Expressive and Modular Rule-Based Classifier for Data Streams

Duyen Thien Le

Submitted in partial fulfilment of the requirements of
University of Reading for
the degree of Doctor of Philosophy

Department of Computer Science
January 2019

Abstract

The advances in computing software, hardware, connected devices and wireless communication infrastructure in recent years have led to the desire to work with streaming data sources. Yet the number of techniques, approaches and algorithms which can work with data from a streaming source is still very limited, compared with batched data. Although data mining techniques have been a well-studied topic of knowledge discovery for decades, many unique properties as well as challenges in learning from a data stream have not been considered properly due to the actual presence of and the real needs to mine information from streaming data sources. This thesis aims to contribute to the knowledge by developing a rule-based algorithm to specifically learn classification rules from data streams, with the learned rules are expressive so that a human user can easily interpret the concept and rationale behind the predictions of the created model. There are two main structures to represent a classification model; the 'tree-based' structure and the 'rule-based' structure. Even though both forms of representation are popular and well-known in traditional data mining, they are different when it comes to interpretability and quality of models in certain circumstances.

The first part of this thesis analyses background work and relevant topics

in learning classification rules from *data streams*. This study provides information about the essential requirements to produce high quality classification rules from data streams and how many systems, algorithms and techniques related to learn the classification of a static dataset are not applicable in a streaming environment.

The second part of the thesis investigates at a new technique to improve the efficiency and accuracy in learning heuristics from numeric features from a streaming data source. The computational cost is one of the important factors to be considered for an effective and practical learning algorithm/system because of the needs to learn from continuous arrivals of data examples sequentially and discard the seen data examples. If the computing cost is too expensive, then one may not be able to keep pace with the arrival of high velocity and possibly unbound data streams. The proposed technique was first discussed in the context of the use of Gaussian distribution as heuristics for building rule terms on numeric features. Secondly, empirical evaluation shows the successful integration of the proposed technique into an existing rule-based algorithm for the data stream, eRules.

Continuing on the topic of a rule-based algorithm for classification data streams, the use of Hoeffding's Inequality addresses another problem in learning from a data stream, namely how much data should be seen from a data stream before starting learning and how to keep the model updated over time. By incorporating the theory from Hoeffding's Inequality, this study presents the Hoeffding Rules algorithm, which can induce modular rules directly from a streaming data source with dynamic window sizes throughout the learning period to ensure the efficiency and robustness towards the concept drifts.

Concept drift is another unique challenge in mining data streams which the underlying concept of the data can change either gradually or abruptly over time and the learner should adapt to these changes as quickly as possible.

This research focuses on the development of a rule-based algorithm, Hoeffding Rules, for data stream which considers streaming environments as primary data sources and addresses several unique challenges in learning rules from data streams such as concept drifts and computational efficiency. This knowledge facilitates the need and the importance of an interpretable machine learning model; applying new studies to improve the ability to mine useful insights from potentially high velocity, high volume and unbounded data streams. More broadly, this research complements the study in learning classification rules from data streams to address some of the unique challenges in data streams compared with conventional batch data, with the knowledge necessary to systematically and effectively learn expressive and modular classification rules from data streams.

Acknowledgements

The work and content of this thesis have been shaped and supported by the limitless encouragement of a whole host of people, to whom I wish to express my deepest appreciation.

First of all, I would like to acknowledge the trust and unconditionally supportive guidance of my supervisor, Dr. Frederic Stahl, throughout the journey of my PhD programme. His expertise, a countless number of timeless discussions as well as his extensive knowledge in the domain have motivated and assisted me to materialise my research and contributions to the domain and the community.

A very special thank you to my beloved parents for their love, continuous support and encouragement.

Also, I appreciate the co-authors of my publications during my research for their permission and collaboration to use extracts from the papers.

Finally, I am sure that many other people should be acknowledged. It is not possible to name everyone, but to my friends, colleagues and those who supported me in any form, thank you so much.

Original Authorship

Declaration: I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Duyen Thien, Le

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Research Context and Motivation

Volume, velocity, variety and veracity are defining the new dimensions of digital data that the world is generating and relying on every day [77, 96, 37]. From traffic pattern and real-time monitoring of fleets of vehicles [80] to real-time data sensing in the chemical process industry using soft-sensors [79], digital data is being created, stored and analysed at the fastest rate ever. Simply put, the philosophies and the abilities to understand and learn from digital data have evolved rapidly, and are even considered to be a "Big Data Revolution", (Professor Gary King, Weatherhead University) [81].

However, the stated "revolution" does not solely lie with large scale storage and computational power, but the statistical and algorithmic methods to ingest, learn and adapt to the available data. The desire to extract knowledge from high velocity and possibly infinite flows of data in real-time has led to the growing field of streaming data analytic model, which is also known

as the key area of Data Stream Mining (DSM) [55]. Insights from streaming data sources can produce a major source of information for decision making. Since the early 2000s, many DSM techniques have been proposed, for a recent review and categorisation of streaming mining techniques/algorithms, one of which is referred to an advanced review of data streaming mining by Gaber [52].

While it is not possible to cover every area within knowledge discovery and data mining within the scope of this research, the main topics form the fundamentals of this research are as follow:

Streaming Data. The definition of streaming data has become more apparent in recent years due to the explosion of technologies, wireless communication network, and connected devices (Internet of Things). Large volume and high velocity are the two key drivers to creating changes in perception related to knowledge discovery from raw data, as well as rules of some developed algorithms and systems for data mining. Traditional algorithms and systems for static data are not designed to deal with sequential characteristics of streaming data such as unbounded, sequential order, concept drift, and online arriving. These challenges are the underlying rationales for this research to move forward in a direction that allows useful and actionable information to be derived from streaming data sources, as well as incorporating some well-researched studies from the data mining community into streaming environments.

Expressiveness of Classification Models is still an area of research that has attracted less attention in data stream mining, despite its practical importance, and many critical applications, which still need to interpret

decisions from a machine learning model by a domain expert. In particular, the idea of having an expressive model from streaming data in real-time is even more limited in practice, as well as in the literature, to the best of the author's knowledge. **Chapter 2** outlines some important remarks for tree-based and rule-based algorithm for a classification model in both static and streaming environments.

Rule-Based Representation for Classification Models. Knowledge discovery from data has been a well-studied topic for the last few decades as well as different ways to represent the learned concepts from raw data. In particular for classification tasks, tree or hierarchical structures have been the primary method of encoding learned data but, the tree structure is also prone to some drawbacks as raised in **Chapter 2**. Moreover, rule-based models are another well-known alternative to the tree-based model because of its modular and expressive structure. The aforementioned properties of rule-based models are even more valuable factors when considering the suitability of model representation for learning from streaming data sources, which can help decision makers with informed predictions (white-box).

Numeric Features in Streaming Data. An algorithm sees the input vector in data in the form of numeric and categorical features. Depending on the underlying information, it may be more appropriate to represent the input data as either categorical or numeric feature. For example, information such as colours, nationalities, ethics, and names can be naturally appropriately represented as categorical features. On the other hand, information such as measurements, age, height, and distance can be most appropriately considered numeric features. While many well-established algorithms for

classification can work with both categorical and numeric features out-of-the-box, there are challenges for working efficiently and effectively with numeric features in streaming environments, because the data is assumed to be unbounded and the underlying distribution can change over time. A stream centric approach to working with numeric features is proposed along with empirical evaluations to investigate the possibility of increasing the overall computational efficiency but still being competitive in terms of accuracy for rule-based classification algorithms.

Dynamic Sliding Window. Due to the continuous and unbounded characteristics of streaming data, the sliding window technique is known as a common technique to learn from streaming data and has been widely adapted by well-known studies related to streaming data such as [72], [115], [55], and [94]. Traditionally, the learning process starts when data collection is completed, and the learned model aims to reflect the concept encoded in the collected data. However, in a streaming environment, a concept evolves over time and a valid model at a given point in time does not necessarily imply validity at a later point in time. Therefore, the sliding window technique means continuously learning and evaluating data examples from recent history and then continuously adapting to any change in underlying data over time. Robust and dynamic window size are essential to the success in learning from streaming data, but the challenge of getting the right size for a window of historical data is one to be investigated on. Hoeffding's Inequality was studied in this research, and it shows a great promise, along with empirical validations to incorporate the equality with sliding window technique to create dynamic windows sizes throughout the learning time, with competitive

accuracy.

Research Motivation

An important method of data stream mining is the classification of unseen data examples. Traditionally, a classification model is built on static batch training sets, allowing several iterations over the data. However, streaming data sources bring new challenges to the well-established classification algorithms such as C4.5 [104, 105], Rainforest [61], RIPPER [30], AQ [93], or CN2 [28] because a streaming data source can be very fast and infinite. Hence, algorithms that only require one pass through the data are necessary.

Moreover, the existence of concept drift, changes of pattern encoded in the data over time, is more apparent and of concern in the streaming environment. Having concept drift as a primary learning requirement, a streaming classifier is required to be able to detect such concept drifts and adapt to reflect the current concept in the data as accurately as possible.

Most data stream classification techniques are based on the 'Top Down Induction of Decision Tree' (TDIDT), also known as the 'Divide-and-Conquer' approach [115]. However, a hierarchical format such as a decision tree is a major drawback and often requires irrelevant information to be available to perform a classification task [26]. The complexity and exposed bureaucracies of tree-based models as in [123] show that on a large dataset and streaming data source, a tree-based model limits the ability to reveal the expressiveness of the captured concept, wherein humans can easily interpret the model/concept. The hierarchical structure of tree-based models also poses a challenge

to adapt to concept drift in real time.

The expressiveness of decision models in the data stream is an area of research that has attracted less attention, despite its paramount practical importance. Output expressiveness increases trust in streaming data analytics, which is one of the challenges facing adaptive learning systems. Particularly, the Local Interpretable Model-Agnostic Explanations (LIME) algorithm [107] was proposed to generate a short explanation for each new classified or regressed instance out of a prediction model, in a state in which the information can easily be understood by humans (possibly expressed as rules, in the same way). LIME itself is not a learner that creates a model from a given set of training data examples, but an express technique that aims to explain the predictions from any classifier. Also, LIME does not consider the environment how the train model was created (static or streaming); rather the predictions from the trained model.

Unsurprisingly, the work attracted a great deal of media attention and has emphasised the need for expressive and interpretable models. The model's trust has been concerned and further emphasised.

1.2 Research Questions

To advances the knowledges for classification tasks in streaming environment, the following research questions are posed:

Research Question 1: *How can a trained predictive model from a streaming data source be reliably interpreted and understood by the users?*

Accuracy has been the dominating measurement of interest in comparing

classifiers in both static and streaming environments, it is evident that real-time decision making based streaming models still suffer from the issue of trust.

By looking to develop a rule-based technique, a user can determine an accuracy loss, such that a model can be expressive enough to grant trust, and at the same time, the shortfall in accuracy can be tolerated compared with any other best performing classifiers which are less expressive or can even be completely black box.

Research Question 2: *What are the most efficient approaches to improve the efficiency in learning heuristic for numeric features from a high-volume and high-velocity streaming data source?*

To date, there is no single optimised rule-based method to deal with numeric features. However, methods for working with numeric features have been studied extensively in the past two decades for static/batch data environments. These methods suffer performance in learning or even have no ability to capture learning insights from a streaming data source.

Research Question 3: *What impact have the properties of streaming data sources had on the faith and accuracy of the predictions for classification tasks?*

Abstaining is desirable and necessary in critical applications where misclassification is very costly, such as in medical or financial applications. Tree-based techniques always return a prediction for a given data example because of its hierarchical structure. However, a model that can draw a clear line between objective and random subjective prediction is highly desirable.

Research Question 4: *How have concept drifts and dynamic behaviours in streaming data been posing challenges in learning a reliable and accurate classification model?*

The key challenge of streaming data classification lies in the need of the classifier to adapt in real-time to concept drifts, which is significantly more challenging if the data stream is high velocity.

By looking at classification in data stream environments, this research does not stop at the assumption of early machine learning techniques which typically expect to work with a complete set of training data examples, but even more demanding and infinite data sources. An example of such an application is the monitoring of a network of sensors to determine whether the unbounded and sequential flow of data may be too large to consider storing and revisiting as being impractical.

1.3 Aim and Objectives

The primary aim of the research is to *investigate common issues across most rule-based techniques in data stream mining and develop a set of techniques as a system to improve the efficiency and feasibility of predictive learning from streaming data sources.*

To achieve this aim, this research focuses on a number of investigations, experimental studies and empirical evaluations that involve discussing the following proposition:

Develop a rule-based algorithm which can induce modular classi-

fication rules from a streaming data source in real time with the ability to adapt to concept drifts. Expressive rule-based algorithms have been well studied in last decades, but were not designed to understand and deal with challenges from high volume and high velocity streaming data.

A classification rule set is a way to represent models for classification. A rule-based classifier typically stores the model as a set of 'IF - THEN' rules of the form:

IF condition(s) THEN conclusion

The 'IF' part (left-hand side) of a rule, is typically referred to as the precondition(s), whereas the 'THEN' part (right-hand side) is referred to as a consequence.

