing and pre-processing. Poor quality data can make for misleading analysis and meaningless results. To use a common phrase in machine learning: 'garbage-in garbage out'. Before applying any models, check for anomalous data and missing values as shown in Sect. 6.1.2, and then either replace or remove them from the dataset. The analysis in the previous step can be used to choose the appropriate replacement values.
4. **Model Selection and Training**: As shown in Chaps. 9 and 10 there are a wide range of possible forecast methodologies and choosing the correct models requires utilising the learning from the data analysis, considering the specific requirements for the application, as well as learning from the forecasters own experience. The validation set can be an essential tool for narrowing down the choice of models. It is also vital that appropriate benchmarks (see Sect. 9.1) are selected to help assess the accuracy of the core models. Section 12.2 presents Further criteria which can be considered to help select the initial methods.
5. **Testing and Evaluation**: The trained models must be applied to the unseen testing set. By scoring and comparing the forecast methods with the error measures (See Chap. 7) a better understanding can be forged about what makes some methods more accurate and what are the important (or unimportant) features. This step will allow the forecaster to develop further improvements in future iterations of the models.

Each of the above steps will be illustrated in the following case study.

14.2 Residential Low Voltage Networks

This application considers short term (in this example up to four days ahead) load forecasting for residential low voltage demand on substation feeders and will be used to demonstrate probabilistic (Chap. 11) as well as point forecast methods (Chaps. 9 and 10). The entire section will be based upon the authors research presented in [1].

Here, the term residential low voltage (LV) network demand (or just residential networks) is used to describe the network connected to the secondary substations of the electricity distribution network within a residential area (Sect. 2.1). Although the connected customers will usually be residential they may also consist of small commercial customers such as offices, hairdressers, etc. The demand time series represents the aggregated demand of consumers fed electricity directly from the substation (ignoring any electrical losses in the cables of course). This typically consists of around 40–50 consumers. Furthermore, since these consumers are typically residential, human behaviour tends to be a strong determinant of the demand patterns and hence daily and weekly periodicites are expected. The data considered here will be for 100 residential feeders in the area of Bracknell, a medium sized town in the southeast of England.

At the low voltage, demand is much more volatile than higher voltage due to the low aggregation of consumers (Sect. 2.3) which means probabilistic forecasts can be quite useful for quantifying the uncertainty in the demand.

14.2.1 Initial Experimental Design

The data consists of half-hourly load data for 100 residential low voltage feeders beginning on 20th March 2014 up to the 22nd November 2015 inclusive, a total of 612 days. Typically there are 4–6 feeders which come from a residential low voltage substation and on average there are around 45 consumers per feeder with the largest having 109 residential consumers. A further seven had no available connectivity information due to missing information in the database, so it is not known who is connected. The feeders typically feed residential consumers and 83 of the 100 are purely residential, the others are typically mixes except for one which is known to feed only the landlord lighting of a large office block. The average daily demand across the feeders is approximately 602 kWh and a maximum and minimum daily demand of around 1871 kWh and 107 kWh respectively.

The first decision to be made is how to split the data into testing and training data sets. The data set is reasonably sized, although more data would of course be preferable, especially for residential feeder demand which is expected to have annual seasonality. Ideally, to accurately model annual seasonality several years of data would be available so that the typical year-to-year behaviour could be captured. However, for the purposes of short term forecasts the length of data is sufficient. The final two months were kept over as the out of sample testing set. This consists of 53

days from 1st October 2015 to 22nd November 2015 inclusive. Notice this is just under 9% of the data and is less than the common split of training and testing into a 4:1 ratio (i.e. 20% testing data) as discussed in Sect. 8.1.3. There is a number of reasons for this, firstly this increases the amount of training data whist retaining a reasonable sized testing set and secondly it ensures that the forecasts are made for some of the colder months of the year, which are typically higher in demand and of particular interest to network operators who are concerned about excessive peak demand.

It should be kept in mind, that since the training is 559 days and only 1.5 years long there may be some limitations in capturing annual seasonalities and therefore the methods here cannot be reliably extended to medium term (one month to a year ahead) or longer term (over 1 year ahead) forecasts.

Hourly temperature forecast data and observed temperature data are also available for the same time period. The forecasts all begin at 7 AM each day and then produce hourly forecasts up to a horizon of 4 days ahead (96 h ahead). This means temperature effects can also be studied but since the forecast origin (where the forecast starts from) is limited to 7 AM each day, they must be treated with caution when using them as inputs to the forecast models. In particular, it would be expected that forecasts become slightly less accurate the further ahead they forecast which means that, e.g. the four hour ahead temperature forecasts (i.e. the ones at 11 AM) will be more accurate than the forecasts five or more hours ahead (i.e. those from noon onwards).

With these datasets several situations can be tested

1. **Case 1**: How does the accuracy of a forecast model change with horizon from 1 h ahead to 96 h (four days) ahead?
2. **Case 2**: Are all residential LV feeders forecast with similar accuracy? If not what are some of the distinguishing factors between them?
3. **Case 3**: What is the effect of including temperature within a forecast model for residential LV network demand?

To allow comparison between models with and without temperature forecast inputs, all forecasts will generate hourly four day ahead forecasts starting at the forecast origin of 7AM of each day of the testing set. This requires aggregating the half-hourly demand time series up to the hourly resolution (see Sect. 6.1.4) to facilitate using temperature data as an input to the models.

To allow comparisons between the different forecasts, some forecast error measures need to be chosen as presented in Chap. 7. To allow comparison between different size feeders, relative measures which don't depend on the size of the feeder (i.e. typically demand magnitude) are required. MAPE is a common relative measure used for demand forecast but since it can be conflated by small values, a modified version of the MAE (see Eq. (7.46) in Chap. 7) is also used which takes the usual MAE but is scaled by the average hourly load of each feeder over the final year of the training data. This will be referred to as the Relative MAE or RMAE. Since the experiment will also include probabilistic forecasts, probabilistic scoring functions will also be required. In this experiment the continuous ranked probability score

(CRPS) is used (see Eq. (7.50) in Chap. 7). The CRPS error for each feeder is also divided by the average hourly demand for that feeder to produce the Relative CRPS or RCRPS.

14.2.2 Data Analysis

As defined here, residential LV feeders are predominantly connected to residential households but may also connect to a small number of shops, offices, churches, schools and other small-to-medium enterprises. For these reasons it would be expected that demand patterns are largely driven by human behaviour and hence contain strong daily, weekly and annual seasonalities. An example of the demand for a few of the feeders is shown in Fig. 14.1 for different numbers of consumers connected (labelled with the variable *NumMpans*) and different numbers of residential consumers (labelled *NumRes*). The time series plots identify several features. Firstly that there is a wide variety of behaviours, even between the two purely residential feeders (Labelled Feeder 4 and 15 in the plot) with similar numbers of connected consumers (44 and 42). Although they both exhibit annual seasonality with larger demands over the Winter period, there is large periods of low demand during the Christmas and Easter holidays for Feeder 15 but not for Feeder 4. Although this won't be considered in this work, it does suggest holiday periods should be treated as special inputs to the model and this could be an important extension to the more general models presented here (see Sect. 13.6.2). Another important distinction is between the largely residential Feeders (4, 10 and 15) and Feeder 23 which is connected to a single commercial consumer. The commercial consumer doesn't have strong annual seasonalities, and in contrast to the purely residential feeders, has relatively low demand during the Winter period. These time series have identified two important properties of these time series, firstly annual seasonality is an important feature to include in the models, and secondly there is a wide difference between different feeders which suggests there may not be a one-size-fits-all model which will be accurate for all feeders.

The time series plots have identified annual seasonality as an important feature to include in the forecast models. Other inter-annual seasonalities can be identified by considering the autocorrelation function (ACF) plots (see Sect. 6.2.2). In fact, in each of the ACF plots there are relatively large spikes at lags of a day (a lag of 24 h) and a week (lags of 196) and multiples of these. Of these, the weekly periodicities are the strongest autocorrelations as expected. Since there are 100 feeders it is difficult to consider all of their respective autocorrelation plots, instead particularly important lags can be given special focus. The weekly seasonality information is considered in Fig. 14.2, by showing the autocorrelations at lag 196 as a function of the size of each feeder (average daily demand in kilowatthours (kWh)). This shows that the strongest weekly autocorrelations are associated to the largest feeders. One explanation for this is that feeders with larger demands consist of aggregations of larger numbers of residential consumers, increasing the prominent regularities in weekly behaviour.

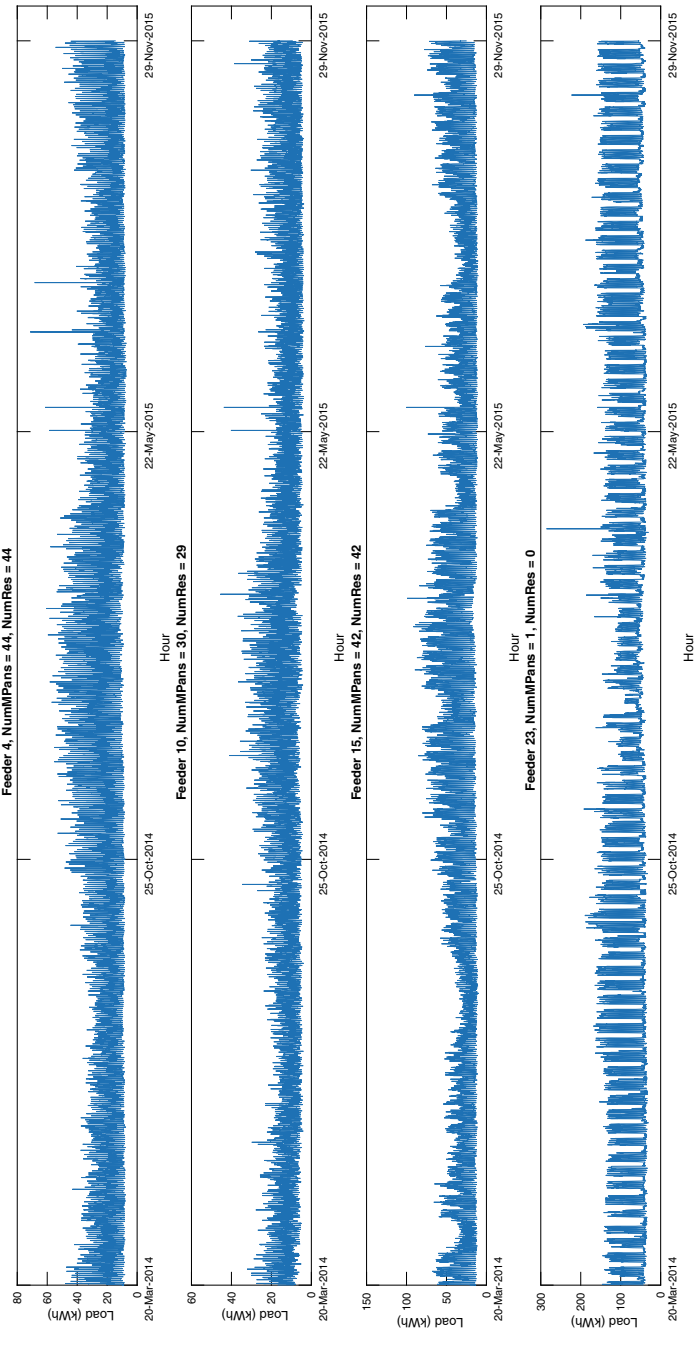


Fig. 14.1 Example of demand time series for four different residential LV feeders. Shown in the title to each is the number of MPANs (consumers), NumMPans, connected to the feeders and also how many of them are residential, NumRes

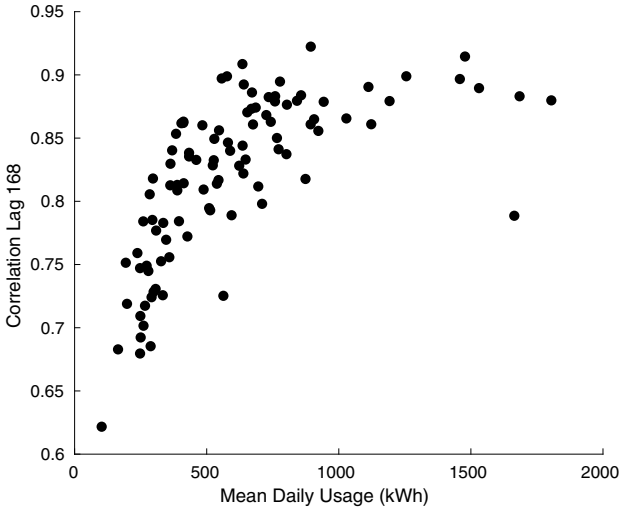


Fig. 14.2 Autocorrelation at lag 168 (weekly seasonal correlation) for all 100 feeders against the mean daily demand. Reprinted from [1] with permission from Elsevier

Given these weekly periodicities what does the average weekly demand look like for an LV feeder? Figure 14.3 shows an example of the normalised average weekly demand for three feeders each with forty consumers connected, two of which are purely residential whilst one consists of a single commercial consumer and 39 residential consumers. The data has been normalised (i.e. divided by the average weekly demand) so that the distribution of demand over the week for different feeders can be compared without being obscured by the magnitude of the demand. The plot indicates some important features:

- Daily and weekly seasonalities are quite prominent.
- Often weekdays (Monday to Friday) are very similar but Saturday and Sunday may be different from weekdays and from each other. This important observations suggests that different days of the week should be treated differently in the models (see later in Sect. 14.2.3).
- The feeder with the single commercial consumer has different patterns and distribution of demand compared to the purely residential feeders. During the week the mixed feeder has its peak demand during the day which indicates that the commercial consumer is likely dominating the demand. In contrast at the weekend, the demand more closely resembles a residential feeder with the peak in the evening which indicates the commercial consumer is no longer dominant and is likely non-operational or has reduced operation during the weekend.

The final observation suggests that there may not be a strong connection between one feeder and another and therefore there is less scope for transferring learning (Sect. 13.4) over low voltage feeders. This will not be tested. Although there is not many anomalous values in the hourly data there are still some missing values due to

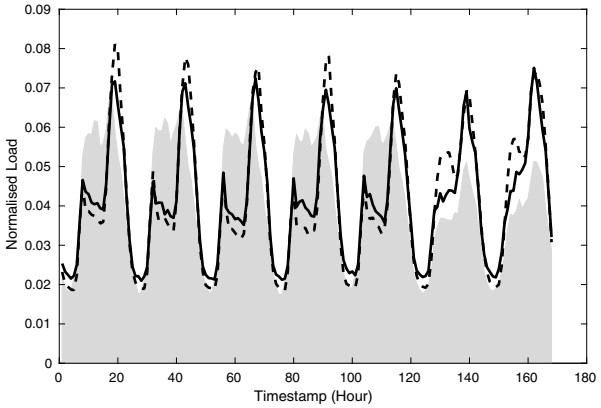


Fig. 14.3 The average normalised weekly demand for three feeders with forty consumers. The shaded profile represents a residential feeder which includes a single commercial consumers, whereas the other profiles (lines) represent feeders with only residential consumers. The profiles start on Monday

communication and sensor faults. Recall from Sect. 6.1 that anomalous/missing values can either be retained and then ignored by the model in the training phase or they can be replaced with informed estimates. The latter simplifies the training process and hence was the chosen option here. The above analysis shows there is strong evidence of weekly periodicities and autocorrelations, and this can be exploited to produce sensible values from which to impute missing data as described in Sect. 6.1.2. Each missing value is replaced with an average of the adjacent hourly demand and the value at the same hour of the previous two weeks. This ensures a final value which is weighted between the magnitude of the weekly seasonality and the locally recent demand.

Having now identified important autoregressive and time period effects in the data it is also worth considering external or exogenous variables. Temperature is often associated with load, for example, in colder temperatures more heating is required and therefore more energy is used [2, 3]. Fortunately for this trial, weather data is readily available from a nearby weather station approximately 16 km from the centre of Bracknell.

The relationship between the demand (in kWh) versus temperature (day ahead) forecasts (in degrees C) is shown in Fig. 14.4, for one of the feeders (which happens to have a particularly strong correlation with temperature) for four different time periods of the day. Also included are the lines of best fit (see Sect. 9.3) and the adjusted coefficient of determination (see Eq. (6.39) in Sect. 8.2.2) which describes how strongly the line explains the relationship. There are a few main observations from this plot

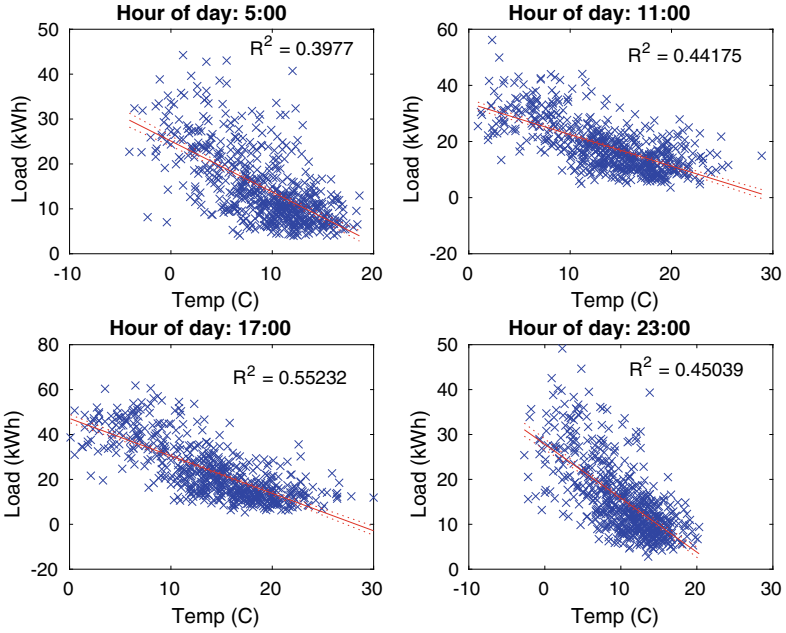


Fig. 14.4 Load versus temperature for a particular feeder for four different hours of the day. Linear fits (red lines) and adjusted R^2 values are also shown

- There is often a negative correlation between demand and temperature. The colder the temperature the more demand required. This is likely due to more electric heating and lighting being required.
- Different hours of the day have different correlations and have different adjusted coefficients of determination. Since heating behaviour is likely driven by whether the house is occupied this explains why some hours are more strongly related to temperature than others.
- Although this feeder has one of the strongest linear correlations with temperature it still isn't particularly strong ($R^2 < 0.56$).

As would be expected, the accuracy of the temperature forecasts reduce with increasing horizon. For the period 31st March 2014 to 28th Nov 2015 the day ahead temperature forecasts have a MAPE of 11.85% this reduces for every subsequent daily horizon up to 23.80% MAPE for four days ahead (i.e. between 73 and 96 h ahead) forecasts. If temperature is one of the most important factors for demand then one would expect that the accuracy of the models would decrease with increasing forecast horizons.