Research Objectives

Following research objectives would facilitate the achievement of the aforementioned research aim:

- **Objective 1:** To understand the background of learning a predictive model from data streams.
- **Objective 2:** To assess the limitations of existing methods in learning predictive model from streaming data source.
- **Objective 3:** To investigate and develop techniques to overcome the unique challenges in learning predictive model from streaming data sources.

- **Objective 4:** To identify factors that affect the reliability of predictions for classification tasks in streaming environments.
- **Objective 5:** To develop a set of techniques as a system to learn modular and expressive classification rules from streaming data sources.

Objective 1, **2**, **3** and **4** are designed to address important aspects and significant challenges in learning predictive models for classification tasks in streaming environment. Altogether, they are the foundation to **Objective 5** to deliver the defined aim of the research.

1.4 Research Methodology

This section focuses on reflecting the nature of this research and the required methods to tackle the research area properly. It discusses the paradigm of the research and identifies the characteristics of the research methods which facilitate the research to be conducted in a proper way and recognised manner. Chosen methods and procedures for data sources, validation, and evaluation are described that are used in conducting this research.

1.4.1 Adopted Research Paradigm and Philosophical Stands

Generalising findings from results of some experiments is the primary goal of research which contributions/outcomes could be used or applied beyond the area under investigation [47, 119]. Therefore, the primary purpose of any research remains the same, no matter of in what branch and category of the

science it focuses on. However, depending on the branch and science, the way, which the scientists see a phenomenon and raise questions about it, is significantly differs. From this perspective, there is a number of recognised paradigms of research among the scholars such as positivism, constructivism/interpretivism, and critical theory [39, 63].

First, the positivism paradigm is a methodological philosophy in quantitative research where the methods of natural sciences are applied to discover the study of social science [36]. In this respect, any understanding of a phenomena needs to be evidently measured and supported by official statistics.

Next, to the interpretivism paradigm, the methods of interpretivism support the understanding of knowledge related to human and social sciences which cannot be the same as the methods used in physical sciences because humans interpret their acts and the world based on such interpretations while the world does not [75]. Therefore, within the interpretivism paradigm, a single phenomenon may have more than one interpretation rather than a truth that can be realised by process of measurement.

Finally, critical theory is also known as the "transformative paradigm" [109] where the ontology is based on relativism. From this aspect, critical researchers intentionally use the ethical, political standard and moral to judge the situation and practice their research with the consideration of social, political, economic and cultural context for specific research's objectives [109].

This research is in the stream of computing in general, and computer science in particular. Research in computing discipline typically contains two aspects, science and engineering [66]. For this reason, there are two primary

dimensions for discussion in Computing, *theoretical* and *experimental*. From the paradigmatic perspective, this research falls in the positivism paradigm. Accordingly, this research focuses on using quantitative methods to address the raised research questions.

1.4.2 Research Methods

To realise the objectives of this research, a quantitative research approach was held. There are several specific methods in Computer Science which typically has its base in logic and mathematics. Computer Science is also regarded as an empirical discipline, in which a written piece of software (code) can be seen as an experiment, the structure and behaviour of which could be realised [95]. There are also well-established methods which are suitable and specific to Computer Science discipline [13]:

Experimental method shows the experiments that will occur in order to extract outputs from real-world implementation. The veracity of theories can be learned and proven by the experiments. This method is also popular in several different fields such as automating theorem, Natural Languages Processing (NLP), analysing performance and behaviour, artificial neural network, etc. It is essential to make sure that the created experiments can be reproducible and independently verified.

Simulation method is very much suitable in Computer Science because it offers a possibility to investigate systems or hypothesise that are outside of the experimental domain. Some fields that also adopt computer simulation methodology are physics, economics, and astronomy.

Theoretical method can probably be regarded as the most classical methodology, which also suits Computer Science discipline since it is related to logic and mathematics. There is a good evidence about the inheritance of theoretical method in Computer Science such as popularity of *iteration*, *recursion* and *induction*. A theory is essential to develop logic and semantic models and the reason for the programs to demonstrate the correctness. Nevertheless, the theoretical method can help in finding new mathematical models or theories, but this method still needs other methods to prove and facilitate reasonable proofs.

In order to discover and disseminate new knowledge, a multi-method approach is employed in this research. The approach involves proposing novelties from an existing mathematical concept (*Theoretical method*) and empirically evaluating (*Experimental method*) the proposed concepts. The combination of which provides a comprehensive pipeline in learning the new novelties as well as deeper empirical understandings of the problem area.

1.4.3 Data Sources and Gathering

For this research, both real life and synthetic datasets are used for evaluating classification tasks for data streams. The KDD [68] and UCI ML [12] repositories have the most common used datasets for benchmarking machine learning algorithms, but many of these datasets are not suitable for evaluating data stream classification. The *Forest Covertype* dataset is one of the largest with about 600,000 data examples, and it is suitable to be used to simulate a data stream. Synthetically generated data is also used for stream-

ing evaluation as it is easier to produce, and there is a little cost in term of storage and transport. Examples include the Random Tree Generator [40] and SEA Generator [116]. The using of both synthetic and real-time to provides different experimental settings which are important for the stated research questions because the faith of a prediction is important to be verified with real-life dataset and the challenges of streaming environment can be simulated with synthetic data sources respectively.

Data Controls

During the evaluation, the data is generated on-the-fly (for both real life datasets or synthetic feeders). This aspect directly influences the number of training data examples that can be supplied at any given time period. Therefore, the number of features and learning time of an algorithm will also be important metrics for evaluating between algorithms. The learning time of an algorithm can be regarded as a direct metric for the efficiency of learning a model from a data stream, which is the primary concern in *Research Question 2*.

1.4.4 Evaluation Practices and Procedures

The accuracy of a learning algorithm is the critical variable to be measured. Having an optimal error rate, or equivalently its converse accuracy, may not be the only concern, but it is typically the most important one. Having a reliable estimation of accuracy can enable comparison of different methods so that the best available method for a given problem could be determined.

However, classification tasks in a streaming environment have different requirements from traditionally static data. Instead of maximising the use of a collected dataset, the focus in evaluating classification tasks from a streaming data source shifts to trends over time (concept drifts) and the response time.

Prequential or 'Interleaved Test-Then-Train' procedure is adopted in this research for evaluating classification algorithms for data streams. Each data example is used to test the model before it is used for training, and from this, the error rate can be incrementally minimised and monitored over time. This scheme is particularly suitable to evaluate streaming algorithms because it makes maximum use of available data as well as measuring both time and accuracy.

1.5 The Organisation of the Thesis

This thesis is organised into seven chapters which aim to support the fundamental rationales behind the contributions in knowledge to the domain. **Figure 1.1** provides an overview of how the research questions were investigated and answered in each chapter and how they were brought together to deliver the aim of the research collectively.

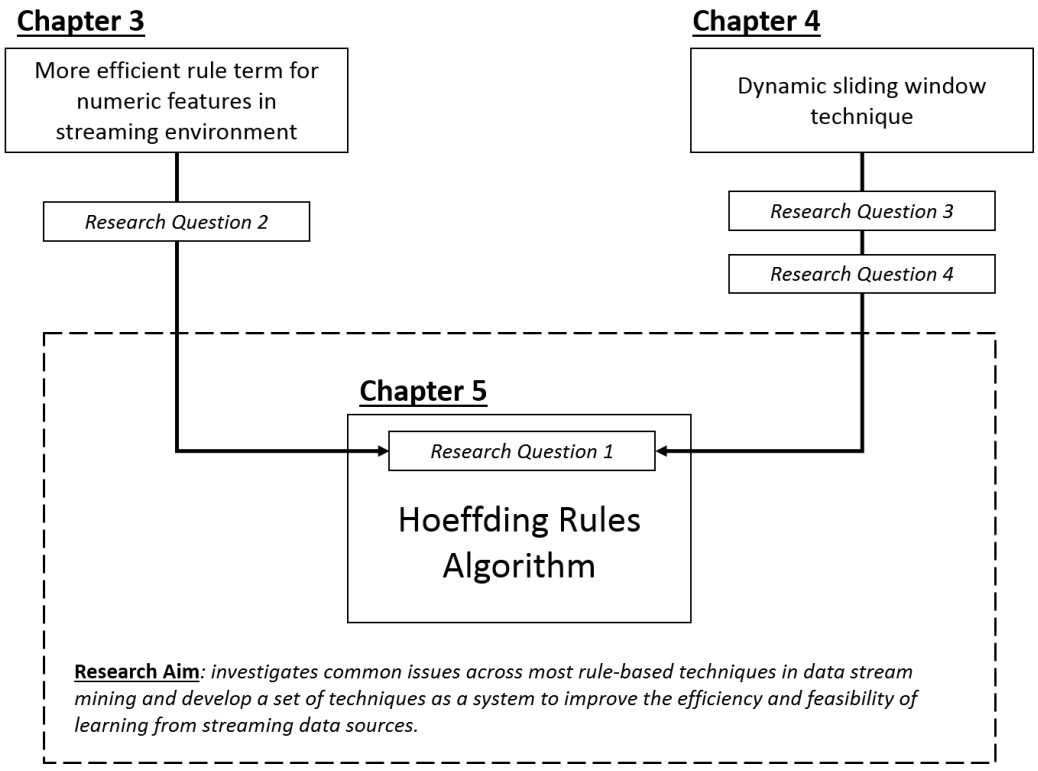


Figure 1.1: Overview of how research questions are answered to deliver the defined research aim.

Chapter 2 introduces essential background in different forms of representation for predictive learning in data mining and characteristics for each type of representation in a streaming environment. Limitations of existing classification algorithms for data streams were evaluated and compared with others. In this chapter, there is also a comparison between traditional methods for evaluating classification algorithms with a streaming oriented method, prequential.

Chapter 3 develops a new technique to overcome the arising challenges in learning heuristic from numeric features from data streams. The technique

addresses the unique challenges such as high-velocity and high-volume data in a streaming environment. G-eRules algorithm was developed from the integration of the proposed technique and an existing rule-based algorithm, eRules. G-eRules algorithm shows a significant improvement in learning efficiency while maintaining competitiveness in term of accuracy.

Chapter 4 proposes a method that focuses on the challenges in buffering historical data to retain, and continuously learn from a potentially infinite data stream, yet optimising the required buffering space for historical data. The method incorporates the use of Hoeffding's Inequality with the sliding window technique to dynamically and reliably buffer recent data examples for classification learning tasks.

Chapter 5 develops Hoeffding Rules algorithm, an expressive and robust rule-based classification algorithm for streaming data on top of the works from **Chapter 3** and **Chapter 4**.

Chapter 6 presents empirical evaluations and discussions of the proposed technique in **Chapter 3** and the Hoeffding Rules algorithm in **Chapter 5**. The empirical experiments validate the novelty contributions of the research in relation to the raised research questions under a controlled and standardized framework. Subsequently, some limitations of Hoeffding Rules algorithm are realised and discussed.

Chapter 7 concludes the research by drawing together all aspects of the study and findings from the empirical evaluations. The contributions of this research along with the consideration for future direction are presented.

Moreover, each individual chapter also provides an overview of related work to provide a more focused attention for the key concepts as well as an informative comparison of the contributions of this thesis in relation to the domain.

Chapter 2

Background of Classification Tasks for Data Streams

This chapter provides an overview and related backgrounds regarding the topics in predictive learning from data streams in relation to the main proposition of the research:

> "*This research investigates common issues classification tasks in data stream mining and introduces a set of techniques as a system to improve the efficiency and feasibility of learning from streaming data sources.*"

This research focuses on modular classification rules, learning from streaming data sources, a number of proposed methods and ideas in this research have been coined within other paradigms of knowledge discovery, e.g., model representation or numeric features handling.

This chapter first introduces the different model representation techniques and paradigms used in streaming related works/researches (**Section 2.1** and **Section 2.2**). Limitations of tree-based model for data streams are described in **Section 2.3**.

Next, the chapter focuses on existing works and algorithms in learning predictive models from streaming data sources in **Section 2.4**.

Section 2.5 describes ways in which designed algorithms and methods for streaming data can be empirically evaluated.

Finally, **Section 2.6** highlights the limitations of existing algorithms for classification tasks in relation to different dimensions of interest for a performant algorithm for classification tasks in a streaming environment.

2.1 Data Model Representation

Data mining is a task to learn and discover hidden knowledge and information from available data. The patterns or concepts of the data need to be expressed in a form or structural pattern within which they can be understood and worked with [21, 94].

Figure 2.1 shows a hierarchy of output expressiveness, with rule-based models being at the top of all the other classification techniques.

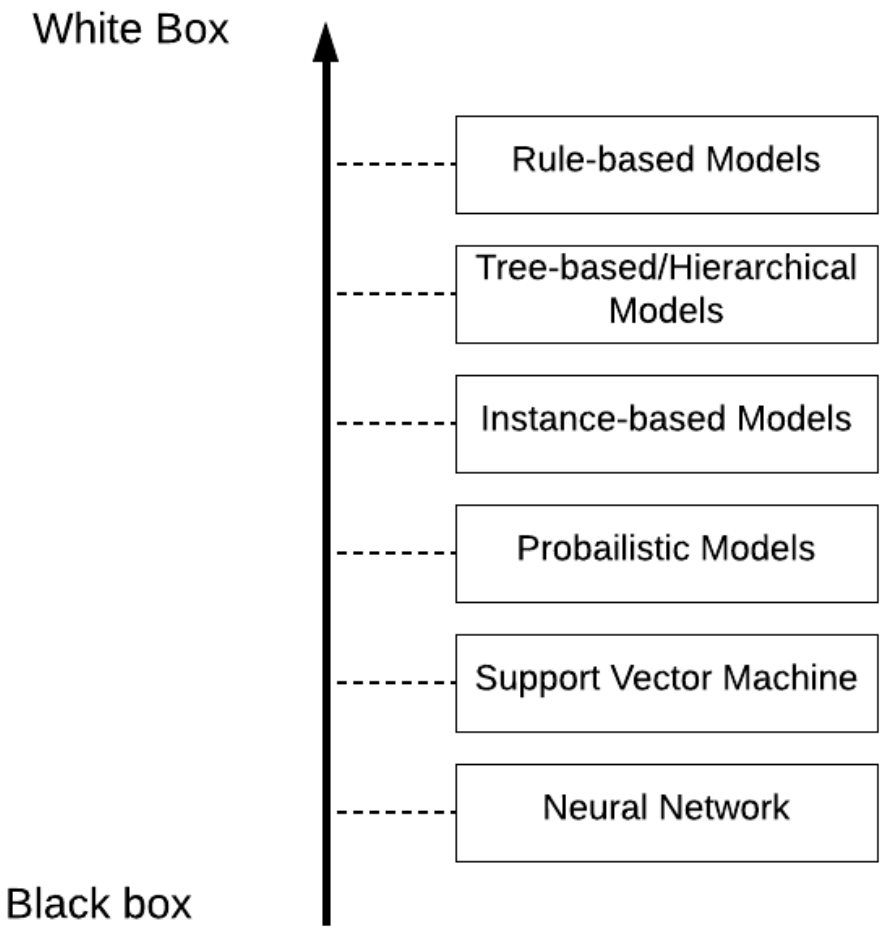


Figure 2.1: Hierarchy of output expressiveness.

There are many different ways to represent data concepts that can be learnt from data mining tasks. At the simplest end of the scale, modelling is performed on a single data table. A row in the data corresponds to a data example to be analysed in terms of its characteristic (feature) and the concept (class) to which it belongs. The ultimate objective of a trained model

is to discover one or more patterns that reflect the underlying truth of the data.

Classification is a category of supervised learning where the main task is learn from a set of training data examples that are classified (labelled by class labels) [83, 35]. Subsequently, a trained model will be able to classify label for newly encountered data examples. Examples of classification tasks can be classification of countries based on climate, classification of cars based on fuel consumption, or prediction of a diagnosis based on a patient's model record/condition.

Generally, a classification can be formulated as a set of observations described with a fixed number of features, F_i, and a designed class feature, C. The learning/modelling task is to find a mapping, $f(...)$ that is able to compute the class value, $C = f(F_1, F_2, ..., F_n)$ from the feature values of seen and previously seen data examples.

The form of a decision tree and a set of modular classification rule are two of many available structures to represent learnt models/concepts from data for classification tasks [104, 108, 23, 121]. The popularity of using tree-based and rule-based methods to represent learnt models is no exception for classification tasks in the data stream community [123, 3, 76, 115].

2.1.1 Tree-based / Hierarchical Model

A decision tree is a flowchart-like data structure implementing the "Divide-and-Conquer" strategy and is made up of internal decision nodes and terminal leaves [123, 102]. An internal node (non-leaf node) denotes a test on

a feature and the test will either produce a branch or a leaf node as shown in **Figure 2.2**. A branch represents an outcome of the test and a leaf node represents a class label. The process starts at the root and internal nodes are recursively produced until leaf nodes are produced for all the tests [123].

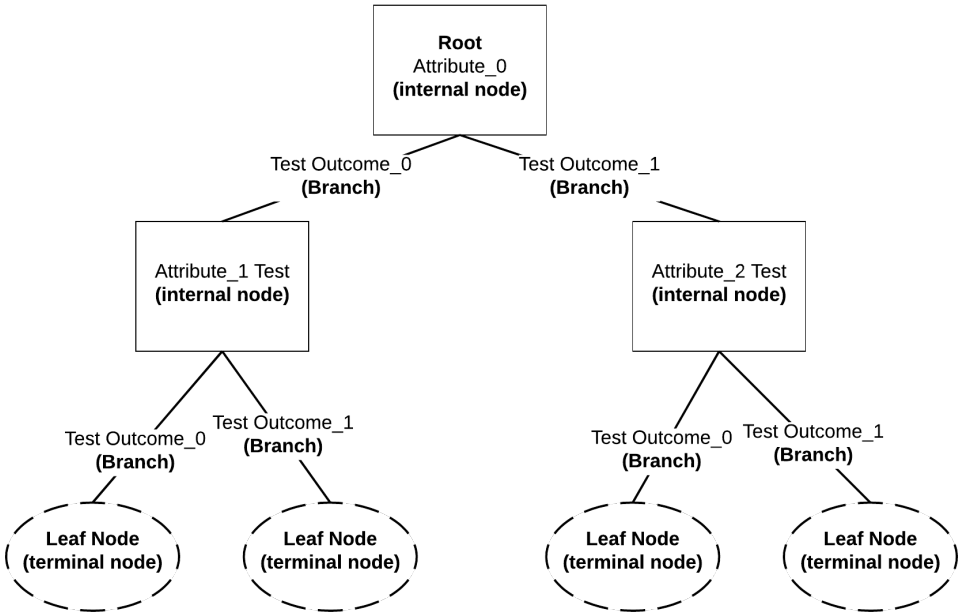


Figure 2.2: An example of a decision tree structure based on the description of constructing decision trees [123].

For classification, a decision tree is a well-known nonparametric method which has been used for classification in many applications such as financial analysis, astronomy, and medicine. Given a data example in which the associated class label is unknown, each feature of the data example is tested against the created decision tree. A path is traced from the root to the leaf node where the leaf node represents the class label prediction for that data example [65].

As previously mentioned, a decision tree is a nonparametric method and it does not require any domain knowledge or parameter settings, which makes it ideal for exploratory knowledge discovery. Multi-dimensional data can also be handled by decision trees [65].

In the late 1970's and early 1980's, a version of tree-based algorithm known as ID3 (Iterative Dichotomiser 3) [18] was developed by J.Ross Quilan. C4.5 algorithm [104] was later introduced as the success of ID3 algorithm and it is frequently used as a benchmark for comparison between learning algorithms [65]. A general pseudo-code for building decision trees is shown in **Algorithm 1**:

Algorithm 1: A general pseudo-code for building decision trees [83].

```
1 Check for base cases;
2 foreach each feature, α do
3 |   Find the selected criteria from splitting on α;
  end
5 Let α_best be the feature with the height normalised information gain;
6 Create a decision node that splits on α_best;
7 Recur on the sub-lists obtained by splitting on α_best, and add those
    nodes as children of node;
```

Generally, a decision tree is created in a top-down manner, starting with the most basic tree with only a root node, and then extending it to a more specific tree structure. For instance, if a tree consists of only the root node then it is too general, while the most specific tree would construct a leaf node

for every single data example, which would be too specific, or overfitting. One of the main criteria for a good decision is to have a tree at the right level which will adequately generalise the training data examples to enable a high predictive accuracy on unseen data examples.

Furthermore, the book CART (Classification and Regression Trees) [65] by L. Breiman, J. Friedman, R. Olshen, and C.Stone was published in 1984, describing the generation of binary decision trees. In fact, ID3 and CART (Classification AND Regression Tree) were introduced independently of one another at approximately same time but they both cover a similar approach for learning decision trees from data [65]. The implementations of these algorithms are available in the WEKA workbend [64, 71], as open source.

2.1.2 Modular Rule-Based Model

Another important representation of classification is a set of collective classification rules. A rule set is a very popular alternative to decision trees, where the rules are generated by the "Separate-and-Conquer" search strategy. The studies of rule-based model have been active as early as the 1960's [49, 48].

Typically, a rule-based classification has a set of 'IF...THEN...' rules. Each rule has a conjunction and feature-value on the left-hand-side (LHS) of the rule, and a class label in the rule consequent, or right-hand-side (RHS) of the rule. As an option, probabilistic rules can also be generated where the consequent of these rules consists of a list of probabilities or number of covered training data example for each possible class label [27]. Rule sets are generally simpler and more comprehensive, compared with decision trees.

The ideas for learning classification rules are quite similar to the ideas used in a decision tree induction. The "Divide-and-Conquer" search in decision tree recurrently partition the training data examples by advancing the purity measurement of the succeed node. On the other hand, rule learning algorithms only expand one single successor at a time, thereby learning a complete rule that covers part of the training data examples. After a complete rule has been learned, all data examples that are covered by the created rule are removed from the training set, and the process is repeated with the streaming data examples until all data examples in the training set are covered by a rule, and this strategy is known as "Separate-and-Conquer" search.

Pseudo-codes for "Divide-and-Conquer" and "Separate-and-Conquer" searches in general are described as in **Algorithm 2**:

Algorithm 2: General pseudo-codes for "Divide-and-Conquer" and "Separate-and-Conquer" searches.

Divide-and-Conquer:

2 **Divide** problem into a group of smaller problems of the same type;

3 **Conquer** the smaller problems by solving them recursively;

4 **Combine** the solutions to get a solution to the main problem;

Separate-and-Conquer:

6 **Learn** a rule that covers part of the given training set (the *separate* part);

7 **Recursively** learn another rule that covers some remaining examples (the *conquer* part);

8 **Repeat** this process until no examples remain;

If using a "Divide-and-Conquer" searching approach, then the rules are generated directly from a decision tree, such as the C4.5 classifier [123], while a "Separate-and-Conquer" search induces 'IF...THEN...' rules directly from training data examples, without constructing a decision tree [26]. A rule is deemed to be complete if it only covers data examples from a target classification. A process of generating rules directly from data examples can be insulated as in **Figure 2.3**.

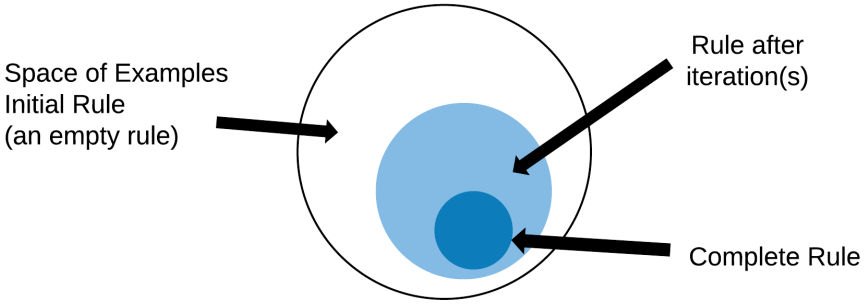


Figure 2.3: Inducing modular classification rules by using a "Separate-and-Conquer" search.

Also, a decision tree can easily be converted to a set of classification rules, in which a rule is generated for each leaf. The antecedent of the rules contains a condition for every node on the path, from the root to that leaf [123].

2.1.3 Limitations of Tree-base Representation

Cendrowska [26] notes that a decision tree may grow excessively large and complex, and she introduces the Prism algorithm, a "Separate-and-Conquer" search based algorithm that can introduce modular rules which do not necessarily fit into a decision tree. For example, the concept below can easily be represented by two modular rules and this concept does not fit into a decision tree structure without creating an irrelevant feature for duplicating certain information.

IF a = 0 **AND** b = 1 **THEN** class = X

IF c = 0 **AND** d = 1 **THEN** class = X

Otherwise, class = Y

However, the above three rules cannot be precisely represented by a decision tree without redundant information, because the rules do not have any features in common. Representing these rules in a decision tree structure would require adding unnecessary and meaningless rule terms, which is also known as the *replicated subtree problem* [123, 114]. **Figure 2.4** illustrates the simplest tree to represent the above rules and highlights the issue of redundant information when representing some kinds of concept with a tree-based structure.