This section has highlighted some features which may be important and will be tested within the forecast models introduced in the next section. Of course, further analysis and techniques could be applied, such as those introduced in Sect. 6.2, to find further features (for example the possible holiday effects as suggested by the

time series plots in Fig. 14.1). However for the purpose of generating forecasts which capture the main features of the demand, the current features should suffice.

14.2.3 Model Selection

The features identified in the data analysis are not only important as inputs to the forecasts models but can be used to inform the choice of models themselves. The forecast models described here are all based on those presented in Chap. 9. Recall the aim is to generate accurate point and probabilistic, four day-ahead forecasts. By comparing the forecast models, insights can be gained on which models are most accurate, but also identify some of the more important features for describing LV level demand. Throughout the section $L_1, L_2, \ldots,$ will denote the demand time series with L_t the demand at time step t. For the probabilistic forecasts 99 quantiles will be generated for each time step in the forecast horizon.

To begin, four basic benchmarks will be defined in order to properly assess the inputs and compare to the main forecast models. As described in Sect. 8.1.1 there are several categories of benchmark models. Since there is no state-of-the-art available the focus will be on simple and common benchmarks.

Benchmark 1: Naïve Seasonal Persistence (LW)
As described in Sect. 9.1, for time series with seasonalities, a simple seasonal persistance model can be an effective choice of benchmark. For a series which has a seasonal period of s_1 this is defined as

$$\hat{L}_{t+k} = L_{t+k-s_1}. \tag{14.1}$$

Given that a weekly period is one of strongest auto-correlations in the LV demand time series (Recall Fig. 14.2) $s_1 = 168$ is chosen. This model will be called **LW** to indicate that it is the **L**ast-**W**eek-as-this-week persistence forecast.

Benchmark 2: Simple Moving Average (SMA)
The seasonal persistence forecast captures some of the seasonality in the demand time series but as shown in Sect. 9.1 it suffers from the natural variations in the demand from week-to-week. In order to smooth out these deviations the simple moving average was proposed. This simply takes the average over the same period of the week for the previous p weeks. It is defined as

$$\hat{L}_{t+k} = \frac{1}{p}\sum_{i=1}^{p} L_{t+k-i\times s_1} \tag{14.2}$$

where $s_1 = 168$ is again the weekly period for hourly data. The main parameter p is to be found over the training period and $p = 5$ is found to be the optimal. This

model retains the seasonality of the **LW** method but does not suffer from the random weekly variations. This model is denoted **SMA** or **SMA**-pW to indicate the p weeks of data used in the average.

Benchmark 3: Empirical CDF
A simple probabilistic model can be generated by estimating the distribution of the historical load data. For each period of the week define an empirical distribution function (see Sect. 3.4) using all the load data from the same time period over the final year (using only one year reduces any potential seasonal biases, i.e. selecting one month more than others) of the historical data. In other words to estimate the distribution of points for 2 PM on a Monday, for a particular feeder select all load values from 2 pm on a Monday. From the resultant empirical distribution, quantiles can the be selected. The median of this distribution can also be chosen as the corresponding point estimate. For more details on an empirical distribution see Sect. 3.4.

Benchmark 4: Linear Seasonal Trend Model (ST)
The previous benchmarks focus on the weekly seasonal behaviour. A multiple linear model as described in Sect. 9.3 is a relatively simple benchmark which nevertheless can model more sophisticated relationships. Motivated by the analysis in Sect. 14.2.2 a linear model is constructed to produce day ahead forecasts (three other equivalent models are developed to achieve two, three and four day ahead forecasts respectively) which takes into account the annual, weekly and daily effects. One of the easiest ways to include seasonal behaviours is to use sine and cosine functions as basis function (see Sect. 6.2.5), e.g.

$$\sum_{k=1}^{H}\left(a_k + b_k\eta(t) + \sum_{p=1}^{P}(c_{k,p})\sin\left(\frac{2\pi p\eta(t)}{365}\right) + (d_{k,p})\cos\left(\frac{2\pi p\eta(t)}{365}\right)\right), \tag{14.3}$$

where, H is the number of time steps in a day (24 for the hourly data here), $\eta(t) = \left\lfloor \frac{t}{H} \right\rfloor + 1$, is an identifier for the day of the trial (with day 1 the first day of the trial set: 20th March 2014). The function $\lfloor x \rfloor$ here is the *floor function* and rounds down the number to the largest integer less than or equal to x, so for example, $\lfloor 2.1 \rfloor = 2$, $\lfloor -5.4 \rfloor = -6$, and $\lfloor 12 \rfloor = 12$. This simple model is a good start for describing the annual seasonality but it does not take into account the daily seasonality which was observed in the data analysis. The model can therefore be updated using dummy variables (see Sect. 9.3), e.g.

$$\sum_{k=1}^{H}\mathcal{D}_k(t)\left(a_k + b_k\eta(t) + \sum_{p=1}^{P}(c_{k,p})\sin\left(\frac{2\pi p\eta(t)}{365}\right) + (d_{k,p})\cos\left(\frac{2\pi p\eta(t)}{365}\right)\right), \tag{14.4}$$

where $\mathcal{D}_k(t)$ is the daily effects dummy variable and is defined by

$$\mathcal{D}_j(t) = \begin{cases} 1, & \text{if } j = t + Hk, \text{for some integer k} \\ 0, & \text{otherwise,} \end{cases}$$

This new model captures annual seasonality by effectively producing 24 models, one model for each hour of the day. The data analysis also showed that there is strong weekly periodicities in the LV demand time series, hence one more adjustment can be applied to give the final model

$$\hat{L}_t = \sum_{k=1}^{H} \mathcal{D}_k(t) \left(a_k + b_k \eta(t) + \sum_{p=1}^{P} (c_{k,p}) \sin\left(\frac{2\pi p \eta(t)}{365}\right) + (d_{k,p}) \cos\left(\frac{2\pi p \eta(t)}{365}\right) \right) + \sum_{l=1}^{7H} f_l \mathcal{W}_l(t), \tag{14.5}$$

where $\mathcal{W}_l(t)$ a weekly dummy variable defined by

$$\mathcal{W}_j(t) = \begin{cases} 1, & \text{if } j = t + 7Hk, \text{for some integer k} \\ 0, & \text{otherwise,} \end{cases}$$

The $\sum_{l=1}^{7H} f_l \mathcal{W}_l(t)$ term adjusts each daily hour model depending on the hour of the week and this allows the modelling of different behaviours on the weekends and the weekdays and can capture the features which were observed in Fig. 14.3. One of the hyperparameters to choose is the number of seasonal terms P. For simplicity and to avoid overfitting (see Sect. 8.1.2) this is set to $P = 3$ (Although a validation set as described in Sect. 8.1.3 could be used to properly choose this).

Although the formula looks relatively complicated the model is actually quite simple and is still a multiple linear model. Due to the presence of the dummy variables, there is in fact 168 separate models for each hour of the week for the day ahead forecasts.

The model can be easily extended to include further inputs. In this case the nonlinear relationship between temperature, T_t, at time t can be included by adding a simple polynomial of the temperature. In this case, since the relationship between demand and temperature is not too strongly nonlinear a simple cubic is considered: $\alpha_1 T_t + \alpha_2 T_t^2 + \alpha_3 T_t^3$, where α_1, α_2, α_3 are the coefficients for the temperature components of the multiple linear model.

Any linear model can be easily extended to a univariate probabilistic model by using the model within a quantile regression for each quantile as described in Sect. 11.4. An example of a quantile regression fit on the 6 PM data in the training set for a specific feeder is shown in Fig. 14.5 for the 10, 50 and 90 percentiles. Notice the main annual seasonality captured by the model and the small increases in demand that occur on weekends. The variation in the demand may not appear to be smooth

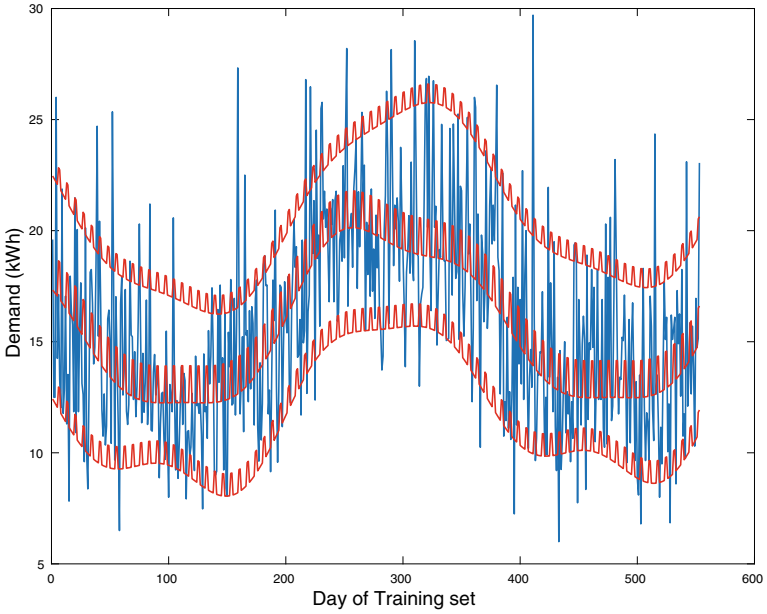


Fig. 14.5 Quantile regression fit of the simple linear model for the demand at 6PM on a specific feeder for the 10, 50 and 90 percentiles

and it may be tempting to try and force the data to have a more simple annual seasonal shape, however it is important to not try and second-guess the patterns in the data as the only true assessment of the model will be on the test set. Besides, since this is a benchmark model it is not necessary to try and make the model perfect. The variation in the annual seasonality could also be due to the small numbers of complete years in the training set. It is likely to be smoother if several years of data were available.

The above benchmarks include a number of important properties that have been discovered by the data analysis, including weather variables, and daily, weekly and annual periodicities. However, they do not include any autoregressive effects. The benchmarks will therefore be compared to a number of slightly more sophisticated models of demand which will include this feature.

Main Model 1: Seasonal Exponential Smoothing (HWT)

The double seasonal exponential smoothing method (or Holt-Winter-Taylor (HWT) method after its creators) described in Sect. 9.2 is well suited for LV demand forecasting due to its ability to incorporate two levels of seasonality and incorporate localised autoregressive behaviour. In this case the two periods parameters used are $s_1 = 24$, and $s_2 = 168$, since daily and weekly seasonalities respectively have been shown by the data analysis to be two of the most important components of the demand time series. In the HWT model, recent data contributes more to the final forecast than

older data, this means that it also implicitly models the seasonal component since the overall level of the forecast is based on locally recent information.

Once the parameters are trained for this model, a probabilistic forecast can be generated by bootstrapping the 1-step ahead residuals as described in Sect. 11.6.1. As with the other models, the median is used as the point forecast model.

Main Model 2: Auto-Regressive Models (ARWD, ARWDY)
Another way to incorporate autoregressive information is to generate an AR model on the residuals of a sensible forecast model. This is the same process as described in Sect. 7.5 which describes autoregressive correction for improving forecast models. More generally, any forecast model μ_t which estimates a time series L_t can be improved using this method if there is autocorrelation structure remaining in the residual time series $r_t = L_t - \mu_t$. In this case an autoregressive model is applied to the residual time series

$$r_k = \sum_{k=1}^{p} \phi_k r_{t-k} + \epsilon_t, \tag{14.6}$$

where ϵ_t is the error, and the most appropriate order p can be found by, e.g. calculating the Akaike Information Criterion (AIC), or other information criterion (see Sect. 8.2.2) for a range of different values $p = 1, \ldots, p_{\max}$.

The choice of underlying forecast model μ_t is very general. As the focus is on testing the autoregressive effects, the baseline models will be kept relatively simple.

The analysis showed the importance of weekly seasonality, hence the first choice of forecast model is a simple linear model

$$\mu_t = \sum_{j=1}^{7H} \beta_j \mathcal{W}_j(t). \tag{14.7}$$

where, $H = 24\,\text{h}$ and $\mathcal{W}_j(t)$ is the period of the week dummy variable

$$\mathcal{W}_j(t) = \begin{cases} 1, & \text{if } j = t + 7Hk, \text{for some integer k} \\ 0, & \text{otherwise,} \end{cases}$$

as used in the ST benchmark forecast. The parameters to train are the coefficients β_j and these are estimated by simple ordinary least squares (OLS) (see Sect. 8.2) over the initial prior year of historical loads. From this model the residuals are calculated and the estimates from the residual model, $\hat{r}_t$ is added to μ_t to give the final forecast

$$\hat{L}_t = \mu_t + \hat{r}_t. \tag{14.8}$$

This model will be denoted **ARWD** to signify autoregressive model with weekday mean.

A second forecast model is also consider which adds onto the ARWD mean model by including a term for annual seasonality seen in the data analysis, this is given by

$$\mu_t = \sum_{j=1}^{7H} \beta_j \mathcal{W}_j(t) + \sum_{k=1}^{K} \alpha_{1,k} \sin(2\pi tk/A) + \alpha_{2,k} \cos(2\pi tk/A) \tag{14.9}$$

with parameters β_j and $\alpha_{j,k}$, and $A = 365H$ as annual seasonality. The annual seasonality is modelled by a Fourier approximation of order K which is fixed to $K = 2$ to reduce the complexity. The dummy variable $\mathcal{W}_j(t)$ is as in Eq. (14.7). As with the **ARWD** model the μ_k is estimated by OLS between the model and the training data. This model is used to calculate a new residual time series which is then also trained by OLS and is added to the mean model in Eq. (14.9) to give the final forecast $\hat{L}_t$ given by

$$\hat{L}_t = \mu_t + \hat{r}_t. \tag{14.10}$$

as with the ARWD model. This model is denoted **ARWDY** with the Y signifying the yearly periodicities included through the Fourier terms. Note that separate ARWD/ARWDY models are used depending on whether the forecasts are one, two, three or four days ahead.

Notice the subtle differences between the modelling of the seasonalities in this model versus the ST model. In the ARWDY model the periodicities are not separated for different periods of the day. If there is significant differences in how seasonalities effect different times of day then perhaps the ST will perform slightly better. However, the ARWDY also includes autoregressive effects which now incorporates more interdependencies between hours of the day. As with the ST methods the weather effects are included by adding linear terms to the mean equations.

These regression models will serve as point forecasts. To extend them to probabilistic forecasts the slightly more sophisticate GARCH type model, described in Sect. 11.6.2 will be considered. In this the variance itself will be modelled by considering the final model residuals $\epsilon_t = \hat{L}_t - L_t$ which are assumed to have the form $\epsilon_t = \sigma_t Z_t$ where σ_t is the conditional standard deviation of ϵ_t and $(Z_t)_{t_e \mathbb{Z}}$ is an independent identically distributed random variable with $\mathbb{E}(Z_t) = 0$ with $\mathbb{V}ar(Z_t) = 1$. The method is described in detail in Sect. 11.6.2 and requires a model for the standard deviation. Since the variation in demand is likely to be correlated with the size of the demand (larger demands have more variation) the same mean model for the point forecast will be used for the standard deviation. For example in the case of ARWDY the model will be

$$\sigma_t = \sum_{j=1}^{7H} \tilde{\beta}_j \mathcal{W}_j(t) + \sum_{k=1}^{K} \tilde{\alpha}_{1,k} \sin(2\pi tk/A) + \tilde{\alpha}_{2,k} \cos(2\pi tk/A). \tag{14.11}$$

where the coefficients $\tilde{\beta}_j$, $\tilde{\alpha}_{1,k}$, $\tilde{\alpha}_{2,k}$ are to be found and the tilde over the parameters is used to distinguish them from the coefficients from the mean model. Once the

standard deviation σ_t and mean $\hat{L}_t$ are found the bootstrap procedure (as introduced in Sect. 11.6.2) can be employed to generate an empirical distribution (see Sect. 3.4) for each time step in the forecast horizon. To do this perform the following operation, for the forecast starting at time step $t = N + 1$

1. Draw a sample $\hat{Z}$, from empirical distribution of the random variables Z_t.
2. Scale the variable with the standard deviation to give a residual $\epsilon_{N+1} = \sigma_{N+1}\hat{Z}$.
3. Add this to the mean forecast $\tilde{L}_{N+1} = \hat{L}_{N+1} + \epsilon_{N+1}$.
4. Use this current value within the forecast inputs to generate the forecast for the next time step $\hat{L}_{N+2}$.
5. Repeat this process until forecasts have been generated for all time steps in the forecast horizon.

The bootstraps generate a multivariate probabilistic forecast, but these can be transformed into a univariate probabilistic forecasts for each time step by fitting a distribution or calculating the empirical quantiles at each time step from the generated points (Sect. 3.4).

14.2.4 Testing and Evaluation

A diverse selection of models have been described in Sect. 14.2.3. Since they all have slightly different structures and use different features as inputs they can be used to test a variety of hypothesis and assumptions. As discussed in Sect. 14.2.1 there are the following main questions that can be analysed via the errors on the test set:

1. What is the effect of temperature?
2. How does accuracy change with forecast horizon?
3. How does accuracy change for different feeders?
4. Which features are the most important for an accurate forecast?

The models are all trained on a training set which covers the dates from 20th March 2014 to the 30th September 2015 inclusive and rolling four day-ahead forecasts have been generated for the 53 day testing set starting 1st October 2015. Notice that no validation set has been used in this case (Sect. 8.1.3) this is for two reasons, firstly all the models use a relatively small number of parameters and hence have low chance of being over fitted to the data, hence the model selection step on the validation set has been skipped for this trial. Secondly, although there is around a year and a half of data this is not particularly large for a data set with annual seasonality, hence a validation set would require splitting the data further and would potential reduce the reliability of the results on the test set. Hence the larger training set increases the chance of properly training the model parameters.

To begin consider a comparison of the forecast models generated in Sect. 14.2.3. Table 14.1 shows the average score over all four day-ahead forecasts for the entire 53 day test period for all 100 feeders using the MAPE, RMAE and RCRPS measures (Sect. 14.2).

First consider the point forecasts (MAPE and RMAE scores). The main observations from the results are as follows:

- Of the benchmark models the simple persistence model (LW) is the least accurate whereas the moving average using 5 weeks (SMA-5W) and the simple seasonal regression (ST) are the best performing. This suggests averaging the weekly historical data is useful for producing more accurate models than using a simple last-week-as-this-week value.
- Of the two best benchmarks the ST forecast is slighlty more accurate than SMA-5W (only 2% lower MAPE), which suggest including annual seasonality can be beneficial but only slightly.
- The main models (ARWD, ARWDY and HWT) are more accurate than the benchmarks. These models all have autoregressive features and hence suggests there are important temporal interdependencies in the demand time series.
- The ARWD and ARWDY methods are slightly better than the HWT method. One of the main differences between these models is that HWT only explicitly uses the previous lag (although the previous lags are included implicitly as smoothed historical terms) and thus suggests whilst the most recent previous demand is an important indicator of the demand, older lags are also important for determining the current demand.
- There is very little difference between the ARWD and ARWDY forecast models, with the ARWD performing slightly better, on average, across all measures. Thus an explicit seasonality term has limited importance in the forecast accuracy compared to the autoregressive term.