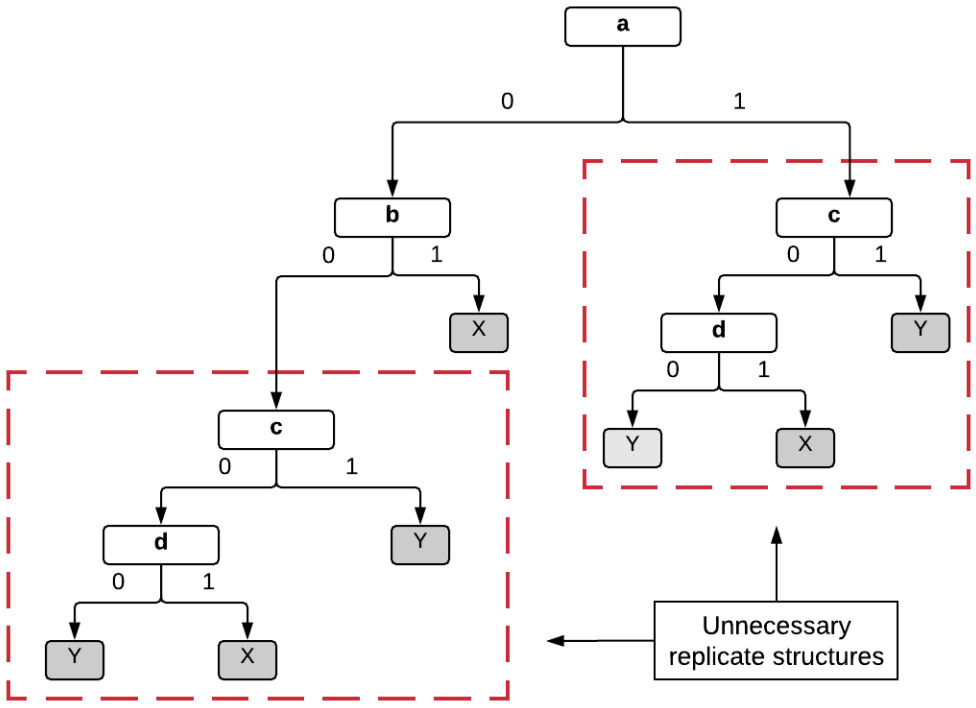


Figure 2.4: An example of a replicated subtree problem for the rule; IF $a = 0 \wedge b = 1$ THEN class = X, IF $c = 0 \wedge d = 1$ THEN $class = X$, otherwise $class = Y$.

The above tree structure was generated under the assumption that there

are only four features (a, b, c, d), with each feature either associated with the value of 0 or 1; and there are only two possible class labels in the stated decision rules. From this example, extracted rules from the "Divide-and-Conquer" (tree structure) can lead to unnecessarily large and complex trees as well as redundant rules [123]. Also, Fürnkranz et al. [50] suggest that decision tree-based models are less expressive, as they tend to be complex and difficult to interpret by human once the tree model grows to a certain size. Nevertheless, Quinlan [104], the author behind the well-known C4.5 machine learning decision tree-based algorithm, acknowledged that pruning a decision model does not guarantee simplicity and can still be too cumbersome to be understood by humans.

Additionally, rules are also popular in their own terms because each rule contains an independent piece of knowledge, thus adding or removing a rule from a rules-set will not affect other rules [65].

2.2 Static and Streaming Data

The works in [5, 14, 40, 41] define a data stream as being a 'sequence of data examples' which are produced at either high rate, high volume, or both. There are a few characteristics that can be used to distinguish between data streams and conventionally static data:

- The data examples arrive online and in sequential order. For this reason, the order in which the data examples arrives cannot be controlled, either within a data stream or across the data stream.

- Unbounded, the data examples always arrive at the system but the exact quantity is unpredictable and the seen data examples may not be able to fit in memory or on disks over time.
- Once a data example has been seen and processed, it should be discarded or archived away from the main memory.

Given these characteristics, conventional data mining algorithms are not suitable for data streams, because they were not designed for rapid and continuous flow of data examples, and continuous queries [117] are not supported, though supporting a continuous flow of data examples is an essential prerequisite in data stream applications [14].

Practical Examples

Real-life applications involving data streams can be found in many domains, including financial applications, networking, sensor monitoring, and web applications.

Sensor monitoring [25, 90] is a very good example of the applications of mining a data stream. More and more sensors are used in either our daily lives or the industrial sector. Sensors behave very differently from traditional data sources because the data is periodically collected and pushed away immediately, keeping no record of historical information. Their limitations make conventional mining techniques and approaches inappropriate for queries over sensors.

Tradebot systems [88, 2] offer solutions to analysing queries about real-time streaming financial data, such as stock tickets and news feeds. At any

time, Tradebot regularly accounts for 5% of the total trading volume on the US stock market, and Tradebot systems typically hold stocks for just 11 seconds for each automatic transaction [2].

Big Data Stream Mobile Computing (BDSMC) [15] is a paradigm that relies on advances in mobile communication and real-time mobile cloud computing. Nowadays, a spatially distributed network of thousands of low energy devices (e.g., smartphones, PDAs, tablets, RFIDs and smart home appliances), are all capable of acquiring and communicating in real time, which creates a huge volume of heterogeneous data streams. The paradigm outlines a new generation of inter-connected mobile/wireless computing infrastructures that can be used to extract hidden insights from an ever increasing volume of time-series correlated heterogeneous data streams.

Social networking websites (Facebook, Twitter, LinkedIn) tailor their content to individual users. In order to do so, these services need to look at and analyse data generated from their users in real time to keep the content up-to-date. For example, Facebook [97] has over 1.35 billion active users monthly and some of its generated data includes 4.5 billion likes, 300 million photo uploads, and 4.75 billion pieces of content shared daily.

Internet of Things (IoT) Stream Mining has been recognised as one the most exciting and key opportunities for both academia and industry [37, 96, 44]. The IoT is driving a new industrial revolution, creating a new source of wealth insights and unprecedented levels of innovation. Smart manufacturing, smart cities and the automotive industry are among the notable examples of the uses of IoT devices, which result in high volume and high velocity of data from machine-to-machine (M2M) communication. Sub-

sequently, there is the need to learn and mine actionable insights from such data streams .

2.3 Challenges for Classification Tasks in Streaming Environment

When mining data streams, some very unique challenges are faced. [14, 5, 100, 115], mentioned, and discussed. Alternative approaches have been proposed for resolving these challenges. Also, in limited resource settings such as sensor networks [60], the characteristics of data streams are even more concerning and require careful planning so that a working solution can be delivered.

2.3.1 Unbounded Memory Requirement

For data stream settings, the amount of required physical hardware and resources is often uncertain. There are algorithms [120] that use external memory for handling data which is unable to fit into main memory. However, such algorithms are not developed to work with data streams specifically, because they cannot handle continuous queries and they are typically not able to give instant or near real-time responses. A continuous data model is most applicable to situations where quick responses are required and the data is continuously generated at a high rate over time. Systems designed for data streams are required to accept new arriving data examples while processing old data. Therefore, the time spent on each data example needs

to be low, otherwise the latency of computation will be too high and the system will not be able to keep pace with the data streams [120]. For this reason, data stream mining systems or algorithms need to confine themselves to main memory without accessing secondary storage.

Arasu [11] published some very early work on characterising data models which can be precisely learnt using a given bounded amount of memory and models that must be approximated if disk access is not allowed. A limited number of data models were considered and a complete list of features for models that require a potentially unbounded limit of memory (proportional to the size of the input data streams) to answer was offered. The work by the author [11] shows that without knowing the size of the input data stream, it is impossible to place a limit on memory requirements for most data models, unless the domain of features involved is restricted (either based on known data properties or through the imposition of models). The main idea behind the work is that without domain restrictions, an unbounded number of feature values must be remembered because they might join with data examples that would arrive in the future.

2.3.2 Concept Drift

Concept drift is a unique property of data stream and does not happen in static (batch) data. A concept represents underlying information and knowledge from the data. However, in streaming data, a concept drift [38, 115, 25, 57, 100] happens when data changes from one concept to another. The changes can be sudden or gradual and a past concept may re-occur in

an unforeseen manner.

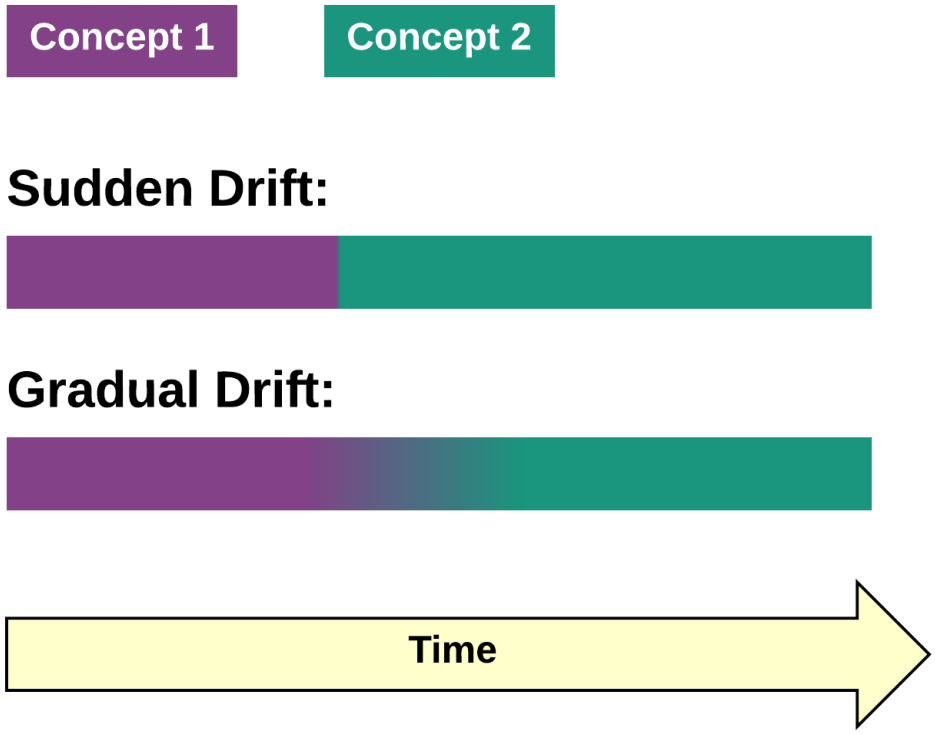


Figure 2.5: Different types of concept drift in data streams.

Generally, in data streams, the concept of the data is expected to change in unforeseen and unpredictable ways. There are various approaches to attempt to overcome this challenge in streaming environment from the researching community. As early as in 1986 [111], Schlimmer and Grainger raised the topic of learning complex and changing environments. They proposed a method that is able to tolerate noise and drift by re-calculating the weight of training data examples based on how well they fit future data examples. Bach and Maloof [16] suggested placing a probability distribution over the lo-

cation of the most recent detect drift point and then using a Bayesian model to compare and update the distribution from the predictions of the model.

Generally, there are two common approaches to tackle concept drift. The first approach is to directly maintain and modify the learnt models as in [72, 122, 115]. The second approach is to maintain and compare the created models for some seen data examples in order to select those that are the best performers for unseen data examples [17, 112, 124, 116].

2.4 Established Classification Algorithms for Stream Classification

Classification perhaps is one of the most widely studied and researched aspects of data mining, and it is the same for data stream mining. However, there are very few algorithms and systems that have been specifically proposed for data stream mining tasks. Some algorithms were proposed as an improvement to existing algorithms for batch settings to work on data stream settings [31, 113, 106]. On the other hand, some were proposed to address the fundamentals in data streams, such as high velocity data and concept drifts [72, 57, 40, 115, 19, 92].

2.4.1 Hoeffding Trees

Domingos and Hulten introduced the Hoeffding Trees algorithm in their paper "Mining High-Speed Data Streams" [40, 72]. The system in their paper was referred to as VFML (Very Fast Machine Learning). At any time, the

system can build decision trees with the use of constant memory and constant time per data examples. The Hoeffding Trees or VFDT (Very Fast Decision Tree) algorithm was the underlying algorithm for their system. VFML overall introduced several enhancements for practical evaluation.

The Hoeffding Trees algorithm can be regarded as the current "state-of-the-art" algorithm in data stream mining because it accomplishes the characteristics of an algorithm which are needed for mining tasks for data streams, and are able to cope with a high velocity flow of data while remaining computing efficiency as well as adequately accurate [82]. Additionally, the Hoeffding Trees algorithm uses the Hoeffding's Bound [69] to guarantee that the learned model is asymptotically almost identical to that of a conventional learner [40].

The Hoeffding Bound [82] states that with probability $1-\delta$, the true mean of a random variable of range, R will not differ from the estimated mean after n independent observations of more than ϵ, as shown in the **Equation 2.1**:

$$\epsilon = \sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}} \tag{2.1}$$

The use of the Hoeffding Bound is very helpful because it gives an estimation of the number of samples needed to learn data examples from a data stream, as the classifier learns from an infinite number of data examples regardless of the distribution generating the values. The Hoeffding Bound depends only on the range of values, number of samples, and desired confidence.

Algorithm 3 describes the process of inducing a decision tree from data

streams using the Hoeffding Trees algorithm.

Algorithm 3: Hoeffding Trees algorithm [21]

```
1  Let HT be a tree with a single leaf (the root);
2  forall all training examples do
3      Sort example into leaf l using HT;
4      Update sufficient statistics in l;
5      Increment n_l, the number of data examples seen at l;
6      if n_l mod m_min = 0 and data examples seen at l not all of same
         class then
7          Compute G̅_l(X_i);
8          Let X_a be feature with highest G̅_l;
9          Let X_b be feature with second highest G̅_l;
10         Compute Hoeffding Bound ε = sqrt( R^2 ln(1/δ) / 2n );
11         if X_a ≠ X_θ AND (G̅_a − G̅_b > ε OR ε < τ) then
12             Replace l with an internal node that split on X_a;
           end
14         forall Branches on the split do
15             Add new leaf with initialised sufficient statistics;
           end
       end
   end
```

Domingos and Hulten showed in their empirical studies that a decision tree learned by the Hoeffding Trees will be similar to a decision tree learned

via batch learning, which produces a tree of the same quality as a batch data learned tree, despite being induced incrementally. This finding is important because for a long time batch data-learned decision trees had been considered among the best performing machine learning models [125]. The classic decision tree algorithms C4.5 [104] and CART [125] both induce decision tree models from batch data but were developed independently, are widely recognized by the data mining community, and are regarded by many as the *de facto* standard for batch learning.

2.4.2 VFDR: Very Fast Decision Rules

Gama and Kosina proposed an algorithm termed VFDR (Very Fast Decision Rules) that is able to continuously learn compact ordered and unordered rules-sets. Gama and Kosina highlighted the advantages of modular rules not being hierarchically structured like decision trees, in which new rules can be learnt for new concepts or removed when they became out-of-date, without affecting learning efficiency.

The Hoeffding Bound is also used in VFDR to decide when to expand a rule. In order to create rule-set from the data examples, a Hoeffding Bound determines the number of seen data examples after which a rule should be expanded and a new rule induced.

The rule-set (RS) is learned as shown in **Algorithm 4**.

Algorithm 4: VFDR (Very Fast Decision Rules) learning algorithm [57]

input : S: Data stream source;

N_{min}: Minimum number of data examples;

ordered_set: Boolean flag;

output: RS: A set of classification rules

```
1  Let RS ← {};
2  Let default rule L ← θ;
3  foreach example (x, yk) ∈ S do
4      foreach Rule r ∈ RS do
5          if r cover the example then
6              Update sufficient statistics of Rule r;
7              if Number of examples if in L_r > N_min then
8                  r ← ExpandRule(r);
               end
10             if ordered_set then
11                 BREAK;
               end
           end
       end
15     if None of the rules RS trigger then
16         Update sufficient statistic of the empty rule;
17         if Number of examples in L > N_min then
18             RS ← RS ∪ ExpandRule(default rules);
           end
       end
   end
```

Algorithm 5: ExpandRule(): Expanding on Rule [57]

input : r: A decision rule;

H: Split evaluation function;

δ: 1 minus the desired probability of choosing correct feature;

output: $r^{'}$: Expanded rule;

Let h_0 the entropy of the class distribution at L_r;

Compute $\epsilon = \sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}}$ (Hoeffding Bound);

if $h_0 > \epsilon$ **then**

  foreach *feature* X_i **do**

    Let h_{ij} be the $H()$ of the best split based on feature X_i and value v_j;

    if $h_{ij} < h_{best}$ *and* $n_{ij} > 0.1 * n$ **then**

      Let $h_{best} = h_{ij}$;

    end

  end

  if $h_0 - h_{best} > \epsilon$ **then**

    Extend r with a new condition based on the best feature $X_a = v_j$;

    Release sufficient statistics of L_r;

    $r \leftarrow r \cup X_a = v_j$

  end

end

return r

The update process of an existing rule is shown in **Algorithm 5**, which employs a Hoeffding Bound metric to decide whether to split and expand the rule with new terms.

VFDR is able to deal with multiple classes, discrete and numeric features. Additionally, VFDR is shown to be very competitive compared to VFDT (Very Fast Decision Tree), the state-of-the-art in streaming decision tree) learning, while making use of modular rules which can be easier to interpret by humans to improve adaptability.

2.4.3 eRules - A Modular Adaptive Classification Rule Learning

eRules by Stahl, Gaber and Salvador [115] can be regarded as the most recent rule-based tool added to the data stream mining toolbox. Focusing on one of the very unique properties of data streams, which is concept drift, eRules classifier will construct a new model if the current model has a high percentage of unclassified data examples. This feature makes eRules very different from VFDT and VFDR, because eRules may abstain from labelling a data example if that data example is not covered by the rule set, while other classifiers such as VFDT and VFDR always enforce a classification [115]. This feature may be highly desirable in critical applications where misclassification is costly and irreversible, for example, medical diagnosis or financial applications.

Essentially eRules makes use of Cendrowska's Prism algorithm [26] to induce modular classification rules with a "Separate-and-Conquer" search

strategy, as mentioned in **Section 2.1.2**, from data streams by using a sliding window technique [14]. The three main processes of eRules as illustrated in **Figure 2.6** are outlined as follows:

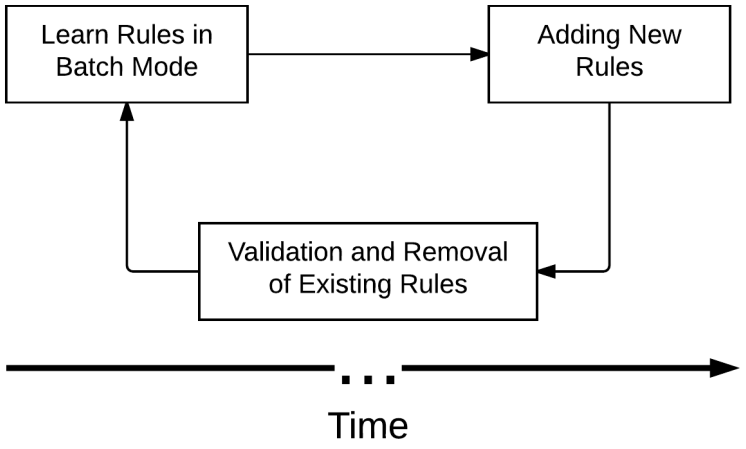


Figure 2.6: The three main processes in eRules algorithm [115].

Learning Rules in Batch Mode – An initial rule-set is learnt using the Prism algorithm on the first batch of incoming data examples. Later data examples are added to a buffer if they are not covered by the current rule-set. If the number of data examples in the buffer reaches a user defined threshold, then its data examples are used to induce new rules (using Prism) and the buffer is emptied.

Adding New Rules – Whenever new rules are generated from the aforementioned buffer, they are added to the current rule-set and thus adapt to new emerging concepts in the data stream.

Validation and Removal of Existing Rules – eRules also removes learnt

rules from the model if the concerning rules are not relevant to the current concept anymore, i.e., if concept drift occurs. This is quantified by the deterioration of the individual rule's classification accuracy over time.

2.4.4 Online Information Network

OLIN, an online classification system was proposed by Last [86]. The system can dynamically adjust the size of training window and the number of unseen data examples to rebuild the classification model with the most recent data examples.

The induced concept of the system uses the **I**nfo-**F**uzzy **N**etwork (INF) to build a tree-like classification model. The window size is controlled by information theory. The main idea is to repeatedly construct a new network from a sliding window of latest examples, according to the classification error rate. Therefore, if the model is stable, then the window will increase accordingly, up to a pre-specified limit. The latest model always classifies the examples that arrive before the subsequent network reconstruction. Typically, a concept is realised by a sudden spike in the classification error rate. If a concept is detected, the system re-calculate the size of the training window.

While the system produces a tree-like classification model, the tree is different compared to conventional decision trees in that each level of the tree represent only one feature, except the root note layer. The nodes represent the different values of the feature. An overview of the OLIN system is presented in **Figure 2.7**. The process of producing a classification is similar to that of conventional decision trees.

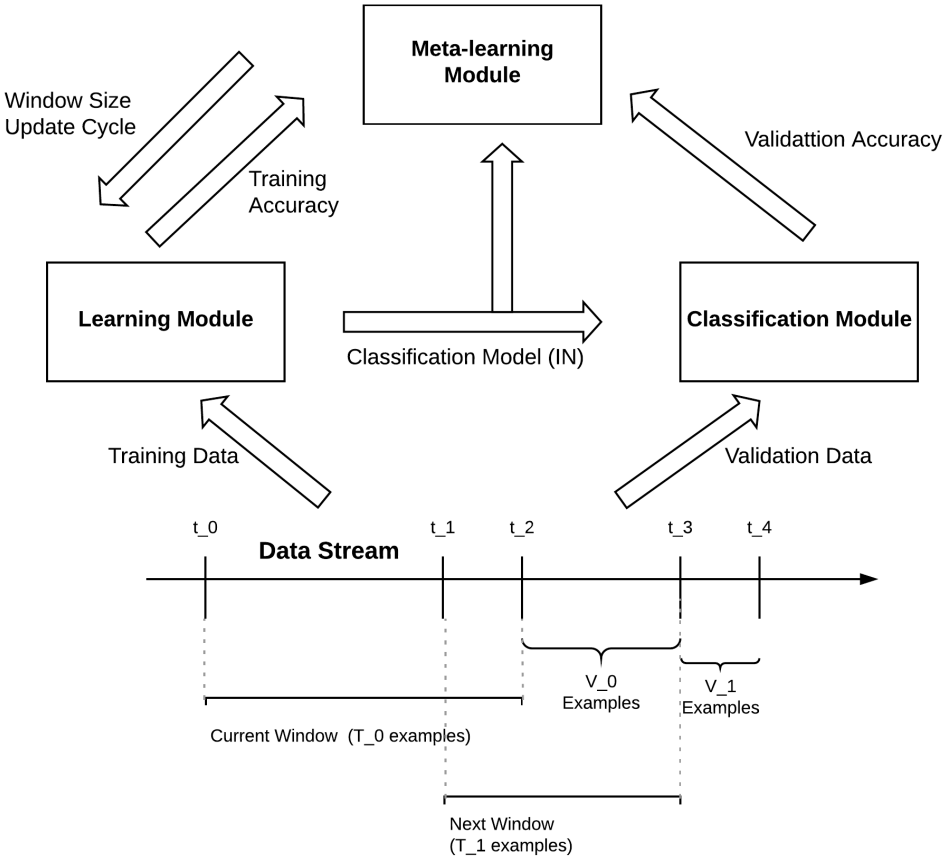


Figure 2.7: Overview of OLIN system architecture [86].

Nonetheless, the process of constructing the tree has been termed the **I**nformation **N**etwork (IN). As illustrated in **Figure 2.7**, each one of V_0 data examples in the validating internal $[t_2, t_3]$ is labelled by the created model from T_0 from the earlier training internal $[t_0, t_2]$. Furthermore, both training and validating intervals do not have to be the same. When it comes with t_3 , the system re-construct the tree-model by the 'Learning Module'

using T_1 data examples from the training internal $[t_1, t_3]$,and subsequently applied to V_1 data examples in the validating interval $[t_3, t_4]$. There is an assumption that the first data example in the interval $[t_3, t_4]$ that it arrives after the model construction has been completed. In streaming environment, unseen data examples can be labelled by partially constructed model in the IN algorithm [29]. Unlike using the Hoeffding Bound in VFDT [40, 41] and VFDR [57], OLIN uses a less conservative condition. The measurement is derived from the mutual conditional information in the IN algorithm by applying the likelihood ratio test in order to assess the statistical significance of the mutual information.

2.4.5 On Demand Classification

Clustering is typically regarded as an unsupervised approach. In [7, 6], Aggarwal et al. have introduced the concept of micro-clusters, which can adequately maintain statistics at both temporal and spatial granularity level. Subsequently, on-demand classification is introduced, in [4] which the idea of micro-clusters is utilised. At a high level, there two main processes that make up on-demand classification. Firstly, the process continuously learns and maintains summaries from the seen data examples. On the other hand, the second process continuously uses the stored summary statistics for a prediction of the class label. Each micro-cluster represents a class label, and the data points within the cluster will have the associated class label as the classification.

Both of the main processes can be executed in an online fashion, and this

can be particularly essential in a streaming environment in which new data examples arrive in real-time. At any point, the realised micro-clusters can be used to label unseen data examples.

2.4.6 LWClass Algorithm

In a proposal for online learning of data streams in a resource constrained environment, Gaber et al. [53] describe a strategy to dynamically adapt the learning process for data streams on the basis of available memory resources. The LWClass algorithm is rooted from Algorithm Output Granularity (AOG) [54, 51] which introduces the first resource-aware data mining approach that can cope with fluctuating data velocity according to the available computing resources at hand. The AOG performs the local data analysis on resource constrained devices that generate or receive streams of information. There are three main stages in AOG, which are learning, adaptation, and knowledge integration, as show in **Figure 2.8**.