The probabilistic forecasts in fact show the same ranking of the methods as the MAPE and RMAE, with ARWD ranked as the most accurate method, followed by ARWDY, then HWT, then ST and then the Empirical method. This is an encouraging result since it suggests that the accuracy of the point forecasts may be indicative of the accuracy of the probabilistic forecasts. Probabilistic methods are typically more expensive to train and therefore if the point forecasts can be used to rank the probabilistic forecasts this significantly reduces computational cost of identifying and training these methods. However caution must be exercised as this is only an empirical observation and hasn't been established theoretically.

Temperature Effect

Table 14.1 does not consider the influence on temperature on the forecast accuracy of residential LV network demand. As shown in Sect. 14.2.2, there seems to be a relatively low correlation between temperature and the demand despite the fact that the temperature is often seen as strongly connected to electricity demand due to its obvious connection to heating and cooling behaviours. The MAPEs for particular point forecasts that use and don't use temperature forecasts are shown in Table 14.2. The table suggests that the weather is not a strong driver of the demand. The benchmark method, ST, does improve slightly, however the most accurate models, ARWD and ARWDY, both become less accurate when temperature is included. Why would this be the case?

Table 14.1 MAPEs, RMAEs and RCRPSs for all forecast methods over all 4 day-ahead horizons for the entire 53 day test period for all 100 feeders. The lowest errors for each score are highlighted in bold. Reprinted from [1] with permission from Elsevier

Method	Error scores %		
	MAPE	RMAE	RCRPS
LW	18.67	18.93	–
SMA-5W	15.73	16.77	–
Empirical	16.19	16.96	12.62
ST	15.42	15.42	10.97
HWT	14.84	15.01	11.06
ARWD	**14.65**	**14.67**	**10.32**
ARWDY	**14.64**	**14.80**	**10.44**

Table 14.2 MAPEs for the methods showing the effect of including temperature forecast data in a selection of methods. Reprinted from [1] with permission from Elsevier

Method	None	Temperature
ARWD	14.65	16.94
ARWDY	14.6	15.16
ST	15.42	15.16

To further investigate the effect of including temperature as an input, consider what happens when the MAPE scores are split according to forecast horizon (at the daily resolution) as shown in Table 14.3. For comparison, the MAPEs of the temperature forecasts themselves are included. The temperature forecast drops in accuracy by more than 80% from one day-ahead to four days-ahead. Thus if there is a strong dependence on temperature it would be expected that the models trained using the temperature would also drop off in accuracy at a comparable rate. In fact the ST demand forecast accuracy changes very slightly and the ARWDY and ARWD demand forecasts drop in accuracy only by 4.3% and 5.6%. Further experiments can be included, for example including lagged temperature values in the demand forecast models. However, in all cases the results are the same: temperature doesn't appear to have a strong effect on the demand forecast accuracy. If we examine the individual feeders the temperature is only shown to improve the forecasts of 19 out of 100 feeders, and in all cases the MAPEs do not improve by more than 4%.

The inclusion of the temperature also doesn't improve the probabilistic forecasts. In Fig. 14.6 shows the reliability plot (See Chap. 7) for the probabilistic forecasts generated using the ARWDY model using no temperature (solid dots), using actual temperature (unfilled dots) and using the forecast of the temperature (crosses). The diagonal line shows the expected line if the quantiles generated from the model (i.e. the predicted spread of the data) matched the empirical quantiles. It is clear that the model not using any temperature is closest to this line hence showing that including the temperature (whether forecast or actual) does not in fact improve the forecast.

Table 14.3 MAPE Scores for different day ahead horizons for a selection of methods which use the forecast temperature values as inputs. Also for comparison is the average MAPE for the temperature forecast themselves. Reprinted from [1] with permission from Elsevier

Method	MAPE			
	Day 1	Day 2	Day 3	Day 4
ARWD	16.51	17.26	17.12	16.89
ARWDY	14.75	15.11	15.31	15.46
ST	15.12	15.21	15.16	15.16
Temperature	8.98	10.57	13.46	16.47

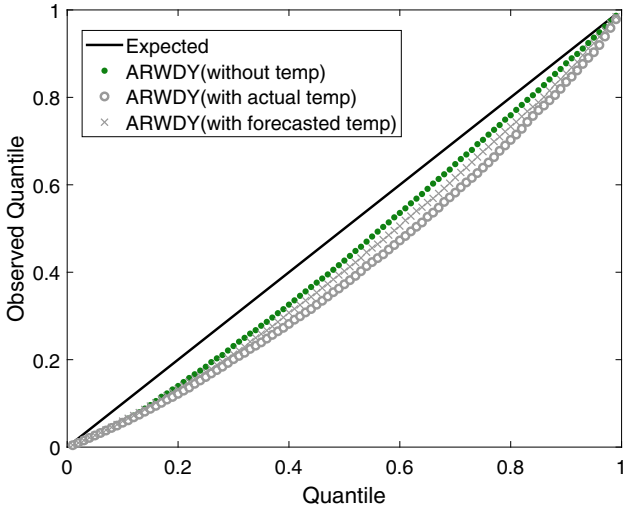


Fig. 14.6 Reliability plot for the ARWDY forecast for different temperature inputs. Reprinted from [1] with permission from Elsevier

There could be many explanations for the lack of influence of the temperature data, for example, the seasonality could be the main driver of demand and the perceived correlation between demand and temperature is actual only due to the collinearity between seasonality and temperature. In fact seasonality may be a confounding variable in this situation (Sect. 13.6.1). In addition, it could be that much of the heating for the consumers on these feeders use gas instead of electrical boilers and hence temperature will only have a minimal effect. However, regardless of the reason, for a forecaster the results for this particular data and test set indicate that temperature is not a particularly important input for these forecast models. This result also highlights an important lesson: even if there is strong intuitive reasons for a explanatory feature to be important, it does not necessarily translate to importance for the forecast model. Going forward with the analysis the temperature is not going to be considered any further.

Forecast Accuracy and Horizon

Most forecasts become less accurate the further ahead they predict (Sect. 7.3). One reason for this is that often recent values can be strong indicators of the near future demand. Further into the future, the most recent observations are much older and hence not as useful at making accurate predictions. If a forecast can retain its accuracy for longer time horizons into the future then they can be useful for more longer term planning. For storage applications this means longer term plans can be made for when to charge and discharge the device.

Table 14.4 shows the forecast accuracy (MAPE) for selected methods as a function of days ahead. In other words the 'Day 1' column means the average error from forecasting between 1 and 24 h ahead, the 'Day 2' column indicates the average error from forecasting between 25 and 48 h ahead etc. For ARWDY the MAPE errors only increase from 14.34% to 14.87%, i.e. a 3.7% increase. In fact there is only a small drop in the accuracy for any of these models and this indicates that the models can offer similar accuracy for estimating tomorrows demand as they do for four days time. Computationally speaking, this can be quite advantageous as it can reduce the cost of model retraining with minimal impact on the forecast accuracy.

How does the accuracy change at the hourly level? This time consider the probabilistic forecasts (recall the results are qualitatively similar whether the point or probabilistic forecasts are considered). Figure 14.7 shows the relative CRPS errors as a function of hourly horizon for selected methods. Recall, the forecasts all begin at 7 AM of each day and hence the first horizon point corresponds to the period 8−9 AM. The following observations can be made

- The intraday shape of the errors are similar. Hence the period of the day is a major indicator of the accuracy of the forecasts. Notice that the areas of highest errors (largest CRPS) correspond to periods typically associated to high demand (and hence high volatility), e.g. around the evening period. The overnight periods (around 10PM until 5AM) have the lowest errors. These are typically periods of low activity. This supports using a GARCH type model where the volatility (standard deviation) is correlated with the average demand.
- Although the shapes are similar, there is a small trend of increasing error from one day to the next.
- Different forecasts are more accurate for different periods of the day. For example, although ARWD is generally the most accurate model, for some evening periods

Table 14.4 MAPE Scores for each method over each day ahead horizon. Reprinted from [1] with permission from Elsevier

Method	MAPE			
	Day 1	Day 2	Day 3	Day 4
HWT	14.56	14.83	14.95	15.04
ARWDY	14.34	14.59	14.75	14.87
ST	15.36	15.41	15.44	15.49

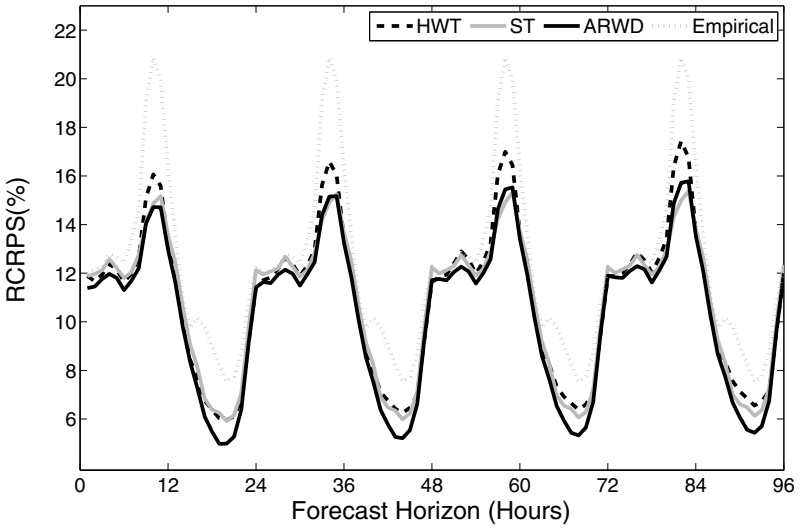


Fig. 14.7 Plot of average Normalised CRPS for selected methods for horizons from 1 h to 96. Reprinted from [1] with permission from Elsevier

the ST benchmark is actually more accurate. Hence one potential way of generating a more accurate forecast could be to take a combination of several models (see Sect. 13.1).

Forecast Accuracy and Feeder Size

The final piece of analysis concerns comparing the accuracy across different feeders. As previously mentioned the feeders come in all shapes and sizes. Some have up to 109 customers connected whereas some have as few as one. Further there is a diversity of the types of customers. Some are commercial, but most are domestic and even amongst the domestic customers there is a wide diversity in their behaviours. Are you like your neighbour? In addition, there are other loads that are not monitored: street lights, elevators, cameras, landlord lighting etc. which all contribute to the load shape and diversity.

Figure 14.8 shows the MAPEs for the ARWDY model for each individual feeder as a function of the average daily demand. Each point represents a different feeder and some of them have been given different icons to signify different categories of feeders. An instant observation is that 88 of the feeders closely fit a power law curve (the bold curve in the plot). The smaller feeders tend to be more volatile and therefore have larger relative error values. In contrast the larger feeders have smaller errors. This can be explained largely by the law of large numbers. The increased size of the feeder corresponds to larger numbers of customers connected to the feeders, which result in smoother and more regular demands that are easier to forecast.

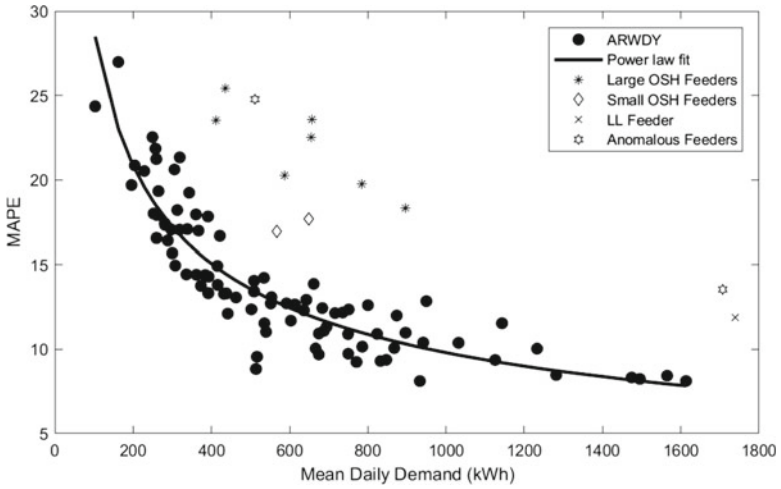


Fig. 14.8 Scatter plot of the relationship between MAPE and mean daily demand for two different forecasting methods. Feeders which apparently have overnight storage heaters or have usually larger errors have been labelled separately as OSH and anomalous respectively. Also shown is a power law fit to the non-OSH/anomalous feeders. Reprinted from [1] with permission from Elsevier

There are twelve feeders which don't fit the power law relationship. This unusual behaviour should prompt the investigator to try and better understand why these feeders don't fit the general trend. By looking at the average profile of seven feeders an immediate observation is that the feeders have unusually large overnight demand. This prompted a consideration of what type of consumers were on these feeders. In fact it was found that 75−85% of customers on each of these seven feeders had large numbers of overnight storage heaters (OSH). Overnight storage heaters are heaters which use energy during the night to store up heat and then release this energy during the day. These are labelled "Large OSH Feeders" in the plot. Further to this, two other feeders had smaller overnight demands, their feeders had 62% and 75% of their customers with OSHs, these are labelled "Small OSH Feeders". This in itself isn't enough to explain these feeders not obeying the power law. It also must be confirmed that none of the other 88 feeders also have large proportions (greater than 60%) of OSH or profiles which have large overnight demands. In fact this was found to be the case with these feeders and hence strongly suggests the presence of a high proportion of OSH effects the accuracy of the chosen forecast models in unexpected ways. What about the other three anomalous feeders? At least one of them was further found to be unique. In fact the largest feeder was found to be a landlord lighting connection for a large office block. The final two are inconclusive as the connectivity information is incomplete.

These results are important on a number of levels:

1. The power law relationship suggests that forecasts are more accurate for larger feeders than smaller feeders. For storage applications this can help make decisions on where to use a storage device. For example, on larger feeders an accurate forecast can be generated which means a storage device is more likely to be optimally controlled. In contrast a storage device may be unsuitable for a smaller feeder since the demand is too volatile.
2. It suggests more bespoke methods are required for those feeders which have unusually large errors (those with large deviations from the power law relationship). Identifying these feeders creates opportunities for developing improved forecasts which will increase the opportunities for applications such as storage control to a wider class of substations and cases.

14.3 Example Code

To demonstrate some of the methods and techniques described in this book, and to show how they are implemented in real code, a python notebook has been shared to show some of the steps in analysing data and developing a model. The code can be found at the following repository: https://github.com/low-voltage-loadforecasting/book-case-study.

The notebook will briefly demonstrate topics including:

1. **Exploratory data analysis** using common Python plotting libraries matplotlib[1] and Seaborn,[2]
2. **Feature modelling** using common Python data library Pandas,[3]
3. **Cross-validation** using machine learning library Scikit-learn,[4]
4. **Model fitting and selection** (including simple benchmarks) in Python. In contrast to the previous section the code will focus more on machine learning models (Chap. 10). In particular, common machine learning packages/libraries such as Scikit-learn,[5] and TensorFlow[6] will be presented.
5. **Model evaluation and diagnosis**.

[1] See https://matplotlib.org/stable/index.html.

[2] See https://seaborn.pydata.org/index.html.

[3] See https://pandas.pydata.org/.

[4] Found at https://scikit-learn.org/stable/.

[5] Found at https://scikit-learn.org/stable/.

[6] See https://www.tensorflow.org/.

14.4 Summary

This chapter has highlighted the major components to creating and analysing a successful demand forecast. The chapter has shown, through a residential LV network application, how to apply the techniques and methods described in the previous chapters in order to properly design a forecast trial given the available data. Basic plots such as ACFs and scatter plots have been used to identify key relationships, and these have been supported through various statistical summaries. Point and probabilistic forecasts have been considered and compared using a range of error measures. The comparison of these models was used to identify some of the important features and key relationships.

The chapter has also highlighted the importance of benchmarks for better understanding the forecasts, the role data analysis plays in creating the models but most importantly the chapter has shown the importance of questioning basic assumptions about data and explanatory variables. For example, for load forecasting of residential demand, temperature is often included in all models as it is assumed to be a driving factor for the demand. However, at least in this specific example, including temperature could reduce the forecast accuracy.

A code has been shared with this book and described in Sect. 14.3. This helps to demonstrate how to implement some of the methods and techniques in practice. The reader is encouraged to experiment with generating their own forecasts. A guided walk-through is given in Appendix C which can be used to go through the main steps, from data cleaning to testing.

14.5 Questions

For this section, the ask is to run your own forecast trial. You can follow the same procedure as in Sect. 14.2, or follow the more extensive steps given in Chap. 12. You can also follow the step-by-step walk-through in Appendix C. To perform the experiment select a demand time series from one of those shared in Appendix D.4.

It is also recommend running the code linked to in Sect. 14.3 to get some ideas for practical analysis and implementation.

References

1. S. Haben, G. Giasemidis, F. Ziel, S. Arora, Short term load forecasting and the effect of temperature at the low voltage level. Int. J. Forecast. **35**, 1469–1484 (2019)
2. S. Rahman, Formulation and analysis of a rule-based short-term load forecasting algorithm. Proc. IEEE **78**, 805–816 (1990)
3. M. Bessec, J. Fouquau, The non-linear link between electricity consumption and temperature in Europe: a threshold panel approach. Energy Econ. **30**(5), 2705–2721 (2008)

Chapter 15
Selected Applications and Examples

Forecasts are very rarely produced for their own sake, and are vital for the optimal performance of a number of applications. This chapter summarises a few such applications as well as some adjacent areas which will use some of the similar techniques and methods presented in this book. The main focus will be for battery storage control which is presented in detail in the following section.

15.1 Battery Storage Control

In a low carbon economy, storage devices are going to be essential for maximum utilisation of renewable generation. Most renewable energy, such as solar and wind, are dependent on weather conditions and may not generate at times when the energy is most needed. Storage devices can be used to 'shift' renewable energy by charging when generation is high and then discharging when demand is high but renewable generation is low.

One popular form of storage are battery energy storage systems (BESS). Although BESS have traditionally been an expensive solution, the rapid reduction in cost in recent years is beginning to make them competitive with traditional reinforcement such as replacing existing assets or demand side response (Sect. 15.2). BESS can be deployed for a wide variety of network solutions including, demand smoothing, voltage control, and phase balancing. In this chapter, the primary focus will be on the application of peak demand reduction for the purposes of increasing the network headroom and maintaining the thermal capacity of the network (Sect. 2.2).