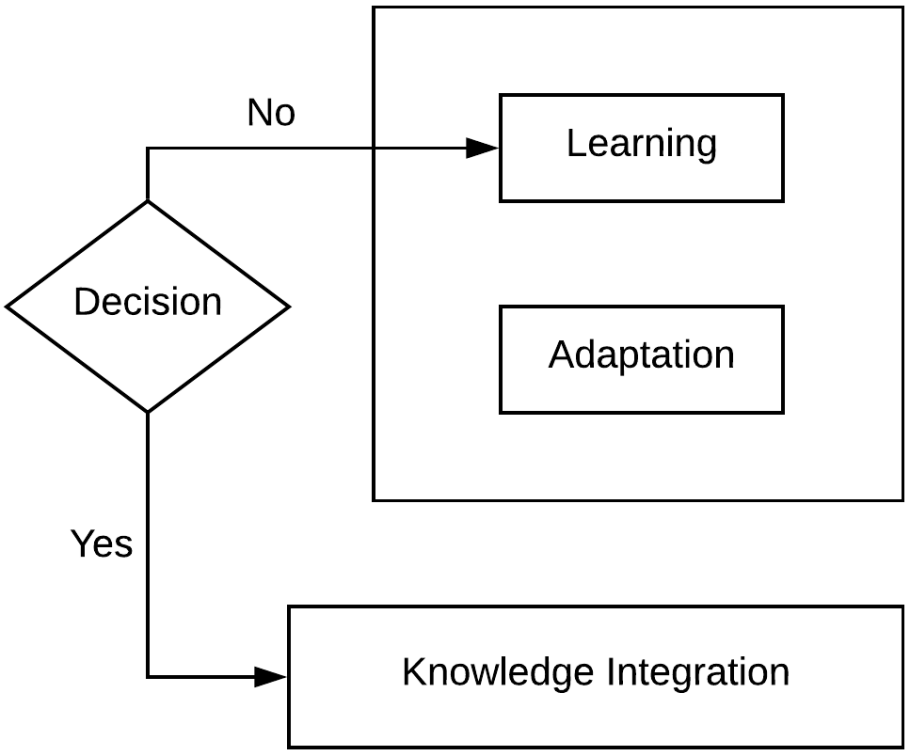


Figure 2.8: Three main processes in Algorithm Output Granularity (AOG) [53].

LWClass algorithm starts by calculating out the number of data examples which could be loaded in the memory, according to the available space. Once there is a labelled data example available, the algorithm will search for the nearest seen data example already stored in the main memory. There is a pre-defined distance threshold for the aforementioned task. The threshold represents the similarity metric acceptable by the algorithm to consider candidates for entry to the matrix. The matrix is also a summarised version of the seen data examples. Once the weight is zero, the entry is released from the memory

2.4.7 SCALLOP Algorithm

In [46], Ferrer-Troyano et al. introduce a scale classification algorithm named SCALLOP (**S**calable **C**lassification **A**lgorithm by **L**earning decisi**O**n **P**atterns) that introduce a model online based on a certain set of user defined parameters.

Generally, it is inherently difficult to construct stable high quality rule-based classifiers from data streams, because of the challenges in keeping the underlying rule-statistics throughout the time horizon. This is one of few rule-based classifiers related to classification tasks for data streams that focuses on a scalable algorithm to classify numerical, low dimensionality, high-cardinality and time-changing data streams.

The algorithm is started by taking a number of user-specified labelled data examples where α is rules per class label. Afterwards, the algorithm focuses on maintaining the rule-set after the arrival of each new data example. At the point of arrival, each data example falls into one of these three possibilities:

- **Positive covering**: The data example adds more strength to a current discovered rule.
- **Possible expansion**: The data example is covered by any generated rules but there is a rule that can be extended to cover the data example without overlapping with different labelled rules.
- **Negative covering**: The data example weakens to a current discovered rule.

For each of the above cases, a different procedure is defined as follows:

- Positive Covering: The values of positive support and confidence of the existing rule are re-calculated.
- Possible Expansion: In this case, the generated rule is extended if it satisfies the following two conditions:
    1. The data example is bounded within a user-specified growth bounds to avoid a possible incorrect expansion of the rule.
    2. There is no intersection between the newly revised rule and any generated rules for the same class label.
- Negative Covering: In this case, the negative support and confidence level are re-calculated. If the new confident value is smaller than the user-specified threshold, then a new rule is added.

The set of rules is refined for every γ new data example, where γ is a user-specified parameter. If the rules have the same class label and are within a user-defined acceptable distance, the rules are merged. Meanwhile, it is necessary to make sure that the rules do not intersect with rules associated with other class labels. The resulting hypercube of the merged rules should also, be within certain growth bounds. **Figure 2.9** illustrates the key concept of SCALLOP.

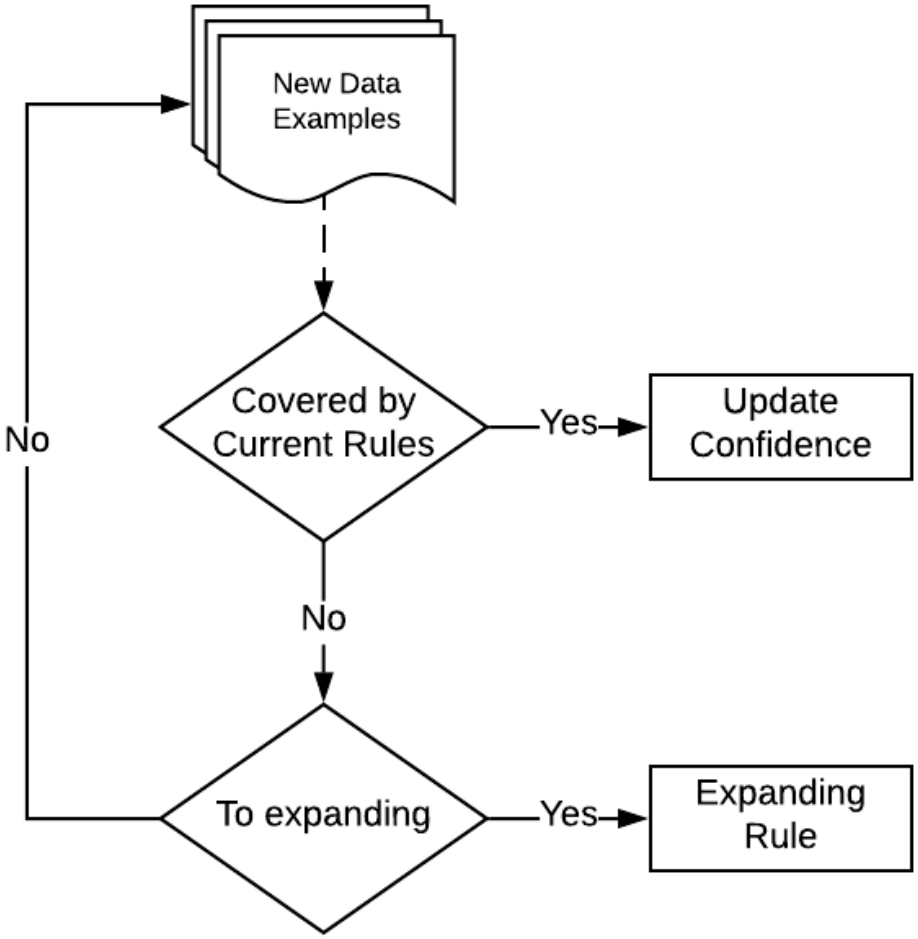


Figure 2.9: Main processes in SCALLOP algorithm [46].

Finally, there is also a refinement stage which releases the undesirable rules from the current models. Particularly, uninteresting rules have less than minimum positive support. Additionally, the rules that are not covered by at least one of the data examples of the last user-defined number of received data examples are released.

For prediction, a voting system is utilised to classifying the unlabelled

data examples. If there is a rule covering the current record, the class label associated with that rule is used as the classifier output. Otherwise, voting over the current rules within the growth bound is used to infer the class label.

2.4.8 ANNCAD Algorithm

Law et al [87] introduced an incremental classification algorithm, termed ANNCAD (an **A**daptive **N**earest **N**eighbour **C**lassification **A**lgorithm for **D**ata Streams). This uses multi-resolution data representation to locate adaptive nearest neighbours of a test point.

At the beginning, the algorithm begins with an attempt to locate the data example according to the majority of nearest neighbours at finer levels. In case the finer levels do not differentiate between the classes with a pre-defined threshold value, the coarser levels are used in a hierarchical way. In case of concept drifts, an exponential fade factor is used to decrease the weight of old data in the classification process. Ensemble approach is used to overcome the errors of the initial quantization of data. **Figure 2.10** shows the main processes that make up the ANNCAD algorithm.

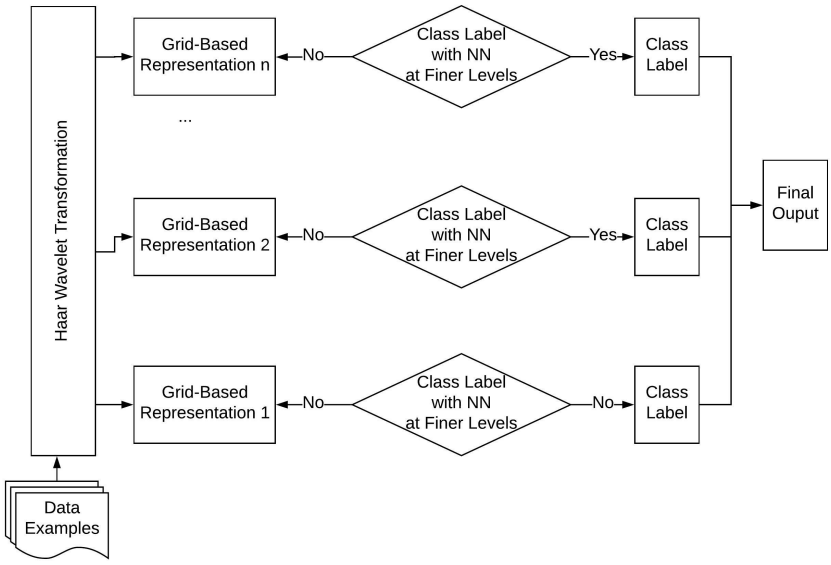


Figure 2.10: Main processes in ANNCAD algorithm [87].

As claimed in [87], the experimental results of the ANNCAD algorithm were superior compared with VFDT [72]. However, this algorithm does not seem to have the ability to deal with sudden concept drifts as the exponential fade factor takes a while to cease having an effect. As a matter of fact, the choice of the exponential fade factor could have undesired effects on either an over or under-estimate of the rate of concept drifts. Both cases would result a in reduction in accuracy.

2.5 Evaluation Procedures of Learning Algorithms for Stream Classification

The evaluation process is an important task to identify various critical measurements of an algorithm. Normally, accuracy, or equivalently its converse

error, is the most interesting concern, and it also is the most applicable [21].

In classification, accuracy is typically measured as the percentage of correctly classified unseen data examples after being trained on a given number of seen data examples. In other words, the set validating the data example is never exposed to an algorithm during the learning phase, rather this occurs at the evaluating phase to avoid the bias of the training model. A small number of mistakes when predicting labels of data examples indicates the reliability of the learning algorithm, and achieving the highest possible accuracy is the most immediate and obvious goal of most data learning algorithms. If there is a found nearest neighbour, it checks the class label. If the class label is matched, then the weight of the instance is increased by one; otherwise, it is decreased by one.

Because of its unique characteristics, a data stream has different requirements compared with batch data settings. For carrying out a proper evaluation, batch data learning algorithms can usually have access to the data without being limited the amount of time that data examples can be seen or analysed. In data stream settings, the focus shifts to trend over time, where the data concept develops over time and can be realised at different stages of growth [21].

Furthermore, data stream classification is still a relatively new field in the data mining community. For this reason, evaluation practices are not nearly as well researched and established as they are in the batch setting. The amount of literature related to data streams is increasing rapidly, but there is only a limited amount that specifically addresses or relates to data stream classification as defined in this research. The papers found that most

closely related to the content of this research are:

- Mining high-speed data streams by Domingos and Hulten [40].
- Accurate decision trees for mining high-speed data streams by Gama, Rocha and Medas [59].
- Forest trees for on-line data by Gama, Medas, and Rocha [58].
- Efficient decision tree construction on streaming data by Jin and Agrawal [76].
- A streaming ensemble algorithm (SEA) for large-scale classification by Street and Kim [116].
- Multi-interval discretization of numeric-valued features for classification learning by Usama and Keki [74].
- Online ensemble learning: An empirical study by Alan and Givan [45].
- eRules: A modular adaptive classification rule learning algorithm for data streams by Stahl, Gaber and Salvador [115]
- Learning decision rules from data streams by Gama and Kosina [57].
- New Options for Hoeffding Trees by Pfahringer, Holmes, and Kirkby [100].

The evaluation methods used in the above literature show that single holdout was the most used method to obtain estimated accuracy, although some papers used five-fold cross-validation, while repeated sampling methods are used in [45] and test-then-train is used in [57, 115].

2.5.1 Holdout

At a certain quantity, cross-validation on batch data can be too time consuming, and the measurement from a single holdout set is often accepted. If the division between the train and test sets have been predefined, the results from different studies can be directly compared. Applying data stream settings as a large scale of batch learning, it is appropriate to use the holdout practice on data stream learning [21].

Unlike batch settings, the learning model for a data stream should be monitored periodically to detect changes and concept drifts from the data stream. However, the learning model should not be tested too frequently because testing the model may often significantly slow down the evaluation process, depending on the size of the test set. In data stream settings, a set of new examples from the stream that have not yet been used to train the learning algorithm can be considered a source of holdout examples.

A possible source of holdout data examples is a new data example from the data stream that has not yet been used to train the learning algorithm. A procedure can 'look ahead' to collect a batch of data examples from the stream for use as test data examples. This method would benefit the learning efficiency of scenarios with concept drifts, as it could be used to measure the ability of the learning algorithm to adapt to the latest trends in the data. However, in case of no concept drift, it is assumed that a single static holdout set should be enough, which avoids the problem of varying estimates between potential test sets [21].

2.5.2 Interleaved Test-then-Train or Prequential Testing

A popular alternative to the holdout method that has been used is to interleave testing with training. A data example is used to test the existing model before it is used for training, and the overall accuracy is incrementally updated [41, 21]. By conducting the testing and training in this order, the model is always being tested on data examples it has not seen. This method has the advantage that no holdout set is required for testing, which makes the maximum use of data examples. Also, the plot of accuracy is likely to be smoother over time, as shown in **Figure 2.11**, as a single data example is less significant to the overall average.

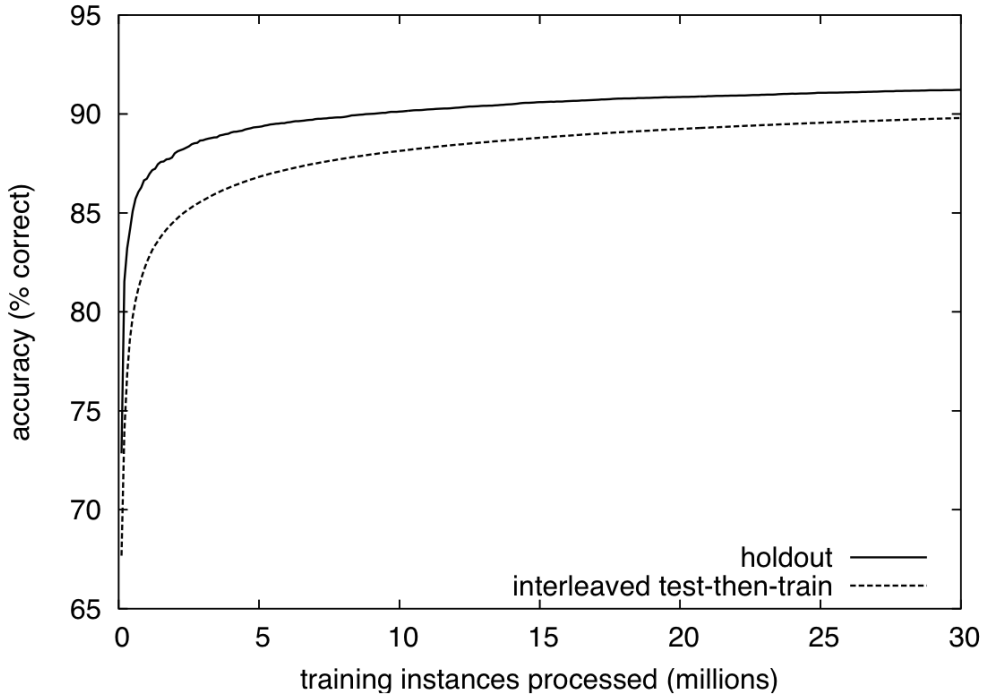


Figure 2.11: Accuracy plotting produced from learnings from the same data stream by holdout and Prequential methods, at a rate of every 100,000 data examples [21]

However, there is also a realised disadvantage to this method, in that it is difficult to separate and measure between training and testing times. Also, the accuracy at any given time is unlikely reflect to the actual accuracy of the learning algorithm at that exact point. Therefore, data stream learning algorithms using this method of evaluation will be punished for early mistakes, regardless of the level of accuracy they are capable of at a later time, although this effect tends to be balanced out over time as shown in **Figure 2.11**.

2.6 Limitations of Existing Predictive Algorithms for Streaming Data Mining

A common characteristic of the majority of the systems and techniques described is that they are either not expressive (with eRules and VFDR being the exceptions) or robust in learning data examples from streaming data sources. For instance, ANNCAD algorithm is not able to deal with sudden concept drifts because the exceptional fade factor takes a while to cease having an effect. Another example, SCALLOP algorithm relies on a set of defined statistical metrics to maintain the learned rules and this mechanism shows the inflexibility in adapting to concept drifts.

The key challenges in data stream classification are represented in high-velocity rate, concept drifts, and the unbounded memory requirements. For instance, when dealing with high-velocity data input, an algorithm is constrained to deal with recently arrived data examples within a fixed period, which can influence the reliability of the learned model. Concept drifts may affect the overall accuracy or error rate due to the invalidation of a confirmed concept from the past if there is no mechanism to detect these changes over time. Unbounded memory refers to the space dimension where the concept should be learnt from a given number of seen data examples rather than from a complete training set like in static data.

While many approaches have been proposed to address some of these challenges, they are often unable to address these challenges simultaneously. **Table 2.12** outlines the previously reviewed techniques in terms of addressing the aforementioned challenges.

Technique	High Velocity	Memory Requirements	Concept Drift
Hoeffding Tree	•	•	
Very Fast Decision Rules (VFDR)		•	•
eRules		•	•
Online Information Network			•
On Demand Classification	•		•
LWClass	•	•	
SCALLOP			•
ANNCAD			•

Figure 2.12: Summary of reviewed algorithms for classifications tasks in streaming environment.

While Hoeffding Trees can be regarded the as state of the art in terms of accuracy compared with others, but it produces a tree-based model which is complex and hard to understand when it reaches a certain size. Furthermore, the ability to abstain/refuse from classifying when there is a high degree of uncertainty is not present in the majority of the algorithms/techniques above.

Furthermore, some of the algorithms are developed to induce classification rules directly from training data examples such as eRules and Very Fast Decision Rules (VFDR). However, they both use an inefficient method in learning heuristic from numeric features, binary splitting. This method is appropriate for a static environment where all possible values of the numeric feature are known, but this is not the case for streaming environments.

Finally, the works and studies related to classification for the streaming environment are far from being complete. A number of challenges still exist

in stream classification algorithms; particular with respect to concept drift and resource adaptive classification.

2.7 Summary

This chapter provides a background of the differences between two primary forms of data representation for a predictive model in data mining and how a tree-based model can be problematic to represent a classification model in the streaming environment due to the replicated sub-tree issue as raised in **Section 2.3**. Particularly, the tree-based structure is hard to remove an invalid concept which was valid at the early stage of the learning, and this is the case of concept drifts data streams.

The key challenges for classification tasks in a streaming environment were described, such as concept drifts and high-velocity data input continuously in **Section 2.3**.

Section 2.4 reflected on the existing algorithms for classification tasks for data streams. **Section 2.6** identified limitations of these algorithms regarding to the three evaluating criteria (accuracy, speed and space) which are primarily used to assess the performance of an algorithm for classification tasks in streaming environment.

The need for dedicated evaluation procedures for classification algorithms in a streaming environment was highlighted in **Section 2.5**. A traditional method such as Holdout is not suitable to capture necessary metrics (accuracy, speed and space) to measure the performance of a predictive streaming algorithm. Accuracy is typically the primary measurement for evaluation for

classification algorithm with a static dataset, but this is not any more the case in a streaming environment.

While streaming data is still a relatively new topic in the data mining community, driving factors such as the explosion of the Internet of Things and the advances in wireless connectivity mean that data streams will certainly play a larger and more essential role in knowledge discovery. The major research challenges for classification tasks in streaming environments are represented in concept drift, resource adaptivity, high velocity, and unbounded memory requirements.

In subsequent chapters, various aspects related to classification tasks with streaming data are presented in a greater detail, as well as the proposal of statistical approaches and algorithms to directly learn classification rules from streaming data sources along with the corresponding empirical evaluations.

Chapter 3

A new Approach to Improve the Computing Efficiency in Dealing with Numeric Feature for Data Streams

This chapter develops a technique which can effectively and efficiently learn heuristic for numeric features in a streaming environment.

High-velocity and potentially unbounded data input in a streaming environment poses some challenges to learn the necessary heuristic for numeric features efficiently and accurately.

Conventionally, many well-established algorithms can work with both categorical and numeric features out of the box because the training data examples are static and stable at the time of learning. Numeric features often need to be transformed/discretised into categorical features with a separate

process, or as an integrated part of an algorithm, before the actual learning tasks can start. However, these methods often require stable learning values and full exposure to the numeric values of the feature. However, this is not the case for a streaming data source because the data input can arrive with high-velocity and possibly unbounded.

The efficiency of an algorithm the primary concern in one of the raised research questions in **Section 1.2**:

*"**Research Question 2**: What are the most efficient approaches to improve the efficiency in learning heuristic for numeric features from a high-volume and high-velocity streaming data source?"*

In this chapter, various approaches in dealing with numeric features are discussed as well as a novel approach to efficiently deal with numeric features from the core. Subsequently, **Section 3.5** shows the integration of the proposed approach into existing eRules algorithm, and its empirical evaluation.

3.1 Working with Numeric Features

Traditionally, algorithms designed for batch settings can learn from numeric features because they have access to all available values of the numeric features at learning time. Approaches for handling numeric features have been studied extensively. Some algorithms such as 'support vector machine' [34, 33] and KNN [10, 32] can naturally work with numeric features as input due to the way that they are designed to operate based on Euclidean space. Other algorithms expect categorical features as inputs, but need to have an

extra component or step to process numeric features; Naive Bayes [126] and C4.5 [104, 105], for instance. Commonly, the step to process numeric features is normally separated from the learning algorithm. A discretising algorithm can transform raw inputs to discretise values in a pre-processing step that is independent from the learning algorithm. By having the step to process numeric features, an algorithm may process data examples with or without numeric features, by learning the transformed data examples which would just have categorical feature. Even if an algorithm can work with numeric features natively, Dougherty et al. [42] shows that in certain cases, learning from transformed/pre-categorised data examples can lead to a better result.

Methods for working with numeric features have been studied extensively in the past two decades for batch data. An overview of well-known methods for data discretisation for batch setting is described in [65]. However, this is not the case for data streams, where data examples are constantly arriving at high speed, while the learning process is iteratively repeated. Only a small number of discretising methods for batch settings can be directly applied to streaming data settings without any modifications and, consequently, they are unlikely to perform as well as they do on static data.

From the best of the author's knowledge, there are very few techniques specifically designed to deal with numeric features in a streaming environment. The work from Gama and Pinto [56] is a rare example of incremental discretisation methods that are intended to operate on data streams. This discretising technique is split into two main layers. The first layer processes values from a given numeric feature and generates some statistical data about the seen values without storing these values. From the memorised statistics

in layer one, the final discretisations are created by either equal width or equal frequency in the second layer.

3.2 Streaming Approaches

This section focuses on the problem of complexity in computational efficiency when generating rules from numeric features, especially, for "Separate-and-Conquer" search based algorithms from streaming data sources.

3.3 Computational Issues in Dealing with Numeric from a Streaming Data Source

As described in **Section 2.4**, the original PRISM [26] algorithm works on categorical features only and produces a set of rule term of the form $(\alpha = c)$. This limitation was overcome by eRules [115] and Very Fast Decision Rules (VFDR) [57], which are among few rule-based algorithms specifically developed for learning rules directly from streaming data sources. For numeric features, eRules and VFDR produce rule terms of the form *($\alpha <$ numeric_constant)* or *($\alpha \geq$ numeric_constant)*, and *($\alpha \leq$ numeric_constant)* and *($\alpha >$ numeric_constant)* respectively.

Generally, the process of how eRules algorithm and VFDR algorithm deal with numeric features in a streaming environment can be described as follows:

Algorithm 6: A general pseudo-code of how eRules and VFDR deal with numeric features.

For a given set/sub-set of training data examples:

1 For each possible value, α_i, of a numeric feature, α, calculate the conditional probability for a given target class;

2 Return the rule term with the overall best conditional probability for a given target class;

It is evident that the above process in dealing with numeric features requires many cut-point calculations for the conditional probabilities for each possible value, α_i, for a numeric feature, α.

Feature X (Numeric)	**Feature Y** (Categorical)	**Class Label** (A or B)
1	YES	B
2	NO	A
3	NO	A
4	YES	B
5	NO	A
6	NO	B

Categorical Feature:

$P(class = A \mid Y = YES) = 0$
$P(class = A \mid Y = NO) = 1.0$

2 Calculations

Numeric Feature:

$P(class = A \mid X \leq 1) = 0$	AND	$P(class = A \mid X > 1) = 1.0$
$P(class = A \mid X \leq 2) = 0.34$	AND	$P(class = A \mid X > 2) = 0.66$
$P(class = A \mid X \leq 3) = 0.66$	AND	$P(class = A \mid X > 3) = 0.34$
$P(class = A \mid X \leq 4) = 0.66$	AND	$P(class = A \mid X > 4) = 0.34$
$P(class = A \mid X \leq 5) = 1.0$	AND	$P(class = A \mid X > 5) = 0$
$P(class = A \mid X \leq 6) = 1.0$	AND	$P(class = A \mid X > 6) = 0$

12 Calculations

Figure 3.1: Example of a large number of cut-point calculations to learn heuristic form numeric features even for a very simple dataset.