Forecasts are used to estimate the distribution of demand throughout the day, in particular the magnitude and timing of the peaks and troughs (the maximums and minimums). This can be utilised into a control algorithm so that the storage device knows the most appropriate times to discharge (at the peaks) and charge (during the troughs). An illustration of the charging and discharging of a storage device is shown in Fig. 15.1. The 'Final demand' in the plot is the resulting demand on the grid and

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5_15

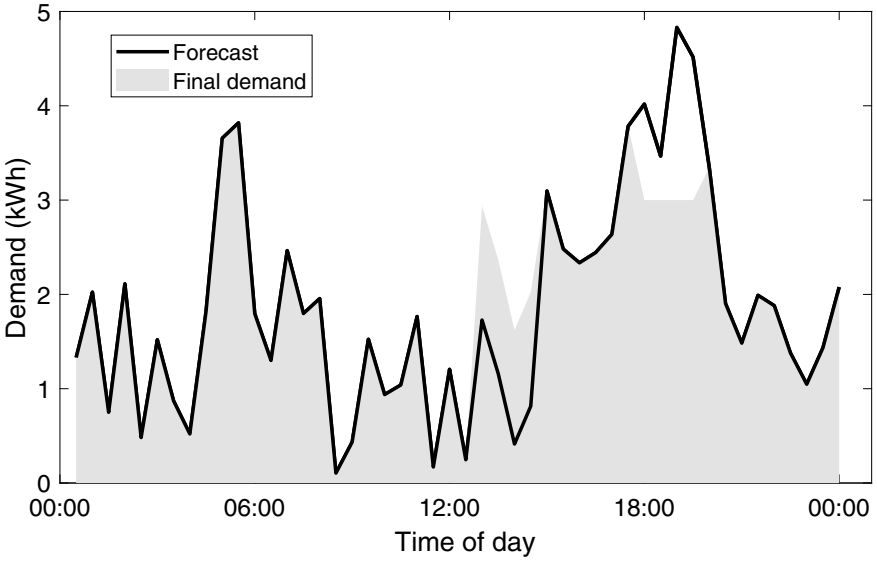


Fig. 15.1 Illustration of how forecasts (black line) are used in storage control. The final demand on the network (shaded) is created by charging the battery when the forecast estimates low demand and discharging when the forecast estimates high demand

is a supposition of the original demand and the charging/discharing of the device at periods of relatively low/high demand.

This process is not as easy as it appears. The demand being considered here is at the low voltage which is much more volatile than higher voltage (HV) and system level (e.g. country wide) demand. Low voltage feeders typically consist of around 40 households and hence have higher degrees of uncertainty than HV systems (see the Case Study in Chap. 14 for a detailed analysis of 100 LV feeders). In particular, an inaccurate forecast may cause a storage device to charge during a high demand period causing an increasing in the peak.

In this section, the peak demand application is based on the results from a previous publication by one of the authors and can be found in [1]. Note that some elements have been simplified to maintain the focus on the impact of forecasts rather than diving into the details of control theory! In the following sections several forecasting methods are developed and then incorporated into a control algorithm which simulates a BESS.

Although a basic control method is considered in this book, more advanced control methods such as Optimal Control, Model Predictive Control (MPC), and Stochastic Model Predictive Control (SMPC) could also be considered which can more optimally solve the peak reduction problem.

15.1.1 Data

This section will be focusing on a few of the same low voltage feeders as used in Chap. 14 and will use a selection of nine feeders to demonstrate how they can be used to support the storage control of a BESS. The feeders have been selected essentially randomly but to include a range of different average demands and volatilities (defined by standard deviation). A few of the attributes of the chosen feeders are shown in Table 15.1. The chosen feeders are labelled according to their approximate size with smaller feeders labelled S1, S2, S3, the medium sized feeders, M1, M2, M3, and L1, L2, L3 the larger feeders.

The monitored data for the selected feeders consists of half hourly energy demand (kWh) for the period from 10th March 2014 to 15th November 2015. The two-week period, from 1st to 14th of November 2015, is used as a test period for assessing the storage control algorithms with the remaining data used for parameter selection (via cross-validation) and training of the forecast models.

15.1.2 Forecast Methods

This section describes several methods used for the load forecasts made up of a mix of machine learning (Chap. 10) and statistical methods (Chap. 9). Throughout this application $L_1, L_2, \ldots,$ will denote the monitored demand time series with L_t the demand at time step t, with the predicted value at the same time step denoted $\hat{L}_t$. Let $t = N$ be the last observed time step, and hence the forecast value at the forecast horizon h will be given by $\hat{L}_{N+h}$.

Table 15.1 Summary features for the feeders considered for the analysis in this chapter. This includes average half hourly demand, standard deviation of demand, and the maximum recorded half-hourly demand (measured over a year). All values are in kWh. Reprinted from [1] with permission from Springer

Feeder	Mean demand	STD	Max demand
S1	7.57	3.32	36.20
S2	5.42	2.89	30.00
S3	11.15	5.86	42.14
M1	11.11	6.05	64.65
M2	16.43	11.14	63.60
M3	21.58	9.40	111.50
L1	30.97	14.43	204.00
L2	37.13	18.04	205.15
L3	24.26	9.74	110.00

The forecasts will generate half hourly forecasts for the next two days starting at midnight prior to the first day. Since the data is the same as in the Case Study in Chap. 14 the same data analysis is relevant here and some of the same models are included. This also helps identify some of the inputs to include in the models such as daily and weekly seasonalities, and autoregressive features.

Method 1: Linear Seasonal Trend Model (ST) The first forecast method is based on the simple seasonal model (ST) presented in Sect. 14.2.3 consisting of annual seasonal terms, and linear trend for different half hours of the day. This generates a mean model forecast, μ_t. This method is labelled **ST**.

Method 2: ST with Autoregressive Correction (STAR) The ST method does not utilise the most recent available information. As mentioned in Sect. 7.5, a standard method for improving these forecasts is to include autoregressive terms. Once the mean equations are found for the ST model a residual time series can be created defined by

$$r_t = \sum_{m=1}^{M_{\max}} \phi_m r_{t-k} + \epsilon_t \tag{15.1}$$

where $r_t = L_t - \mu_t$ and ϵ_t is the error term, and μ_t is the estimate from the ST method. The optimal order $M_{\max}$ is found by minimising the Akaike information criterion (AIC) searched over a maximum order of $m = 15H$ (i.e. an optimal order of up to 15 days is used). The autoregressive terms defines an additional model labelled as **STAR**.

Method 3: Random Forest Regression As presented in Sect. 10.3.2, Random Forest regression **RFR**, is a popular machine learning method, based on combining an ensemble of decision trees. To forecast the load at time $N + h$ the following features are used as input:

- **Load from the past day**. The past $H = 48$ available demand values (a day) $L_N, L_{N-1}, \ldots, L_{N-47}$. This means that for further ahead forecasts the historical input data is less recent than for those for shorter horizons.
- **Load from past weeks**. The load at the same time of the week for the past four weeks, i.e. the inputs $L_{N+h-n_w}, L_{N+h-2n_w}, L_{N+h-3n_w}, L_{N+h-4n_w}$ where $n_w = 336$, the number of timesteps in a week for half-hourly resolution data.
- **Time of the day**. This is defined as an integer between 1 and H. I.e. the half hour period of the day.

In total there are therefore $48 + 4 + 1 = 53$ input features per half hour in the horizon. Note that this means there is a different forecast model for each period in the horizon and a different model for each feeder to be trained. That equates to $96 * 9 = 864$ models to train for a two day ahead forecast for nine feeders. Thus it would be desirable to keep the computational cost low if possible.

The number of trees in the ensemble is an important parameter and must be tuned to its optimal value via cross-validation. To select the optimal number of trees in the random forest, a validation period of one week prior to the test period is

used. Ensembles with varying number of trees from 5 to 100 in increments of 5 are considered. A value of 30 trees in the ensemble of a Random Forest was found to be sufficient trade-off between forecasting accuracy and performance, since smaller trees are less computationally expensive.

To train the final forecasts, the Random Forest is trained using the one year prior to the test period, 1st November 2014 to 31st October 2015.

Method 4: Support Vector Regression Recall from Sect. 10.2 that a Support Vector Regression (SVR) can solve a nonlinear regression of the form

$$\hat{L}_{N+h} = \boldsymbol{\beta}^T \Phi(\mathbf{X}_{N+1}) + b. \tag{15.2}$$

for inputs e.g. $\mathbf{X}_t = (X_{1,t}, X_{2,t}, ..., X_{n,t})^T$. This is solved by using kernel functions $K(\mathbf{X}_i, \mathbf{X}_j) =< \Phi(\mathbf{X}_i), \Phi(\mathbf{X}_j) >$ where $<, >$ represents an inner product and can be written in the dual form

$$\hat{L}_{N+h} = f(\mathbf{X}) = \sum_{t=1}^{N} \alpha_t K(\mathbf{X}_t, \mathbf{X}) + b. \tag{15.3}$$

The optimal fit for the weights α and the intercept b can be found by minimising a regularised cost function.

As shown in Sect. 10.2, SVR has two hyperparameters, C and ϵ, that require tuning. The regularisation constant C controls the flatness (or complexity) of the model, a trade-off between empirical error and model flatness, and ϵ determines the amount of error allowed by the model (values far from the model expectation line). In addition, the choice of kernel $\phi()$ is also important for the final model. The hyper parameters can be found via many different methods but in this example, grid search is considered (see Sect. 8.2.3). To simplify the task the error allowance term is set to $\epsilon = 0.1$, and three kernels are considered for the regression: a linear, a radial basis function (RBF) and a polynomial (Sect. 10.2). As an example, the Gaussian Radial Basis Function (RBF) given by

$$K(\mathbf{X}_i, \mathbf{X}_j) = \exp\left(-\gamma ||\mathbf{X}_i - \mathbf{X}_j||^2\right), \tag{15.4}$$

The regularisation constant C is restricted to vary from 0.1 to 100. The RBF kernel has an extra free parameter, γ, which controls the width of the kernel, and varies from 0.01 to 100. Finally, the degree of the polynomial kernel requires tuning too, changing from 2 to 5.

As with the RFR model, hyperparameter selection is performed using the week prior to the test-period as a validation period. The linear kernel is chosen since it outperforms both the RBF and the polynomial kernels for all values of the C parameters. With the linear kernel, large values of $C > 20$ seem to reduce the model accuracy (in terms of MAPE) and so the regularisation constant is fixed at $C = 1$ for all feeders.

Since the Support Vector Regression forecast is more computationally intensive than the Random Forest Regression, a shorter training period of eight weeks prior to the test period, i.e. 5th September 2015 to 31st October 2015, is used.

Benchmark Methods Informative benchmarks are also included to compare with the more sophisticated models (Sect. 9.1). They can also help to understand the factors which best drive the forecast accuracy and the storage control performance. Further, because they are computationally inexpensive, if they perform well, they will scale up well and there is no need to implement more intensive methods.

The first simple model is the simple **seasonal average model** given as

$$\hat{L}_{N+h} = \frac{1}{p} \sum_{k=1}^{p} L_{N+h-kn_w} \tag{15.5}$$

where $n_w = 336$ is the number of time steps in a weekly period. Testing shows that using $p = 5$ weeks of data is the optimal hyperparameter. The model is denoted **7SAV5**. This model is mainly motivated by the fact that the recent past is usually important for the actual behaviour.

The other benchmark considered is the **seasonal persistence** model, which is technically a special case of the average model but simply uses the last week as the current week estimate

$$\hat{L}_{N+h} = L_{N+h-n_w} \tag{15.6}$$

This special case is denoted **SALW**.

15.1.3 Analysis of Forecasts

In this section the accuracy of the forecasts is analysed. A standard error measure, the mean absolute percentage error (MAPE), is used given by

$$MAPE\,(a, f) = \frac{1}{n} \sum_{k=1}^{n} \frac{|a_k - f_k|}{a_k} \tag{15.7}$$

where $a = (a_1, \ldots, a_n)^T \in \mathbb{R}^n$ is the actual/observation and $f = (f_1, \ldots, f_n)^T \in \mathbb{R}^n$ is the estimate/forecast. MAPE isn't ideal or advisable for low voltage feeders, however this work is replicating a previous piece of work and in fact the MAPE is strongly correlated with other relative errors such as rMAE (see Sect. 14.2). However, the MAPE does gives a simple way of comparing and combining the errors of feeders of different sizes, since the differences are normalised by the size of the demand. The results presented here are for the entire two-week test period.

The Table 15.2 shows the MAPE scores for day ahead forecasts for each method and each feeder considered in this trial. The best methods tend to be the STAR

Table 15.2 MAPE for day ahead forecasts for each of the methods described in the main text. The best score for each feeder is highlighted in bold. Also shown is the average score for each method. Reprinted [1] with permission from Springer

Feeder	Methods					
	SALW	7SAV	ST	STAR	RFR	SVR
S1	20.20	15.54	15.29	**14.98**	17.41	16.53
S2	34.72	26.81	29.15	28.28	38.86	**26.75**
S3	24.49	18.97	**17.49**	17.57	25.20	21.06
M1	21.11	14.78	15.94	**13.69**	16.73	14.98
M2	27.92	25.69	29.02	**24.33**	33.03	38.43
M3	17.77	13.58	13.98	**12.76**	16.32	13.87
L1	15.04	11.30	10.71	**10.13**	14.81	12.18
L2	30.86	**10.76**	13.75	11.22	12.83	11.33
L3	18.63	15.37	17.48	**14.62**	22.79	19.33
Average	23.42	16.98	18.09	**16.40**	22.01	19.38

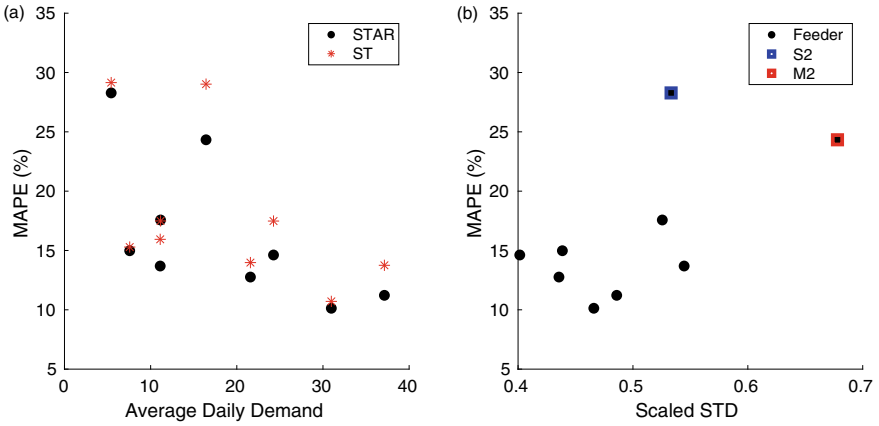


Fig. 15.2 Plot of Forecast errors for each feeder against **a** Average daily demand on for STAR and ST methods, and **b** scaled standard deviation for the STAR method. Also highlighted is feeder M2 (red square) and the S2 feeder (blue square)

methods with a MAPE of 16.40%, although the ST, SVR and 7SAV all produce the best forecast for a feeder each. This is consistent with the results in Sect. 14.2 where the best methods utilised autoregressive components.

The inclusion of the autoregressive component in STAR produces, on average, a 10% improvement over the seasonal trend model, ST. Although this improvement varies depending on the feeder. This is highlighted in Fig. 15.2a which shows the average error vs the average daily demand of each feeder, for both ST and STAR methods. This plot shows that there is a negative correlation between size of feeder and the accuracy of the forecast. This make sense since larger feeders are generally

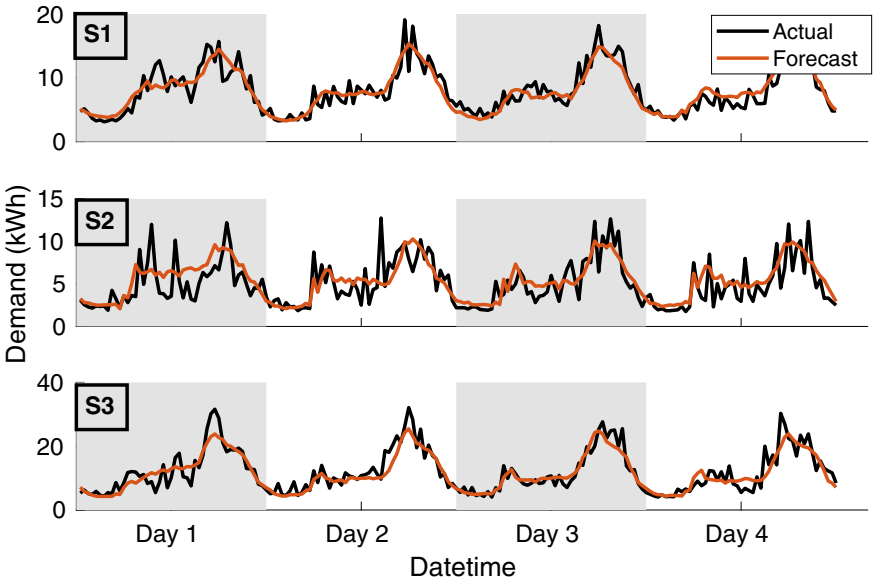


Fig. 15.3 Day Ahead forecasts (orange) using STAR method for the Small feeders for first four days of test data. Also shown are the actuals (black)

smoother and more regular and can therefore be more accurately estimated. However, there are two feeders S2, and M2, which are estimated quite inaccurately compared to the others. One explanation for this is suggested in Fig. 15.2b which shows the errors for the STAR method against the standard deviation (STD) scaled by the average daily demand. This gives a measure of the relative volatility of a feeder and shows that these feeders seem to have also high variability, especially M2.

To better understand the errors (and how easy it will be for a storage device to reduce the peak) a few plots for each feeder are shown for the STAR day ahead forecasts for the first four days of the test set. Figure 15.3 shows the forecasts and the actuals for the small feeders. These feeders seem to have been forecast accurately in all cases although S2 is has much more volatility around the forecast value as already suggested by the high scaled STD value. S1 and S2 seems to have larger demands in the morning and evening periods compared to S3 which generally only has a peak demand in the evening. This means that to reduce the daily peak may be more difficult for S1 and S2 as the battery will have to charge and discharge appropriately in both periods to reduce the peak (and also avoiding overcharging too early prior to the main peak and risk creating a larger morning peak). Hence it is expected the greatest peak reduction for the smaller feeders will be for S3.

Figure 15.4 shows the same data but for the medium sized feeders. The forecasts again seem to do a good job of estimating the general shape of the daily profiles. As with S2, the high standard deviation in M2 is clear. This suggests that it may be difficult to maximally reduce the peaks for this feeder. It also appears that it will be

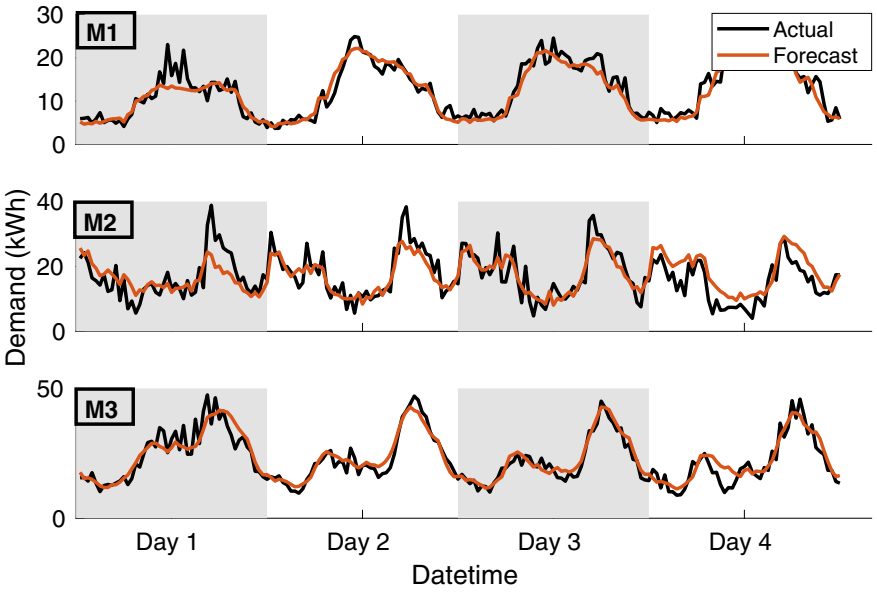


Fig. 15.4 Day Ahead forecasts (orange) using STAR method for the Medium feeders for first four days of test data. Also shown are the actuals (black)

difficult to reduce the peaks on M1 by a large percentage. The demand is relatively large throughout the day suggesting that the feeder is connected to businesses which operate during this period. Since the storage device will have to reduce demand over several time steps there will not be enough capacity in the battery to produce large reductions. Storage applied to M3 will probably produce the largest peak reduction because the main peak is in the evening and it appears to be accurately forecast.

Finally, Fig. 15.5 shows the day ahead STAR forecasts and actual demand for the largest feeders. The forecasts once again seem to produce good estimates of the daily demand of the feeders. Feeder L2 looks like it is a feeder connected to many or one large business. In addition to the large daily demand (and no demand during morning or evenings) the first day (a Sunday) has zero demand suggesting this is a single commercial consumer with no operation on the weekend. This large demand will mean that the storage device will likely not be able to significantly reduce the peak. L1 looks very accurately estimated and in fact has the smallest MAPE on average (Table 15.2). This feeder has a major singular peak which seems to regularly occur in the evening, and therefore there could be significant peak reduction on this feeder. Finally, L3 has not got a prominent peak on most days and but seems to have two peaks on most days, one in the morning and one in the evening. A relatively large peak reduction may not be possible for this feeder if the battery cannot recharge quick enough to reduce the second peak after reducing the first peak.

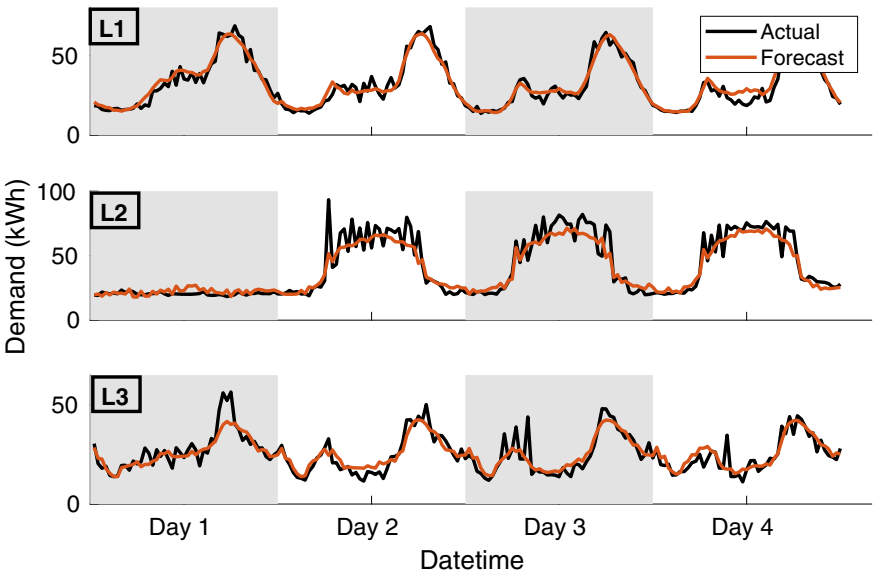


Fig. 15.5 Day Ahead forecasts (orange) using STAR method for the Large feeders for first four days of test data. Also shown are the actuals (black)

15.1.4 Application of Forecasts in Energy Storage Control

The effectiveness of a storage device at reducing daily peak demands depends on the specifications of the battery, namely it's capacity (how much energy it can store), its ratings (how fast it can charge and discharge), and its age (batteries become less effective the more they are used, but also how they are used, e.g. performing many cycles of charging to full and emptying completely). For other applications other criteria such as location on a feeder, whether there is real-time control etc. can also be important. The BESS will be sized so that theoretically a peak demand reduction of 20% can be achieved. The aim is to see what the effect of future demand uncertainty has on the performance, and what part forecast accuracy plays. The ratings and capacity ranges used in this experiment will change for each day in the test set, but obviously in a real world example it would be fixed and could be sized by analysing historical behaviour. The required rating will depend on how high the peak is over any half hour, the higher it is the faster a battery would need to discharge to be able to reduce it. The relative capacity is also related to the size of the peak, but if there are large demands during periods adjacent to the peak then these will also have to be reduced in order to decrease the overall daily peak (This is the case with feeders M1 and L2 as seen in Figs. 15.5 and 15.4 respectively). This will require a larger capacity to reach a particular percentage peak reduction. In general the bigger the demand on a feeder the bigger the capacity required.

Forecasts are used as estimates of the future demand, and the battery control algorithm will assume these are true estimates in order to develop a schedule which maximises the peak reduction. The schedule will be developed at midnight prior to the day of interest. Further another aim is to have the BESS with 50% state-of-charge at the midnight so it is prepared for any early peaks the following day.

Assume that p_t is the power outputs (in kW) from the battery then the following constraints can be defined. First, the ratings are bounded by a maximum charging rate, $P_{\max} > 0$ and a minimum $P_{\min} < 0$, i.e.

$$P_{min} \leq p\,(t) \leq P_{max}. \tag{15.8}$$

Next it is assumed that the capacity c_t of the battery (in kWh) is also constrained between a maximum $C_{\max} > 0$ and a minimum $C_{\min} > 0$, i.e.

$$C_{min} \leq c_t \leq C_{max}, \tag{15.9}$$

The absolute bound on the capacity is obviously zero (empty battery) but this can cause deterioration of the battery and so it may be desirable to set the minimum to be some strictly positive value. Obviously there is also a constraint in terms of how the capacity and the charging/discharging relate from one time step to the next, with the capacity changing depending on how much energy was added/removed from the battery, i.e.

$$c_{t+1} = c_t + 0.5(p_t \mu - \lambda) \tag{15.10}$$

with

$$\mu = \left\{ \begin{array}{ll} \mu & \text{if } p\,(t) \geq 0 \\ \frac{1}{\mu}, & \text{if } p\,(t) < 0 \end{array} \right\}, \tag{15.11}$$

where, μ is the efficiency (96% in each direction), λ is the continuous losses within the BESS (assumed to be 100 W). I.e. not all of the energy will be transferred due to natural battery limitations. Note that the 0.5 in Eq. (15.10) converts the average power (in kW) into average energy (in kWh) because the data being considered is half hourly.

The control method presented here uses fixed day-ahead scheduling. As input it uses the forecasts to decide on the periods of charging and discharging subject to the constraints above. Let $\mathbf{P} = (P_1, P_2, \ldots, P_{48})^T$ be the charging schedule and let $\hat{\mathbf{L}} = (\hat{L}_{N+1}, \hat{L}_{N+2}, \ldots, \hat{L}_{N+48})^T$ be the predicted demand (both in kW) for the day ahead.

To create the schedule the following cost function[1] is minimised with respect to $\mathbf{P}$ and subject to the forecast, $\hat{\mathbf{L}}$ and the battery constraints (15.8)–(15.11)

$$F\left(\mathbf{P}, \hat{\mathbf{L}}\right) = \xi_p\left(\mathbf{P}, \hat{\mathbf{L}}\right) + \xi_{cd}\,(\mathbf{P}) \tag{15.12}$$

[1] Note that this is slightly different from the one in the original publication. This simplifies the presentation and allows focus on the forecast component and its effect.

The first component represent the new peak size and is the Peak-to-Average cost component for peak reduction, self-normalised to the initial conditions, defined

$$\xi_p(\mathbf{P}, \hat{\mathbf{L}}) = \frac{1}{R}\left(\frac{max_{t=1,\ldots,48}(P_t + L_k)}{\frac{1}{48}\sum_{k=1}^{48}(P_t + L_k)},\right)^2 \tag{15.13}$$

where R is the final peak size given an initial schedule $\mathbf{P}^{initial}$

$$R = \left(\frac{max_{t=1,\ldots,48}(P_t^{initial} + L_k)}{\frac{1}{48}\sum_{k=1}^{48}(P_t^{initial} + L_k)}.\right)^2 \tag{15.14}$$

The second component aims to achieve a 50% State-of-Charge (SoC) at the end of the day and is defined as:

$$\xi_{cd}\,(\mathbf{P}) = \frac{(c_{48} - 0.5C_{max})^2}{(c_{48}^{initial} - 0.5C_{max})^2}, \tag{15.15}$$

where c_t is the charge in the battery at time t. The initial end charge based on the initial schedule is given by $c_{48}^{initial}$.

15.1.5 Results

Average peak reduction performance for each feeder for each day-ahead forecasts is given in Table 15.3. The best possible values assuming perfect foresight are also shown. This almost gets the maximum possible peak reduction of 20% for all feeders

Table 15.3 The overall peak reduction by applying a storage control to each feeder for each day-ahead forecast. The best results for each feeder are highlighted in bold. Reprinted from [1] with permission from Springer

Feeder	Best	SALW	7SAV5	ST	STAR	RFR	SVR	Av
S1	19.36	0.75	0.83	**2**	**2**	1.86	1.56	1.5
S2	18.83	0.44	2.41	**5.09**	4.84	2.63	3.34	3.13
S3	19.91	3.03	5.39	7.86	**7.92**	4.65	7.34	6.03
M1	18.75	2.42	2.03	**4.28**	3.76	4.15	2.36	3.17
M2	19.41	2.35	**4.56**	1.8	3.95	1.37	4.24	3.05
M3	19.76	7.57	10.14	**10.56**	10.43	6.8	10.5	9.33
L1	19.12	7.91	7.74	**12.23**	11.16	11.37	9.7	10.02
L2	13.68	1.94	1.12	1.12	2.75	**3.22**	0.25	1.73
L3	19.49	6.02	7.89	8.36	**9.1**	6.04	8.97	7.73
Ave.	18.7	3.60	4.68	5.92	**6.21**	4.68	5.36	5.08

except L2. The question that could be asked is, why isn't it exactly 20% since the future is exactly known? To answer this, recall that the cost function Eq. (15.12) isn't just focused on peak reduction, it also has to ensure the battery is charged to about 50% at the end of the day. The closer the peak is to the end of the day the more this criteria will effect the final peak reduction. Further, if there are multiple peaks in close proximity to each other then the battery may not have sufficient time to recharge between peaks and reduce the subsequent peaks.

The table shows that the STAR method produces the largest percentage peak reduction on average, 4.5% larger than the next biggest (ST). However, it only produces the biggest peak reduction for three feeders, the simpler model ST actually has the biggest peak reduction for five feeders (tying STAR for feeder S1). Recall that STAR was the best forecasting method for most feeders and this highlights an important point: the accuracy of the forecast doesn't necessarily mean it will produce the greatest performance in the application. This can be an easy point to miss. Forecast error measures are usually simple and easy to calculate, in contrast to training directly via the cost function. However, it would be impractical to assess the forecasts using this cost functions due to computational costs and so a compromise is to utilise a related more but simple measure which hopefully still indicates the application performance.

The last column of the table shows the average peak reduction across all forecasts (and doesn't include the "Best"). It shows that although there is a trend of better peak reduction for larger feeders it isn't straight forward. This is despite the correlation between accuracy and feeder size (see Fig. 15.2). Figure 15.6a shows the percentage peak reduction against feeder size for the ST and STAR methods. In this case there is a trend, with lower peak reduction with larger feeder size, but there is at least one outlier with large demand but small peak reduction. This is feeder L2, which as shown in Fig. 15.5, appears to be a single commercial load with no operation on

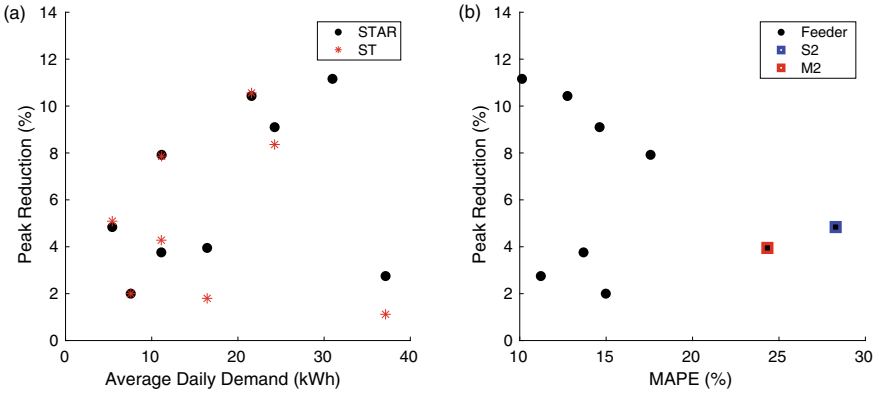


Fig. 15.6 Plot of percentage peak reduction for each feeder against **a** Average daily demand on for STAR and ST methods, and **b** MAPE for STAR method. Also highlighted is feeder S2 (blue square) and M2 (red square)

Saturday and Sunday. Not only does this mean peak reduction is not possible on the two weekend days each week, but the other five days have large continuous daytime demands which mean peak reductions are difficult. To reduce the daily peak demand on this network requires a much larger storage device which can discharge a lot of demand over a larger portion of the day.

Figure 15.6b shows the peak reduction for each feeder against the MAPE for the STAR method. In general the more accurate the forecast (the smaller the MAPE) the bigger the percentage peak reduction. However there are three feeders which do not fit the trend. One of these is L2 which has already been discussed. The others are S1 and M1. In fact M1 (Fig. 15.4) has lower peak reduction due to some of the similar demand features as L2. Again the demand is relatively large throughout the day, possibly due to several commercial consumers connected to this feeder. The low peak reduction for S1 is more difficult to explain but there is large demands in the morning on some days (Fig. 15.3) which may reduce the energy available in the battery for reducing the evening peak.

Highlighted in Fig. 15.6b is the feeders S2 and M2 which you may recall have relatively large standard deviation (Fig. 15.2). These have relatively large MAPE and also have small peak reduction. The volatility of these forecasts mean that the data is relatively spikey and thus makes it difficult to provide an accurate forecast. A storage control schedule based on the forecast may inadvertently charge during higher charge periods or discharge in relatively lower periods. Therefore these feeders only have small peak reductions.

It should be noted that there are only nine points in these plots. Thus there should be some caution with being over interpreting the results and they may not generalise more widely.

A take home message from these results is that there is no one-size-fits all method. A next step may be to consider taking simple averages of the forecasts to see if this improves things (Sect. 13.1). In addition, since the data is quite volatile, probabilistic forecasts may also be a good option (Chap. 11). They may help to improve the results for the more volatile feeders. In addition, there are more advanced control techniques out there such as model predictive control which could also be considered.

15.2 Estimating Effects of Interventions and Demand Side Response

Demand side response (DSR) is deployed by turning on or off demands to react to possible strains on the network or to ensure energy supply matches energy demand. DSR could be as simple as turning on a load to increase the demand or, more commonly, turning off devices to reduce the demand on the network. For example, heat pumps could be turned off to reduce the demand during peak hours. Over a short period of time such interventions may not have a significant impact on the heating comfort within a home since, unless the home is not well-insulated, the temperature should not drop too rapidly.

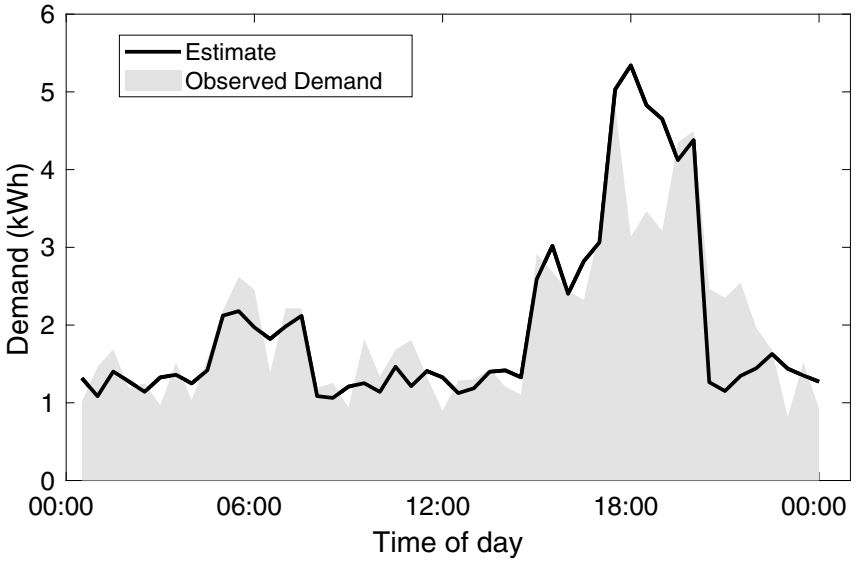


Fig. 15.7 Illustration of DSR turn down. The measured demand (shaded) is compared to an estimate without intervention (bold line). The comparison can be used to estimate the energy saved by turning off devices

An important question for these applications is how much energy was saved by the demand side response implementation? This is also known as "turn-down". Forecasts can help answer this question.

Figure 15.7 shows both the actual demand after demand side response (shaded), and what the demand would have been had no intervention been applied (bold line). The shaded area is the adjusted demand created by the 'turn-down' event, for example by turning off the controllable appliance. The energy saved at 6PM is the difference in the area between the shaded part and the line. Of course, there is no way to know what the demand would have been had there been no DSR which means the consumer has no way of knowing how much energy they saved. In particular, if they are participating in any energy market schemes, they will not know how much payback they may have received.

Forecasting is an effective way to estimate the amount of turn-down since a model trained on "typical" demand can estimate what the demand would have been in the absense of an intervention (as long as the historical data used for training does not include interventions either). Thus the turn-down is simply the difference between the recorded demand (where the intervention has been applied) and the estimated demand from the forecast model.

The estimate in Fig. 15.7 could be estimated using the time series methods introduced in this book. Of course there will be natural variation in the demand but if the forecast is reasonably accurate (and this should be tested of course) then accurate estimates of average turn-down can be produced. Notice that the example in the

figure has a much larger demand than expected after the turn down period. This can occur in some applications, and is known as a 'rebound' effect caused by adjusted behaviour, or extra demand in order to recover from the turn down. For example, this could be extra heating applied by the occupants to recover the comfort levels in the home after the DSR event.

Notice that the model can be trained using demand data from after the DSR event since the application is in fact a **backcast** rather than a forecast and the aim is to estimate, not predict, the demand. This means the estimates may in fact be more accurate than a normal forecast since more information is available.

15.3 Anomaly Detection

Chapter 6 already discussed ways to identify anomalous data. However, similar techniques can be used to identify anomalous *behaviour* rather than errors in the recorded data. This is important to identify things like energy theft, or whether the security of supply to vulnerable customers, with for example medical needs, are at risk (although privacy concerns would have to be considered for such applications).

Such anomalies can be detected if they deviate from the expected demand, and of course models used to create forecasts can be used to estimate typical demand, or model the uncertainty. One example would be to develop point forecast models to estimate the daily demand which can be used to identify unusually large demands or appliances. Load forecasts can also be used to identify unusually small demand. This could indicate the monitoring is broken, or that someone is rerouting their usage to artificially lower their bills! Another example would be to use probabilistic forecasts (Chap. 11), e.g. Quantile forecasts, to identify observations which lie in the extreme outliers. Large numbers of these variables can suggest something unusual is occurring.

Sudden large increases in the forecast errors also may suggest sudden changes in behaviour. This could suggest new occupants, new technologies, or simply large behavioural changes (for example the covid pandemic has led to many individuals working from home). This information could, in turn, lead to new solutions to support the network or help network operators plan their infrastructure upgrades.

15.4 Other Applications

There are many other topics which haven't been explored in the use cases above, but forecasting has many other applications it can support which are briefly outlined below

- **Further Battery Applications**: Forecasts can also be used to optimise solar PV connected batteries, minimise curtailment loss, control multiple batteries in electric vehicles, and regulate voltage.
- **Network Design and Planning**: Forecasts can be used to size assets on the network (capacitors, substations etc.), and also plan the networks themselves (topology, location of batteries, sectionalising switches, etc.).
- **Electricity Price Forecasts**: Energy markets rely on the estimated future demand, and therefore can be valuable inputs to price forecasting algorithms.
- **Simulating Inputs and Missing Data**: Instead of the simple imputation models given in Sect. 6.1.2, more sophisticated load forecast models could be used. Forecasts can also be used to simulate inputs for other applications, for example power-flow analysis.

There are many other low voltage applications for load forecasts and these can be found in the review paper [2].

15.5 How to Use Forecasts in Applications

Below are a few guidelines for experimenting with utilising forecasts in real world applications:

1. Try to understand what features may be most important for the performance of the application and try to design the error measure so it represents or aligns with this.
2. Remember: whatever error measure is used it will not exactly correlate with the the performance of the application (unless you use the associated application cost function for the assessment—which is not often practical).
3. Design the forecasts with the application and practicality in mind. If there is high levels of volatility then perhaps probabilistic forecasts are more appropriate. However, if there is limited data, or limited computational resources, this may not be possible and point forecasts may be more appropriate.
4. In the case were probabilistic forecasts seem appropriate, it may be worth considering point forecasts anyway since the performance difference may be minimal and the savings in resources may be worth the drop in optimality.
5. Use at least one benchmark forecast but preferably several to help investigate the performance of the main models.
6. Try to understand how forecast accuracy relates to performance within the application. If there is a trend, is it dramatic or small? Possibly drops in accuracy do not correspond to a large drop in performance. In which case simpler methods may be appropriate, and there is not much point in spending time perfecting the forecast models. In contrast, if small improvements in forecast accuracy create large performance changes (or large monetary savings) then perhaps a focus on small improvements to the forecast is worth the effort (at least until there is diminishing returns to this effort).

7. It is worth remembering that in-silico tests are limited as there will often be a whole host of other complications and challenges when applying the methods in practice. For example, to control a storage device will require reliable communications equipment, properly functioning power-electronics, and may involve lags and delays in processing etc. Ideally many of these considerations should be included in the design of the algorithms but there will always be some simplifications.

References

1. T. Yunusov, G. Giasemidis, S. Haben, *Smart Storage Scheduling and Forecasting for Peak Reduction on Low-Voltage Feeders* (Springer International Publishing, Cham, 2018), pp. 83–107
2. S. Haben, S. Arora, G. Giasemidis, M. Voss, D.V. Greetham, Review of low voltage load forecasting: Methods, applications, and recommendations. Appl. Energy **304**, 117798 (2021)

Appendix A
Stationary Tests for Time Series

Stationarity as introduced in Sect. 5.1, is an important property for time series analysis and forecasts, especially for applying ARIMA models (Sect. 9.4). Since it is not obvious that a time series is stationary from the time series plots alone, statistical test are often applied to provide further supporting evidence. This section briefly discusses a specific family of stationarity tests: **unit root tests**.

The Augmented Dickey-Fuller (ADF) and Phillips-Perron unit root tests are two of the most popular tests for stationarity. Consider a simple ARMA model

$$\hat{L}_N = \sum_{i=1}^{p} \psi_i L_{N-i} + \sum_{j=1}^{q} \varphi_j \epsilon_{N-j}, \tag{A.1}$$

then the series $L_1, L_2, \ldots,$ is stationary if the absolute value of all the roots of the polynomial

$$1 - \sum_{i=1}^{p} \psi_i x^i, \tag{A.2}$$

are greater than 1 (see [1]). The method tests whether the null hypothesis 'there is a unit root' holds. It center around the concept that a stationary series (i.e. with no unit roots) should revert to the mean and hence the lagged values should indicate relevant information for predicting the change in the series. The details are beyond the scope of this book but the test themselves are often included in the various software packages and hence are relatively simple to apply. These packages typically give significance levels at which the null hypothesis can be rejected. For further information on basics of hypothesis testing see an introduction text such as [2].

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5

Appendix B
Weather Data for Energy Forecasting

Weather is considered one of the most important variables for forecasting energy loads. This is because many household behaviours and appliances are tied to weather, e.g.

- When it is too cold then heating is turned on.
- In warm countries, when it is too cold, air-conditioning is switched on.
- How cloudy it is determines how much generation is produced from solar photovoltaics.

Combination of weather variables can change the effects. For example, temperature effects can be intensified when the humidity is high. The temperature therefore feels higher to people and thus cooling appliances may be turned on at lower actual temperatures. However, as shown in the case study in Sect. 14.2 the effect can be unexpected. The case study showed that utilising temperature as an input did not produce more accurate low voltage level forecasts, suggesting that temperature is not a large determinant of electricity used. Possible explanations were given, in particular that most heating in the UK (the location of the data) are gas heated at this moment and therefore less likely to be influenced by temperature changes than homes which are electrically heated. However, behaviours are always changing (Sect. 13.6.3), and with the increased uptake of heat pumps, electrical load is likely to become more and more linked to weather effects, especially temperature.

This chapter is a short overview of some of the concepts in weather and weather forecasts because of its potentially strong links with load. Some of the main variables are discussed and the different forms they can take. For weather forecasts this section briefly discusses how they are generated so that the reader can understand potential sources of error and uncertainty.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5

B.1 Weather Variables and Types

Section 6.2.6 has already briefly introduced some of the main variables used in load forecasting. They include temperature, wind speed, humidity, solar radiance, visibility. However, other variables such as precipitation and pressure may also be useful as there are interdependencies between different weather variables which can create various effects. It should be noted that meteorology is a very complicated discipline in its own right and most data scientists will have limited knowledge of weather and climate science. For these reasons if a forecaster wishes to include more complicated weather variables, or derivations from them, it may be a good idea to consult with someone who is much more knowledgeable in this area. Having said that, the standard observed variables listed above are usually sufficient for the purposes of short term load forecasting, and additional features may only bring minimal improvements in accuracy. This section will discuss a few features to be aware of when using weather variables in your models.

One of the first things to understand is the units of your data. Although numerical weather centres typically use standard units[1] for their variables there are common variations which are not always clear. The most obvious is whether the temperature is in Celsius or Fahrenheit (or maybe even Kelvin!) which may be more common depending on which country you are from. Pressure data is particularly confusing and can be written in pascals (the SI units), atmospheres, bars, PSI (pounds per square inch) or Newtons per square meter.

Each variable often has variations and some of them may be more useful than others depending on the application and context. For example, the European Centre for Medium-Range Weather forecasts (ECMWF) has a whole host of variants for each variable.[2] Even an "obvious" variable like temperature has a whole host of options from which to choose from. Further, variables like radiation has variations such as longwave radiation, incident shortwave, global horizontal irradiance, direct normal irradiance etc. Although there is guidance shared by the major numerical weather prediction centres the task can still be daunting for a novice. It is usually easier to utilise observation data as it is recorded directly at a particular site. For load forecasting surface temperature and wind speed (usually split into two orthogonal components) are usually of particular interest. However, even using these there are several considerations in how to use and preprocess them (see below and Sect. B.2.2).

Although the forecast data is usually at hourly resolution (observational data may be more frequent) the form that weather variables are reported can also vary. Some values are measured instantaneously at each hour, but other variables are averaged over the entire hour. Care must be taken with the averaged values as it needs to be clear which hour interval is used to produce the final average. Is it the hour prior to the timestamped value, or the hour after, or is it formed from values from half hour either side of the value? This can have implications for how you use this variable in your

[1] The so called International System of Units, or SI units.

[2] See https://apps.ecmwf.int/codes/grib/param-db/ and try searching for various variables: radiation, temperature.

model such as, for example, utilising lagged values. The instantaneous variables can also be problematic since the data may be relatively unusual at the time it is recorded compared to the rest of the hour.

It should be noted that there are many different weather products but they often come in three main forms:

1. **Observations**: As the name suggests these are observed values according to various sensors. This data could be from weather stations, drifting buoys, satellites, or aircraft.
2. **Forecasts**: These are developed by combining observations with a forecast model of the atmosphere. These will be described in more detail in Sect. B.2.
3. **Reanalysis**: After the forecast horizon has passed, the forecast models can be re-optimised using the actual observations. This effectively gives an estimate of the past weather states.

Unlike forecasts, which are defined on a predefined grid, observations locations may be focused in particular areas and hence not near to the site of interest. This means, even if weather variables are strongly related to demand, the distance from the site may mean the observed variables are not useful for the model.

If there are several observation sites then it may be that one site improves the model more than the other, or an improvement could be produced by combining them (Sect. 13.1). Alternatively, all the variables could be included and a method such as LASSO could be used to select the most appropriate variable inputs (Sect. 8.2.4). Also note that lagged values of the weather time series could be useful for the forecast models due to lagged effects (especially if the weather variables are not colocated with the demand site).

B.2 Numerical Weather Forecasts

The production of weather forecasts is known as numerical weather prediction (NWP). There are many centres around the world which create weather predictions but many are at least part funded through governments.[3] NWP is an computational expensive process as it requires the optimisation of cost function requiring many variables. This section dives further into the weather prediction process and how to understand it.

[3] One of the most accurate NWP centres is the European Centre for Medium-Range Weather Forecasts (ECMWF). An independent intergovernmental weather prediction centre which is funded by several European countries. It produces many different forecasting products including a 15 day ahead forecast, and an ensemble forecast.

B.2.1 How Weather Forecasts Are Produced

Weather prediction combines models of the atmosphere with observations, and a prior guess on the current state of the atmosphere to find a optimal estimate for the current state of the atmosphere. This current state is then evolved forward using the forecast equations to estimate the future states of the atmosphere. The process of finding the optimal state of the atmosphere is known as *data assimilation*, and will be described below.

The models of the atmosphere are essentially a collection of partial differential equations which includes fluid dynamics, thermodynamics and ocean-atmosphere interactions. These equations must be trained on the observed data so that they can produce forecasts for several weather variables, for the next few days (usually at hourly resolution) for every grid point in the area of interest. Many NWP centres have to generate forecast for the entire Earth. In fact the grid points are not just at latitude and longitudinal points but also go several levels above the surface. This means the forecast models must solve for a state space with an order of at least 10^8. The problem is the number of observations is often an order or two smaller than this (say 10^6). Therefore the problem is underdetermined. For this reason a prior guess is required to provide estimates for the full state space.

One main form of data assimilation is 4-dimensional variational data assimilation. It is essentially a least squares estimation (Sect. 8.2.1) with a regularisation term and can be written as

$$J(\mathbf{x}_0) = \frac{1}{2}(\mathbf{x}_0 - \mathbf{x}_0^b)^T B^{-1}(\mathbf{x}_0 - \mathbf{x}_0^b) + \frac{1}{2}\sum_{k=0}^{N}(\mathcal{H}_k(\mathbf{x}_0) - \mathbf{y}_k)^T R_k^{-1}(\mathcal{H}_k(\mathbf{x}_0) - \mathbf{y}_k) \tag{B.1}$$

Here the aim is to find the initial state, $\mathbf{x_0}$, that optimises the cost function above, where:

- $\mathbf{x}_0^b$ is an initial prior estimate of the initial state, also known as the background. This is usually the previous forecast value or can be a climatological estimate.
- $\mathbf{y}_k$ are the observations at time step k in the forecast horizon.
- $\mathcal{H}_k()$ is a combination of the weather forecast model and an observation operator. This evolves the current estimate $\mathbf{x_0}$ to time step k and then transforms the state to the same location and type as the observed variable.
- $\mathbf{B}$ is the error covariance (Sect. 3.3) for background errors.
- $\mathbf{R}_k$ is the error covariance for the observations.

Thus data assimilation is a nonlinear optimisation problem where one component measure the difference between the observations and the model forecast of the current guess and the other component measures the difference between the prior guess and the current estimate. The background term acts as a regularisation term (Sect. 8.2.4) and fills in the missing values from the observations. The covariance matrices act as weights for the optimisation so that the model/estimates fit closer to more precise values.

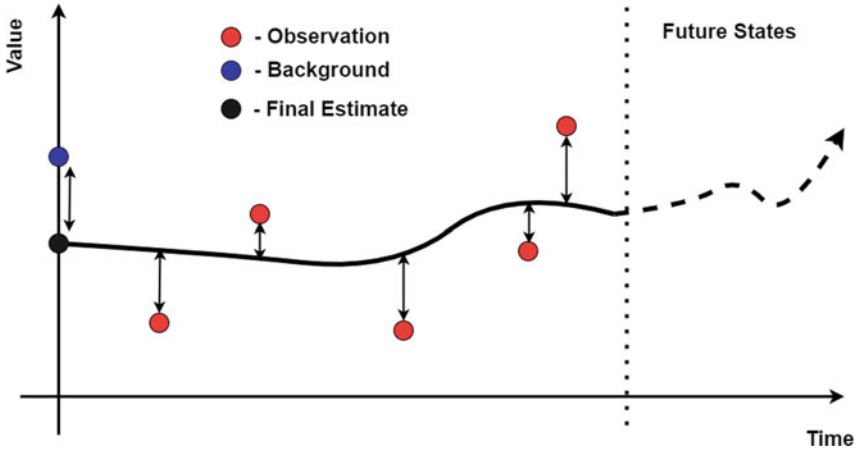


Fig. B.1 Illustration of data assimilation. The final estimate (black) is found by minimising the difference between the observations (red) and the background estimate (blue). This can then be used to generate future forecasts (dotted arrow)

The observation operator $\mathcal{H}_k()$ is worth considering in a little more detail. Note that observations are not at grid points and usually observations are not even of the values of interest (temperature, pressure etc.). Many observations are from satellites which record information across several levels of the atmosphere. This means the observation operator must interpolate to the same location amd also transform into variables which can be compared to the state variables of interest.

An illustration of the data assimilation process is shown for one variable at one grid point in Fig. B.1. The final estimate is found so that its evolved state is closest to accurate observations as well as the prior estimate (the background). The future states are generated by evolving the model even further into the future. The weather prediction models are usually assessed by considering a skill score (Sect. 7.4).

It is worth noting that weather forecasts are not usually updated at every hour. Numerical weather forecast computations are relatively expensive and hence are usually updated once ever six hours. Hence, forecasts at one hour may come from a different model run than another hour.

Reanalysis data is essentially a forecast model equivalent of the observations. This is where the data assimilation process is retrained on the historical observations to create a best estimate of the values of the states at all the grid points. If forecast data is not available then reanalysis data can be useful as an alternative input to a load forecast model, although it should be noted that in a real world application only forecast data would be available.

B.2.2 Preprocessing

Weather data should be subject to preprocessing like most data. Since weather forecasts are defined on a grid this also invites other preprocessing approaches which may not be possible with most time series data. Below we list some ways preprocessing of weather data can be used to improve load forecasts using weather data:

- **Bias correction**: As with any forecasts, weather prediction models may still retain some biases. These should be corrected before utilisation in a load forecast model to reduce the errors in the final model. The calibration of probabilistic weather forecasts can be much more complicated. This topic is investigated in detail in the book [3] which presents several methods for calibrating probabilistic forecasts.
- **Grid point selection**: Since weather forecasts are defined on a grid then feature selection such as LASSO (Sect. 8.2.4) can be used to select which grid point produces the best forecasts.
- **Grid combination**: Just as individual forecast models can be improved by combining them (Sect. 13.1), weather forecasts could potentially be improved by combining across the nearby grid points.
- **Feature engineering within an individual time series**: Exponential smoothing is a useful feature engineering technique since it can take into account the delay effects of temperature on load when used within load forecasts. For building load forecasts this creates a variable that also take into account the thermal inertia. An example of using exponential smoothing for temperature is given in [4].
- **Feature engineering with several different weather variables**: Weather variables can be combined to produce new variables which may be useful for load forecasting. Two common derived values are wind chill and humidity index which can better represent the perceived cold and heat better than temperature alone. For example, wind chill takes into account the combined effect of wind speed and temperature. In cold weather a faster wind speed can make the temperature feel a lot colder than with a slower speed. This higher wind chill may potentially translate to more homes turning on heating and hence higher low voltage demands. Similarly high humidity can translate to high temperatures feeling hotter than when humidity is lower.
- **Feature engineering across the grid**: The grid of data points around a location contain much more information than simply taking an average. The variation across the grid provides further information as well as describes some of the dynamics within the area. Browell and Fasiolo use a feature extraction technique to produce probabilistic net load forecasts in [5]. They derived features such as max, min and standard deviations from the gridded data as inputs to their load forecast models.

Appendix C
Load Forecasting: Guided Walk-Through

This section will consider a forecasting trial and how to select inputs to the models, test some of the different forecasts presented in this book, as well as evaluate their accuracy. The context of this walk-through will be to generate day ahead forecasts (with the forecast origin starting at the beginning of the day). In an ideal situation the forecast models will be retrained at the start of each new day on the extra available data, but for this task only train the data once on the training data (for use on the validation set) and once again on the combined training and validation set (for use on the test set), to reduce the computational cost of constant retraining. Consider other training approaches in future experiments. The steps here will largely follow those presented in Chap. 12.

To run through the assessment select an open dataset. There is a list of data available from here https://low-voltage-loadforecasting.github.io/ but the following are also possible choices:

- Irish Smart Meter data, available from https://www.ucd.ie/issda/data/commission-forenergyregulationcer/. This is one of the most commonly used smart meter data sets.
- London Smart Meter data consisting of half hourly demand data from over 5,500 smart meters in London and is readily downloadable from the kaggle website https://www.kaggle.com/jeanmidev/smart-meters-in-london. It also consists of London weather data for several variables including temperature
- The Global Energy Forecasting Competition (GEFCOM) 2014 data [6]. This data set is at a higher voltage level than smart meter data but does consist of several years of demand including temperature data. Hence although it is smoother and more regular than LV level demand, there is a lot of data for training and testing.

To recreate the analysis in this book, individual smart meter data can be aggregated to simulate LV feeder level data. This will obviously be smoother than the smart meter data but may make the demand easier to predict. Once the data has been collected, first start by exploring patterns and relationships in the data. This is often one of the most important parts of the model development.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5

1. **Quick checks**: Before doing anything perform a quick check that there isn't too many missing values in the data (Sect. 6.1). If the data has too many gaps then cut the data down to a shorter dataset with less than 5% missing values. If using the smart meter data, then choose a few hundred which have at least 95% of their values. The GEFCOM data should be relatively clean.
2. **Split the data**: As shown in Sect. 8.1.3 the data needs to be partitioned into training, validation and test sets. Split the time series into the oldest 60% set of data as the training data, the next 20% as the validation and the final 20% as the test set. Other split ratios can be tested but this is sufficient for this initial study.
3. **Plot the time series**: Plot the training data time series (Sect. 6.2.2), or in the case of the smart meter data, plot several of them to get a better understand of the structures in the data. Notice any patterns, is there annual seasonality? If there is, when is the demand highest and lowest? What about any trends, is the demand increasing or decreasing over time? Zoom in on a few weeks of data, is their daily or weekly seasonality? Are the differences obvious for different days of the week? If there is no obvious seasonality then does it look stationary? In this case, you may want to consider applying a unit root test (see Appendix A).
4. **Seasonal correlations**: For the data with suspected seasonality, plot the autocorrelation and partial autocorrelation plots (see Sect. 3.5). If considering a lot of smart meter data then it won't be practical to check each ACF and PACF plot so instead consider ways of aggregating the information. As was given in the Case Study (Fig. 14.2, Sect. 14.2), you could draw a scatter plot of the ACF and PACF for different lags on the x and y axis which may be significant, e.g. the daily and weekly lags (48 and 336 if considering half-hourly data). This will show a range of different seasonalities. Focus on particular smart meters which have the strongest and weakest correlations and plot their ACF and PACF and compare this to their actual smart meter time series. It may reveal unusual behaviour not previously expected. Alternatively, consider the autocorrelation and partial autocorrelation of the average smart meter profile. When considering the ACF and PACF where are the lags strongest? Is the correlation at lag 336 (a week) larger than the daily lags (48)?
5. **Idenifying anomalous values**: In addition to the missing values (point 1), what are some other anomalous values you can find (Sect. 6.1)? These may have been visible from the time series plot (point 3), showing very large values, or negative values (which perhaps should not be in demand data—unless there is some solar generation connected to the household and the smart meters are showing net demand). You may want to create a simple seasonal model to identify outliers. For example, if there is annual seasonality then for half hourly data, fit a simple seasonal model of the form

$$a + b + \sum_{p=1}^{P} c_p \sin\left(\frac{2\pi pt}{24 \times 365}\right) + d_p \cos\left(\frac{2\pi pt}{24 \times 365}\right), \tag{C.1}$$

where $P = 2$ or 3. Taking the difference between the model and the data should remove the annual seasonality. If there is a linear trend in the series then update (C.1) with additional linear terms. With the resultant residual series, check which points are more than three standard deviations from the mean and select these to be replaced. Of course a more sophisticated model could be chosen using other features discovered in the analysis (for example by including weekly and daily components like the ST model in Sect. 14.2.3) but for now just consider this approach.

6. **Pre-processing**: For the missing and anomalous values identified in point 1 and 5 they should be replaced with appropriate values. For the data with daily/weekly seasonality impute (Sect. 6.1.2) the anomalous and missing values in the training and validation set with averages of the weekly values at the same time period before and after this value. In other words for a missing value at 2 PM on a Tuesday, take the 2 PM on the Tuesday before and the 2 PM the Tuesday after. If there are bigger gaps so that these values are also missing, several runs of the imputation may be required. Since the data chosen is relatively clean this should be sufficient.

The above procedure should produce a relatively clean data set with no missing values which will makes extra analysis much simpler and facilitate creating the forecast models. In addition, the basic analysis above will have highlighted some of the core features of the data, in particular whether the data is stationary and different types of seasonality. The next step is to start a more detailed analysis of other features of the data and walk through choosing a few models to train and test on the hold-out test set. These are described by the following points.

- **Visualisation of explanatory variables**: Start to explore the other relationships in the training set data. The autocorrelations and large patterns and trends in the data have already been considered. If there are other explanatory variables available with the main demand data, such as temperature, then consider some scatter plots, or if there is lots of potential explanatory variables then consider a pair plot (see Sect. 6.2.2). What do the relationships look like? Are they linear, or nonlinear. If they look nonlinear would a simple quadratic or cubic relationship describe it accurately? Do different hours of the day have different levels of seasonality, or stronger correlations with the explanatory variables? Try plotting the cross correlation (Sect. 6.2.3), when do the largest values occur? This will show if some of the lagged values are also important to include in the models. Also consider the difference in the days of the week. Plot an average profile for the different days and see if there is similarities or differences which indicate they should potentially be treated specifically within the models, e.g. through dummy variables (as seen with multiple linear regression in Sects. 9.3 and 14.2.3). When modelling potential relationships consider the adjusted R^2 (Sect. 6.2.3), which variables give the largest value?
- **Initial model selection**: Choose a selection of models which may be suitable based on the data analysis. If the data has strong autocorrelations and linear relationships with explanatory variables then perhaps generate a linear model or ARIMAX type

model (Sect. 9.4). Consider a number of different choices for the explanatory variables: perhaps linear, quadratic and cubic versions. If there are strong seasonalities in the data, then consider the seasonal exponential smoothing models (Sect. 9.2). If there is strong explanatory variables but no seasonal relationship then consider a deep neural networks as in Sect. 10.5 or support vector regression (Sect. 10.2).

- **Benchmark models**: Pick some simple benchmark models based on the features observed in the analysis (see Sect. 9.1 for common benchmarks). If there is seasonalities in the data then consider a seasonal persistence model, or take a seasonal moving average model. If there isn't any seasonality then consider just a simple persistence model. Some of the simpler models chosen in the previous step can also be considered benchmarks. Alternatively base some of the benchmarks on a single explanatory variable. The importance of the benchmarks is to understand some of the main features which are important to the forecast accuracy and suggest potential improvements. Hence simpler benchmarks can be more informative them more sophisticated comparisons. On the other hand comparison to the state-of-the-art can be an extremely useful litmus test for the quality of your forecast.
- **Error measures**: Choose at least one error measure which will best represent the accuracy of your forecast (Chap. 7). If dealing with smart meter data, or data which has many small values then MAPE is unsuitable as the errors will be inflated for smaller values (or not defined!). If you wish to compare the accuracy across several time series then choose relative error measures (such as normalised versions of RMSE or MAE, or even MAPE as long as the values are not small).
- **Training**: Train the selected models (and benchmarks) on the observations in the *training data* (Sect. 8.2), this will also mean training a range of the same type of models (Neural Networks, multiple linear regressions, etc.) with different parameters (for example for neural networks this will mean trying different numbers of nodes and layers, for multiple linear regression using different input variables, possibly including transformations of those variables). There are ways of automatically selecting the inputs or reducing possible overtraining, notably by using regularisation (Sect. 8.2.4) and in the case of likelihood based models, using information criteria (Sect. 8.2.2). However, in this walk-through just consider cross-validation (Sect. 8.1.3) for choosing the hyper-parameters and selecting the final models to test. Most standard packages will train the models according to standard measures/loss functions (for example, multiple linear regression will use least squares—(Sect. 8.2)) however perhaps change the target measure to better fit the error measure chosen in the last part.
- **Model/Hyper-parameter selection**: Use the trained models to forecast on the *validation set* (Sect. 8.1.3) and compare the errors. Select a couple of models from each family which perform the best (have the smallest errors). Keep all the benchmarks, but use this opportunity to see which models appear to have the smallest forecast errors. Which ones are more accurate than the benchmarks, and by how much? This will be interesting to see if there is any change when comparing with the test set. There is usually many models, and variants of the same model, tested on the validation set and therefore there is a possibility a model is the most accurate simply by chance alone.

- **Forecasting**: Retrain the data on the combined training and validation set. Now produce the day ahead forecasts over the test set!
- **Evaluate the results**: Calculate the errors for each method. Now is the time to start to evaluate the results and better understand what are the differences, the similarities, and what are the core features which make one model better than another. Of the most accurate methods what are the common features, are any of these features in the least accurate methods? What features are missing in the worst performing methods? Does the inclusion of particular explanatory variables improve a method? Do different models have different accuracies for different times of the day?

You've now completed a full forecasting trial! However there are several ways to improve the accuracy and quality of your models. Here are some further things to try:

- **Improvements based on the error analysis**: Based on your evaluation of the results is there obvious ways to improve the results? Do any explanatory variables show any improvement when included versus not included? If they are not included in the most accurate model then see if they improve them when added. Is there a feature of the best model that is not in the other models? Perhaps if this feature is included in other models they will have greater accuracy compared to the current best model?
- **Residual analysis**: Plot the residual time series, is there any prominent features in the series, seasonalities or trend? If so then update the model to include this. The residual series should be stationary, check with a unit test (Appendix A). Is there any correlation remaining in the residuals of the most accurate models. If so then, as shown in Sect. 7.5, there is a simple way to improve a forecast by adding further autoregressive components to the models.
- **Combining models**: A common way to improve any individual forecast model is to combine several forecast models together as shown in Sect. 13.1. Generate a new forecast by taking a simple average over some (or all) of the models used in the test set.
- **Feature extraction**: In the above study features were only chosen by comparing different models in the validation set. Now consider more automated methods for selecting the features. Select an ARIMA model, using the Akaike Information Criteria (Sect. 8.2.2). Also consider a Linear Model which uses all available features (and derivations of them) and choose the final variables via a LASSO model (Sect. 8.2.4). Also consider creating models using other methods but with the selected variables from the LASSO.
- **Probabilistic models**: If considering a dataset with lots of historical data, then probabilistic forecasts are valid ways to improve the estimates of the future demand. A host of methods were introduced in Chap. 11. Take the linear model used for the point forecast and use it within a quantile regression for $0.05, 0.1, \ldots, 0.95$ quantiles as described in Sect. 11.4. Residual bootstrapped forecasts are also relatively simple to implement for 1-step ahead forecasts (Sect. 11.6.1). Using the residual errors add adjustments to each forecast at each time step to create a

new realisation of the future demand. Repeat this several hundreds (or preferably thousands) of times to produce a ensemble of forecasts. Now generate empirical quantiles (Sect. 3.4) at each time step of the day to generate another quantile forecast (Sect. 5.2). Probabilistic forecasts require a probabilistic forecast measure such as the Pinball loss score and the continuous ranked probability score (CRPS) as introduced in Chap. 7.

- **Your own investigation**: Outside of this book there is a wealth of models and methods which haven't been covered. To get you started more models and methods can be found from the further reading in Appendix D.1 and D.2.

Appendix D
Further Reading

This book has covered a large number of topics and is an introductory text to forecasting for low voltage electrical demand series. It should provide the reader with sufficient information to design your own trials and implement your own forecasts methods. Whatever your level of knowledge hopefully there is enough methods and techniques to teach you something new, but of course there is so much more research and insightful material out there. This section outlines some further reading which may be useful for expanding on many of the topics in this book as well as other methods which were not discussed.

Note that in some cases, especially concerning energy data, the sources are websites which may be subject to change.

D.1 Time Series Analysis and Tools

Chapters 5–7 covered a wide range of techniques for measuring forecast errors, analysing relationships and extracting features from the data, and methods for model selection. As a general resource, https://robjhyndman.com/hyndsight/ is a highly recommended blog by one of the leading experts in time series forecasting, Prof Rob J Hyndman, which features lots of information on forecasting methodologies and some of the latest research for time series. In addition he has published a free e-book [7][4] which presents further details on times series forecasting principles and techniques, as well as examples of their implementations in R. There are plenty of books out there on time series analysis but the authors have found the book by Ruppert and Matteson [1] to be an excellent resource.

For the interested reader the following is some further reading on some of the specific topics covered in Chaps. 5–7.

[4] Available at https://otexts.com/fpp2/.

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5

- **Detection of outliers** is a vast topic but a sophisticated algorithm for detecting outliers can be found [8] which has an accompanying R package called *stray*.[5]
- **Probabilistic scoring functions** is a rapidly developing field. A detailed and advanced look at these can be found in [9]. A more accessible introduction on the concepts of calibration and sharpness can be found in [10]. A interesting study comparing various probabilistic scoring functions for multivariate data can be found in [11]. An example of using Energy Scores for ensemble forecast evaluation is given in [12] for an offshore wind forecasting application.
- **Bias-variance trade-off** is one of the most important topics in machine learning and forecasting. A popular book on machine learning with a very readable and accessible introduction is [13] which has been made freely available online.[6] This book also has a good overview of many other topics in data science.

D.2 Methods: Time Series and Load Forecasting

Chapters 9–11 has only given a broad overview of forecasting models for time series, especially with regard to probabilistic methods. For the interested readers there is extensive resources available for learning more about time series forecasting, in particular for energy systems. This section will outline some of them. Further reading for forecasting specifically for LV systems will be given in Appendix D.3.

In addition to Rob Hyndman's blog as mentioned in the Appendix D.1, some of the latest research in energy forecasting can be found on Tao Hong's blog, http://blog.drhongtao.com/, in which he posts regularly about topics in energy forecasting as well as about the Global Energy Forecasting Competition (GEFCom)[7] which he co-organises. Tao Hong is also the co-author on a useful introduction to Probabilistic load forecasting [14]. Reference [15] is a great text for implementing machine learning models in python, including accessible introductions to random forest, support vector machines, artificial neural networks, boosting and reinforcement learning. Although not specifically focused on energy, the large open access compendium by Petropoulos et al. [16] provides descriptions of a whole array of topics, and further links to additional literature.

Probabilistic forecast are becoming more common and as a result more packages are becoming available which are specialised to this. For example, ProbCast is an R package which provides a number of probabilistic forecasting methods as well as visualisation and evaluation functionality [17]. This includes implementations of parametric and nonparametric methods and Gaussian copulas.

In the following references there are further details, examples and theory on some of the forecast methods and techniques covered in this book.

---

[5] https://cran.r-project.org/web//packages/stray/index.html.

[6] See https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/.

[7] See http://www.drhongtao.com/gefcom for more details.

- **Exponential smoothing**: An excellent chapter on exponential smoothing can be found in [7]. Holt-Winters-Taylor double seasonal exponential smoothing was first introduced in [18] where it was also applied to short term electricity demand forecasting.
- **ARIMAX methods**: Reference [7] also has an overview of ARIMAX models including the seasonal variants, SARIMA. A detailed investigation into ARIMAX models can also be found in [1].
- **ANN**: Reference [13] provides an detailed introduction to neural networks, including the technique of back-propagation for training them.
- **Classic machine learning methods**: An in-depth overview of classic approaches (though note, not focused on time series forecasting specifically) like additive models, boosting and random forests, support vector machines, and nearest neighbour methods, see the book by Hastie, Tibshirani and Friedman [19].
- **Random forest**: Readers may be interested in reading papers by one of the originators of Random Forests [20]. Examples of them used in short term load forecasting can be found in [21].
- **Support vector regression**: Reference [22] serves as a good introduction and overview to support vector regression.
- **LASSO**: Reference [23] is an excellent example of appling LASSO models to energy forecasting in the GEFCom2014 competition.
- **Generalised Additive Models**: Generalised additive models were the major components of the two winning models in the Probabilistic load forecasting track of the GEFCom2014 competition. The winning paper [24] is definitely worth a look and considers a quantile regression form for the forecast. For the reader interested into diving deeper in GAMs, the introductory book by Simon N. Wood is invaluable [25] who also developed the `mgcv` (Mixed GAM Computation Vehicle with Automatic Smoothness Estimation) package in R. We also recommend the free online resource by Christoph Molnar on "Interpretable Machine Learning" [26] which has an excellent breakdown of GLMs and GAMs and their pros and cons.
- **k-nearest neighbours**: An application for short term load forecasting is given in [27] More specific to the topic of this book this paper gives an example for low voltage demand forecasting [28, 29].
- **Gradient-boosted regression trees:** The tutorial in [30] gives an overview of the basic Gradient Boosting Machine and its most important hyper-parameters. See the documentation of XGBoost[8] and LightGBM[9] for the documentation of the most popular implementations, their interfaces for different programming languages and their respective hyperparameters.
- **Quantile regression and kernel density estimation**: An example of each of these methods including an example of how to combine them to produce an improved forecast is given in [31] for medium term probabilistic load forecasts applied in the GEFCom2014 competition.

[8] https://xgboost.readthedocs.io/en/stable/tutorials/model.html.

[9] https://lightgbm.readthedocs.io/.

- **Copula's and GARCH** models are common in financial applications hence [1] provides an excellent and accessible overview into both techniques and gives many examples of implementations of the methods in R. A comprehensive look at copula's is given in [32].
- **Deep learning approaches**: A more in-depth overview of deep learning models (not focused on time series), is available in the free book by Goodfellow, Bendio and Courville [33]. For more specialised deep learning approaches to time series forecast see the papers on DeepAR [34] and N-BEATS [35] as well as [36, 37].

There was additional techniques and topics discussed in Chap. 13, including model combination and hierarchical forecasting. Below is some additional reading on these topics:

- **Combining Forecasts**: This topic was introduced in Sect. 13.1. Armstrong [38] gives several insights into combining forecasts. An advanced method using copula's to combine forecasts is outlined in [39]. For the application of load forecasting an example of combining probabilistic load forecasts is given in [40]. Rob Hyndman has recently written a review of forecast combination over the last 50 years for those interested in the variety of techniques which are available [41].
- **Statistical Significance Tests**. The Diebold-Mariano test was introduced in Sect. 13.5. A detailed example of using the Diebold-Mariano test for probabilistic forecasts is given in [11]. The paper by Harvey, Leybourne, and Whitehouse [42] dives into significance tests in more detail, including some of the drawbacks and alternative methods available.
- **Hierarchical Forecasting:** An area which has gained increasing interest is hierarchical forecasting. As very briefly introduced in Sect. 13.2, this involves forecasting at different levels whilst considering coherence between them. This topic is considered in [43] for smart meter data. The GEFCom2017[10] included a hierarchical component where the aim was to produce forecasts of zones and then the total load of ISO New England [44]. A brilliant introduction is also available in [7].
- **Special Days**: As discussed in Sect. 13.6.2 special days can have very different behaviour than expected. The authors in [45] present methodologies for forecasting the load on Special days in France.
- **Calibrating Probabilistic Forecasts**: Processing of probabilistic forecasts was only briefly mentioned in Sect. 7.5. These techniques are also important for preprocessing weather forecasts before they are used in load forecasting (Sect. B.2.2). The book [3] presents several methods for calibrating probabilistic forecasts.
- **Other Pitfalls**: The authors of Hewamalagea et al. [46] present common pitfalls and best practice with time series forecast evaluation. This includes benchmarks, error measures, statistical significance, and issues caused by trends, heteroscedasticity, concept shift/drift, outliers and many others.

[10] See http://www.drhongtao.com/gefcom/2017.

D.3 Low Voltage Forecasting Examples

The case study introduced in Chap. 14 has highlighted a real forecast example for low voltage applications. There is numerous other examples of forecasting at the low voltage level, most of it for individual smart meter (household) level demand. Smart meter rollouts means there are many more forecast papers examining household level demand and some of these are listed below. The following review paper by two of the authors of this book investigate the methods, explanatory variables, and applications in LV forecasting [47]. This will help any readers get up to speed with the vast amount of techniques and methods being applied in this area, many of which are described in this book.

The forecast case study Chap. 14 is largely based on a piece of research by one of the authors and the interested readers is encouraged to read the original papers for further details and extra analysis. The low voltage feeder forecast work is based on the paper [48] that also includes the application of various kernel density estimation methods, which were not presented in the Chap. 14.

The literature for low voltage level demand is quite sparse and is mainly focused on aggregations of smart meter data. An interesting implementation of hierarchical probabilistic forecasts is presented in [49, 50] which considers several levels of aggregations of the smart meters. The methods are applied whilst also making the forecasts coherent with each other (ensuring the sum of the forecasts equal the forecast of the aggregation). The example of support vector regression (Sect. 10.2) and random forest regression (Sect. 10.3.2) shared in the storage control in Sect. 15 is based on the paper in [51]. An example of k-nearest neighbours for forecasting low voltage demand is given in [28].

The majority of smart meter forecasting has been for point forecasts but since household demand is quite volatile it can be quite difficult to model accurately. An interesting example of additive models applied to creating probabilistic household forecasts is given in [52], which also includes applications to household batteries. In [53] the authors use partially linear additive models (PLAMs) with an extension to ensure that volatile components of the demand can be modelled. PLAMs are extensions to generalised additive models (Sect. 9.6). Since GAMs are usually restricted to smooth components they aren't necessarily appropriate for household level demand, and is why extensions may be appropriate and necessary. The authors in [54] apply a convolutional neural network (Sect. 10.5) approach to forecasting residential demand.

Due to the volatility of smart meter data, probabilistic forecasting is becoming increasingly common. An excellent example of short term probabilistic smart meter load forecasting for kernel density estimation and probabilistic Holt-Taylor-Winters forecasts is given in [55]. The authors in [56] present an example of producing probabilistic forecasts for smart meters via a form of quantile regression (Sect. 11.4) using a gradient boosted method.

Since peaks are often one of the main interesting features in low voltage demand, an important area of focus is on peak demand forecasting which falls naturally within

the sphere of Extreme Value Theory. An excellent introduction to these methods and their application to low voltage level demand are given in the book by Jacob et al. [57].

Finally for something a little more esoteric the reader is pointed in the direction of [29, 58, 59]. As shown briefly in Sect. 13.3, due to the volatile and spiky nature of smart meter data, peaks may shift in position (someone getting home later from work or university will shift their behaviour accordingly). This means that traditional pointwise methods for measuring errors (like MAPE, MAE and RMSE introduced in Chap. 7 of this book) may not be appropriate due to the 'double penalty effect'. For a peak which is missed slightly (e.g. half an hour too early) will be penalised twice: once for missing the real peak and secondly for the forecast of a peak which didn't occur. The above papers dive further into the adjusted error measure given in Sect. 13.3 and also consider other updates and variants.

D.4 Data and Competitions

The best way to develop a deep understanding about forecasting and applying various techniques is to start coding up and practicing with real data. This section will outline a few publicly available datasets.

One way data is made available is through competitions. In recent years websites such as Kaggle (https://www.kaggle.com/) have hosted competitions for machine learning based problems. Competitions provide opportunities to compete against other participants, trial new methods and learn more about what makes a good forecast. The paper [60] gives a brilliant overview of their history and some of the major learnings from forecasting competitions. Some of the major time series forecasting competitions are

- **The M-Competitions**, one of the first time series forecasting competitions, starting with M1 in 1982. Each competition often has increasing numbers of competitors, complexity and the number of time series. The M4 competition in 2018 consisted of 100, 00 time series to forecast [61].
- **The Global Energy Forecasting Competition**. A forecasting competition started in 2012 focused on energy demand forecasting, the second in 2014 focused on probabilistic energy forecasting [6]. The 2017 competition focused on hierarchical aspects of energy forecasting.

The Global Energy Forecasting Competition review papers are a brilliant resource for learning about some of the state-of-the-art methods in energy forecasting [6, 62]. See also specific papers on some of the methods, particularly in the probabilistic tracks [24, 31]. The data has also been made available for the reader to try their own forecasting methodology (see Tao Hongs blog for this data and others: http://blog.drhongtao.com/2016/07/datasets-for-energy-forecasting.html).

Unfortunately the GEFCom data is typically of a higher voltage than the topic discussed in this book. Hence, although they are good for developing your first load

forecasting methods, the demand is typically much smoother than that which is of interest in this book. Low voltage data is actually much sparser and due to this, publicly available data sets have often been overanalysed, increasing the possibility of them being subject to biases and unrealistic foreknowledge about the dataset.

The authors have compiled a list of LV data sets which may be useful for the readers own investigations.[11] However, below is a specific set of publicly available data contained on that list which may be of use for designing your own models:

1. Irish Smart Meter Data: This is one of the first publicly available sets of smart meter data. It consists of data from about 4000 smart meters. https://www.ucd.ie/issda/data/commissionforenergyregulationcer/.
2. London Smart meter data. Contains half hourly demand data for over 5500 smart meters in London as well as local weather data. https://www.kaggle.com/jeanmidev/smart-meters-in-london.
3. REFIT: High resolution (8 s) electrical load data set for 20 households including appliances. https://pureportal.strath.ac.uk/en/datasets/refit-electrical-load-measurements-cleaned.
4. Open Power System Data Platform: Has data on households data at various resolutions including solar data. Also an important resource for other power system data such as price, weather and demand. https://data.open-power-system-data.org/.
5. UK-DALE: high resolution (6 s) data for five households including individual appliance data. https://jack-kelly.com/data/.
6. GREEND: 1 s resolution data from 8 households. https://sourceforge.net/projects/greend/.
7. Behavioural Energy Efficiency—15min resolution data for 200 households. https://zenodo.org/record/3855575.

Notice that the majority of this data is smart meter/household level since unfortunately there is very little low voltage data. However a estimate of a LV network can be simulated via aggregations of smart meters although it should be noted that these are slightly different and therefore the equivalence isn't exact [48].

References

1. D. Ruppert, D.S. Matteson, *Statistics and Data Analysis for Financial Engineering: With R Examples*. Springer Texts in Statistics (2015)
2. F.M. Dekking, C. Kraaikamp, H.P. Lopuhaä, L.E. Meester, *A Modern Introduction to Probability and Statistics: Understanding Why and How* (Springer, London, 2005)
3. S. Vannitsem, D.S. Wilks, J.W. Messner (eds.) *Statistical Postprocessing of Ensemble Forecasts* (Elsevier, 2018)

[11] See https://low-voltage-loadforecasting.github.io/, where you can also suggest new or missing datasets.

4. T-H. Dang-Ha, F.M. Bianchi, R. Olsson, Local short term electricity load forecasting: automatic approaches, in *2017 International Joint Conference on Neural Networks (IJCNN)* (2017), pp. 4267–4274
5. J. Browell, M. Fasiolo, Probabilistic forecasting of regional net-load with conditional extremes and gridded NWP. IEEE Trans. Smart Grid **12**(6), 5011–5019 (2021)
6. T. Hong, P. Pinson, S. Fan, H. Zareipour, A. Troccoli, R.J. Hyndman, Probabilistic energy forecasting: global energy forecasting competition 2014 and beyond. Int. J. Forecast. **32**, 896–913 (2016)
7. R.J. Hyndman, G. Athanasopoulos, *Forecasting: Principles and Practice*, 2nd edn. (OTexts, Melbourne, Australia, 2018). https://Otexts.com/fpp2. Accessed on July 2020
8. P.D. Talagala, R.J. Hyndman, K. Smith-Miles, Anomaly detection in high-dimensional data (2019)
9. T. Gneiting, A.E. Raftery, Strictly proper scoring rules, prediction, and estimation. J. Am. Stat. Assoc. **102**, 359–378 (2007)
10. T. Gneiting, F. Balabdaoui, A.E. Raftery, Probabilistic forecasts, calibration and sharpness. J. Roy. Stat. Soc.: Ser. B (Stat. Methodol.) **69**(2), 243–268 (2007)
11. F. Ziel, K. Berk, Multivariate forecasting evaluation: on sensitive and strictly proper scoring rules (2019)
12. C. Gilbert, J. Browell, D. McMillan, Probabilistic access forecasting for improved offshore operations. Int. J. Forecast. **37**(1), 134–150 (2021)
13. C.M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)* (Springer, Berlin, Heidelberg, 2006)
14. T. Hong, S. Fan, Probabilistic electric load forecasting: a tutorial review. Int. J. Forecast. **32**(3), 914–938 (2016)
15. A. Gron, *Hands-On Machine Learning with Scikit-Learn and Tensor Flow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st edn. (O'Reilly Media, Inc., 2017)
16. F. Petropoulos, D. Apiletti, V. Assimakopoulos, M.Z. Babai, D.K. Barrow, S. Ben Taieb, C. Bergmeir, R.J. Bessa, J. Bijak, J.E. Boylan, J. Browell, C. Carnevale, J.L. Castle, P. Cirillo, M.P. Clements, C. Cordeiro, F. Luiz Cyrino Oliveira, S. De Baets, A. Dokumentov, J. Ellison, P. Fiszeder, P.H. Franses, D.T. Frazier, M. Gilliland, M.S. Gönül, P. Goodwin, L. Grossi, Y. Grushka-Cockayne, M. Guidolin, M. Guidolin, U. Gunter, X. Guo, R. Guseo, N. Harvey, D.F. Hendry, R. Hollyman, T. Januschowski, J. Jeon, V.R.R. Jose, Y. Kang, A.B. Koehler, S. Kolassa, N. Kourentzes, S. Leva, F. Li, K. Litsiou, S. Makridakis, G.M. Martin, A.B. Martinez, S. Meeran, T. Modis, K. Nikolopoulos, D. Önkal, A. Paccagnini, A. Panagiotelis, I. Panapakidis, J.M. Pavía, M. Pedio, D.J. Pedregal, P. Pinson, P. Ramos, D.E. Rapach, J.J. Reade, B. Rostami-Tabar, M. Rubaszek, G. Sermpinis, H.L. Shang, E. Spiliotis, A.A. Syntetos, P.D. Talagala, T.S. Talagala, L. Tashman, D. Thomakos, T. Thorarinsdottir, E. Todini, J.R. Trapero Arenas, X. Wang, R.L. Winkler, A. Yusupova, F. Ziel, Forecasting: theory and practice. Int. J. Forecasting **38**(3), 705–871 (2022)
17. J. Browell, C. Gilbert, Probcast: open-source production, evaluation and visualisation of probabilistic forecasts. 5 (2020)
18. J.W. Taylor, Short-term electricity demand forecasting using double seasonal exponential smoothing. J. Oper. Res. Soc. **54**, 799–805 (2003)
19. T. Hastie, R. Tibshirani, J. Friedman *Data Mining, Inference, and Prediction*. *The Elements of Statistical Learning*. Springer Series in Statistics (2009)
20. L. Breiman, Random forests. Mach. Learn. **45**(1), 5–32 (2001)
21. G. Dudek, *Intelligent Systems'2014. Advances in Intelligent Systems and Computing*. Short-Term Load Forecasting Using Random Forests, vol. 323 (Springer, Cham, 2015), , pp. 821–828
22. A.J. Smola, B. Schölkopf, A tutorial on support vector regression. Stat. Comput. **14**, 199–222 (2004)
23. F. Ziel, B. Liu, Lasso estimation for gefcom2014 probabilistic electric load forecasting. Int. J. Forecast. **32**(3), 1029–1037 (2016)
24. P. Gaillard, Y. Goude, R. Nedellec, Additive models and robust aggregation for gefcom2014 probabilistic electric load and electricity price forecasting. Int. J. Forecast. **32**(3), 1038–1050 (2016)

25. S.N. Wood, *Generalized Additive Models: An Introduction with R*, 2nd edn. (Chapman and Hall/CRC, 2017)
26. C. Molnar, *Interpretable Machine Learning: A Guide For Making Black Box Models Explainable* (Independently published, 2022)
27. A.T. Lora, J.M. Riquelme Santos, J. Cristóbal Riquelme, A. Gómez Expósito, J. Luís Martínez Ramos, Time-series prediction: application to the short-term electric energy demand, in *Current Topics in Artificial Intelligence* ed. by R. Conejo, M. Urretavizcaya, J-L. Pérez-de-la Cruz (Springer, Berlin, Heidelberg, 2004) pp. 577–586
28. O. Valgaev, F. Kupzog, H. Schmeck, Low-voltage power demand forecasting using k-nearest neighbors approach, in *2016 IEEE Innovative Smart Grid Technologies - Asia (ISGT-Asia)* (2016), pp. 1019–1024
29. M. Voß, A. Haja, S. Albayrak, Adjusted feature-aware k-nearest neighbors: utilizing local permutation-based error for short-term residential building load forecasting, in *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)* (IEEE, 2018), pp. 1–6
30. A. Natekin, A. Knoll, Gradient boosting machines, a tutorial. Front. Neurorobot. **7**, 21 (2013)
31. S. Haben, G. Giasemidis, A hybrid model of kernel density estimation and quantile regression for gefcom2014 probabilistic load forecasting. Int. J. Forecast. **32**, 1017–1022 (2016)
32. D. Kurowicka, R. Cooke, *High-Dimensional Dependence Modelling*, Chap. 4 (Wiley, 2006), pp. 81–130
33. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, 2016). http://www.deeplearningbook.org
34. D. Salinas, V. Flunkert, J. Gasthaus, T. Januschowski, Deepar: probabilistic forecasting with autoregressive recurrent networks. Int. J. Forecast. **36**(3), 1181–1191 (2020)
35. B.N. Oreshkin, D. Carpov, N. Chapados, Y. Bengio, N-beats: Neural basis expansion analysis for interpretable time series forecasting (2019). arXiv:1905.10437
36. A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, Wavenet: a generative model for raw audio (2016). arXiv:1609.03499
37. S. Bai, J. Zico Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling (2018). arXiv:1803.01271
38. J. Scott Armstrong, *Principles of Forecasting: A Handbook for Researchers and Practitioners* (Springer, 2001)
39. Copulas-based time series combined forecasters, Inf. Sci. **376**, 110–124 (2017)
40. Y. Wang, N. Zhang, Y. Tan, T. Hong, D.S. Kirschen, C. Kang, Combining probabilistic load forecasts. IEEE Trans. Smart Grid **10**, 3664–3674 (2019)
41. X. Wang, R.J. Hyndman, F. Li, Y. Kang, Forecast combinations: an over 50-year review (2022)
42. D.I. Harvey, S.J. Leybourne, E.J. Whitehouse, Forecast evaluation tests and negative long-run variance estimates in small samples. Int. J. Forecast. **33**(4), 833–847 (2017)
43. S. Ben Taieb, J.W. Taylor, R.J. Hyndman, Hierarchical probabilistic forecasting of electricity demand with smart meter data. J. Am. Stat. Assoc. **0**(0), 1–17 (2020)
44. Global energy forecasting competition 2017: Hierarchical probabilistic load forecasting. Int. J. Forecast. **35**(4), 1389 – 1399 (2019)
45. S. Arora, J. Taylor, Rule-based autoregressive moving average models for forecasting load on special days: a case study for France. Eur. J. Oper. Res. **266**, 259–268 (2017)
46. H. Hewamalage, K. Ackermann, C. Bergmeir, *Common Pitfalls and Best Practices, Forecast Evaluation for Data Scientists* (2022)
47. S. Haben, S. Arora, G. Giasemidis, M. Voss, D. Vukadinović Greetham, Review of low voltage load forecasting: methods, applications, and recommendations. Appl. Energy **304**, 117798 (2021)
48. S. Haben, G. Giasemidis, F. Ziel, S. Arora, Short term load forecasting and the effect of temperature at the low voltage level. Int. J. Forecast. **35**, 1469–1484 (2019)
49. S. Ben Taieb, R.J. Hyndman, Hierarchical probabilistic forecasting of electricity demand with smart meter data (2017)

50. S. Ben Taieb, J.W. Taylor, R.J. Hyndman, Hierarchical Probabilistic Forecasting of Electricity Demand With Smart Meter Data (2017), pp. 1–30
51. T. Yunusov, G. Giasemidis, S. Haben, *Smart Storage Scheduling and Forecasting for Peak Reduction on Low-Voltage Feeders* (Springer International Publishing, Cham, 2018), pp. 83–107
52. C. Capezza, B. Palumbo, Y. Goude, S.N. Wood, M. Fasiolo, Additive stacking for disaggregate electricity demand forecasting. Ann. Appl. Stat. **15**(2), 727–746 (2021)
53. U. Amato, A. Antoniadis, I. De Feis, Y. Goude, A. Lagache, Forecasting high resolution electricity demand data with additive models including smooth and jagged components. Int. J. Forecast. (2020)
54. M. Voss, C. Bender-Saebelkampf, S. Albayrak, Residential short-term load forecasting using convolutional neural networks, in *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)* (2018), pp. 1–6
55. S. Arora, J. Taylor, Forecasting electricity smart meter data using conditional kernel density estimation. Omega **59**, 47–59 (2016)
56. S. Ben Taieb, R. Huser, R.J. Hyndman, M.G. Genton, Forecasting uncertainty in electricity smart meter data by boosting additive quantile regression. IEEE Trans. Smart Grid **7**(5), 2448–2455 (2016)
57. M. Jacob, C. Neves, D. Vukadinović Greetham *Forecasting and Assessing Risk of Individual Electricity Peaks* (Springer International Publishing, Cham, 2020)
58. S. Haben, J.A. Ward, D.V. Greetham, P. Grindrod, C. Singleton, A new error measure for forecasts of household-level, high resolution electrical energy consumption. Int. J. Forecast. **30**, 246–256 (2014)
59. N. Charlton, D.V. Greetham, C. Singleton, Graph-based algorithms for comparison and prediction of household-level energy use profiles, in *2013 IEEE International Workshop on Intelligent Energy Systems (IWIES)* (2013), pp. 119–124
60. R.J. Hyndman, A brief history of forecasting competitions. Int. J. Forecast. **36**, 7–14 (2020)
61. S. Makridakis, E. Spiliotis, V. Assimakopoulos, The m4 competition: results, findings, conclusion and way forward. Int. J. Forecast. **34**, 802–808 (2018)
62. T. Hong, P. Pinson, S. Fan, Global energy forecasting competition 2012. Int. J. Forecast. **30**(2), 357–363 (2014)

Index

S. Haben et al., *Core Concepts and Methods in Load Forecasting*,
https://doi.org/10.1007/978-3-031-27852-5