Considering the example **Figure 3.1**, there is a fictitious dataset with just six observations, one categorical feature, one numeric feature, and two class labels. It shows how many cut-point calculations eRules or VFDR need to perform in order to search for one best rule term. The number of cut-point calculations needed for a numeric feature is the number of unique values of the feature multiplied by two. Clearly, for such a small number of data examples and minimised number of features and class labels, both eRules and VFDR required a significant number of calculations, which is costly in terms of computational performance. Additionally, both eRules and VFDR rely on the "Separate-and-Conquer" search approach, which may require many iterations to introduce a complete rule.

3.4 Using Gaussian Distribution to Discriminant Classification

There is a major concern in computational efficiency of how two notable streaming rule-based algorithms, eRules and VFDR, deal with numeric features. A new heuristic method is proposed based on Gaussian distribution to improve the computational efficiency when dealing with numeric features.

Gaussian or normal distribution can be regarded as the most important and most widely used distribution in statistics. The name 'Gaussian' is after mathematician Karl Friedrich Gauss [85], but some of the properties of Gaussian distribution were actually realised and defined by Abraham de Moivre [84].

The density diagram of the spread of the data from a Gaussian distribution often form a bell-shaped curve as shown in **Figure 3.2**.

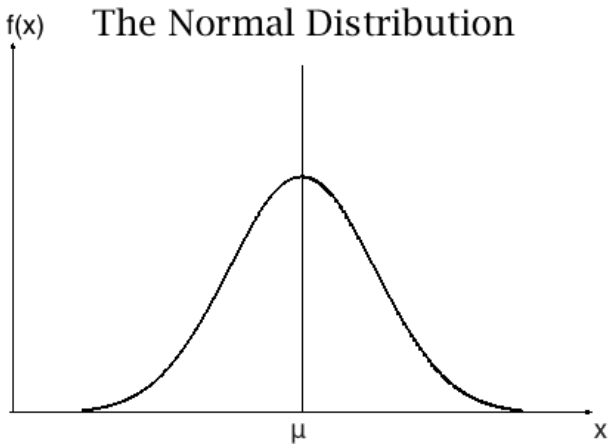


Figure 3.2: Density distribution has a bell shape curve when the values are normally distributed.

The usefulness of the Gaussian distribution comes from the fact that the distributions of many natural phenomena in real-life are at least approximately normally distributed [84]. Nowadays, many natural measurements in daily life are normally distributed, such as human weight, height, and blood pressure. However, one of the first known uses of the Gaussian distribution was to analyse the errors in measurements made in astronomical observations, which happened because of defective instruments and imperfect observers [84].

Because of its popularity and proven accuracy in many applications, a method relying on Gaussian distribution of values from numeric features associated with a given target class is proposed to avoid frequent cut-point calculations, as described in the method used by the eRules and VFDR algorithms.

Typically, for each numeric feature from training data examples, a Gaussian distribution can be generated, as shown in **Figure 3.3**, to represent all possible values of that numeric feature for a target class.

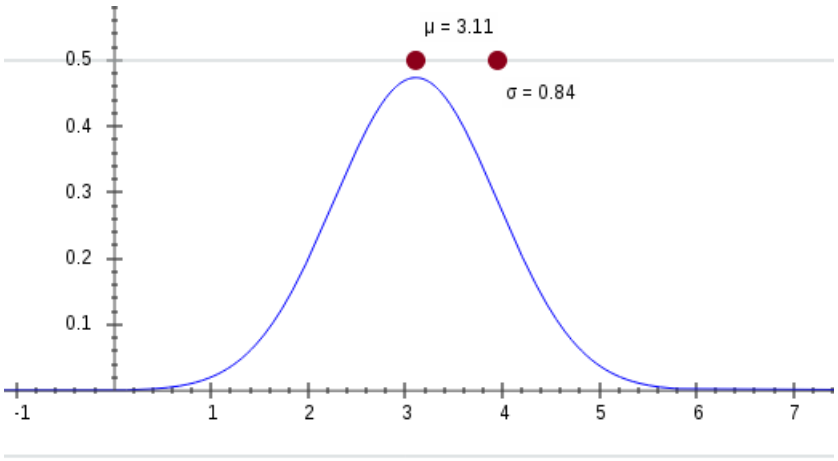


Figure 3.3: Example of a Gaussian distribution for a class label from the values of a numeric feature.

For a given dataset with classification labels, $\omega_1, \omega_2, ..., \omega_i$, if there is a measurement vector (values of a numeric feature), α, then a specific value of the numeric feature can be calculated, displaying whether it is the most relevant to a particular classification label, based on the Gaussian distribution of the values associated with this particular classification label.

The Gaussian distribution is calculated for a numeric feature, α, with mean, μ, and variance, σ^2, from all numeric values with a particular classification label, ω_i. The class conditional density probability is then, given by **Equation 3.1**.

$$p(\alpha_j|\omega_i) = p(\alpha_j|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(\alpha_j - \mu)^2}{2\sigma^2}) \tag{3.1}$$

Hence a heuristic measurement of posterior class probability, $p(\omega_i|\alpha_j)$, or equivalently $log(p(\omega_i|\alpha_j))$ can be calculated and used to determine the probability of a target class for a valid value of a numeric feature as, in **Equation 3.2**.

$$log(p(\omega_i|\alpha_j)) = log(p(\alpha_j|\omega_i)) + log(p(\omega_i)) - log(p(\alpha_j)) \tag{3.2}$$

The probability of regions, Ω_i, are calculated for these numeric values, such that if $\alpha_i \in \Omega_i$ then, α_i, belongs to class, ω_i. This approach may not necessarily capture the full details of the intricate continuous distribution, but it is highly efficient with respect to computation and memory perspectives. This is because the Gaussian distribution only needs to be calculated once and can then be updated when new data stream examples are received, by simply recalculating mean, μ, and variance, σ^2.

The range of values, which extends to both sides from μ, of the distribution should represent the most common values of α, for a target class, ω_i. A candidate rule term can be generated by selecting an area under the curve for a range of values for which the density class probability $p(x < \alpha \leq y|\omega_i)$ is the highest, where, x, and, y, are valid values of numeric feature, α, from the Gauss distribution for the target class, ω_i. As shown in **Figure 3.4**, the shaded area represents the highest density class probability $p(x < \alpha \leq y|\omega_i)$ of a subset from the training dataset.

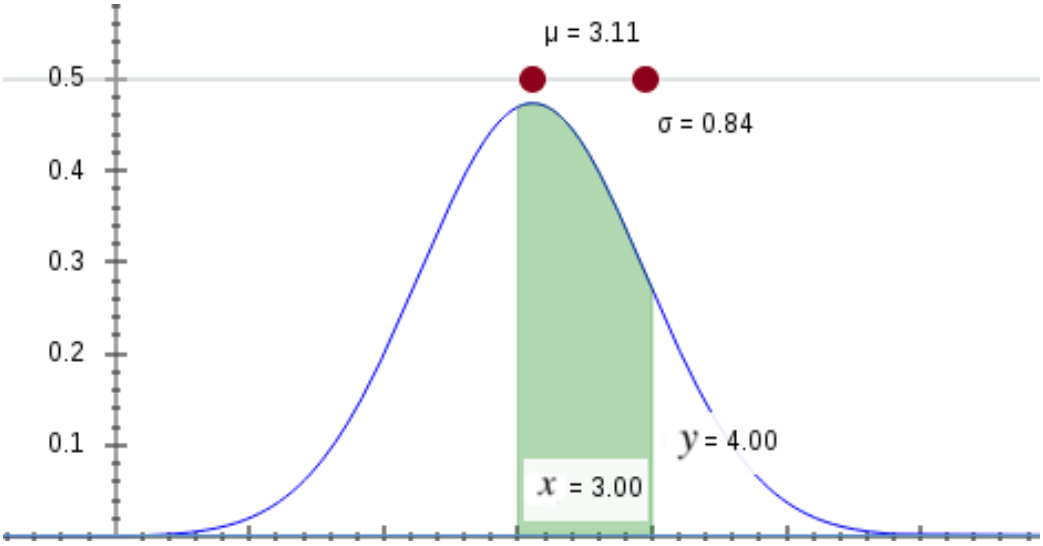
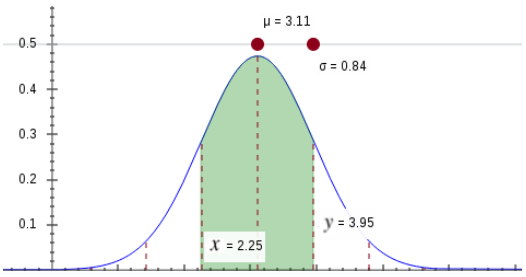
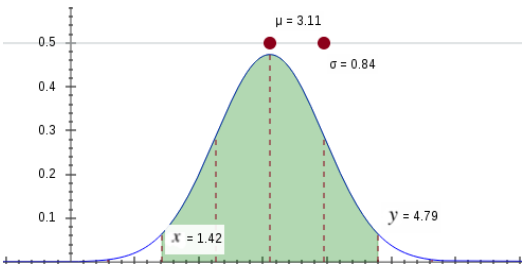


Figure 3.4: Shaded area represents a range of values of numeric feature, α, for class, ω_i.

Globally, the area in the centre under the curve represents the most common values of the feature for a target class. For example, the shaded area of one standard deviation of the mean ($\mu \pm 1\sigma$), as illustrated in **Figure 3.5a**, covers 68% of all possible values of the feature for the target class; or, as illustrated in **Figure 3.5b**, 95 % of all possible values of the feature for the area of ($\mu \pm 1.96\sigma$) [24].



(a) 68% of all possible values



(b) 95% of all possible values

Figure 3.5: Shaded areas represents ranges of values of feature α for class ω_i.

However, distributions for different classifications can also overlap each other. An area of distribution for a target class sometimes cannot be used to precisely distinguish a classification, as shown in **Figure 3.6**.

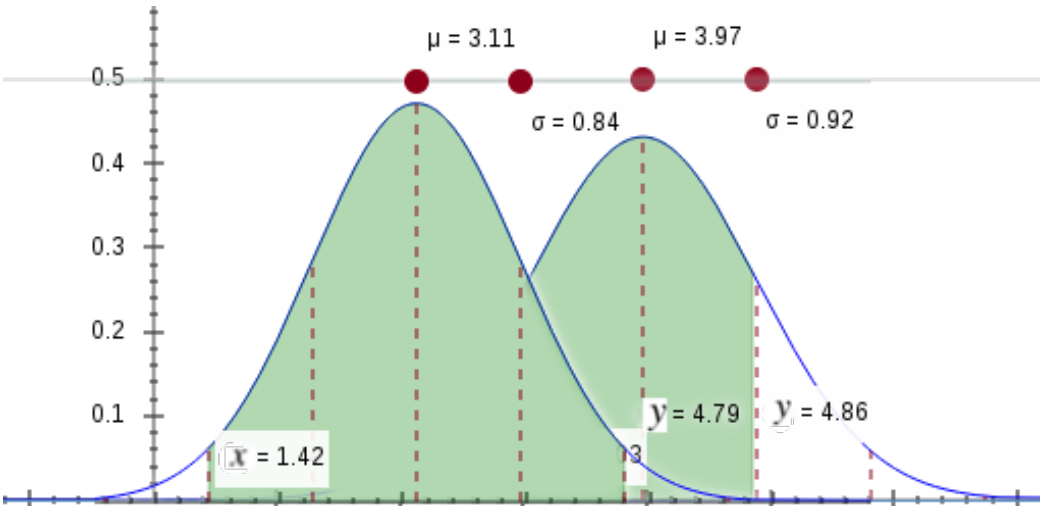


Figure 3.6: Distributions of different classification overlapping.

However, the most interesting rule term is one that can maximise the coverage of the rule for a target class. Therefore, this approach uses density estimation to discover a rule term of the form of $(x < \alpha \leq y)$ by selecting only a highly relevant range of values from a numeric feature, which can then be used to represent a subset of data examples for the target class, along with other rule terms.

Algorithm 7: A general pseudo-code of how to generate a rule term using the proposed method in this chapter.

1 For each numeric feature, calculate a Gaussian distribution with mean, μ, and variance, σ^2, for each classification;
2 Calculate class conditional density and posterior class probability for each numeric feature value for the target class, using **Equations (3.1)** and **(3.2)**;
3 Select the value of the feature with greater posterior class probability;
4 Select the next smaller and larger values from the value chosen in **Step 3** which have the greatest posterior class probability;
5 Calculate density probability with two values from **Step 4** from the normal distribution for the target class;
6 Select the range of the feature ($x < \alpha \leq y$) as the rule term for which density class probability is the maximum;

A high level of the pseudocode is expressed in **Algorithm 7**. This generic way of extracting numeric rule term can be used in any rule-based streaming algorithms in order to increase the data throughput. Next, in **Section 3.5**, the proposed method is incorporated with the eRules algorithm, with empirical experiments to show the superior compared to the existing mechanism in dealing with numeric features.

3.4.1 Illustrations of Rule Term for Numeric Features

To illustrate the described approach above for using Gaussian distribution to generate a rule term for a numeric feature, a simple three classes scenario is created to demonstrate the proposed process compared with the binary-split approach.

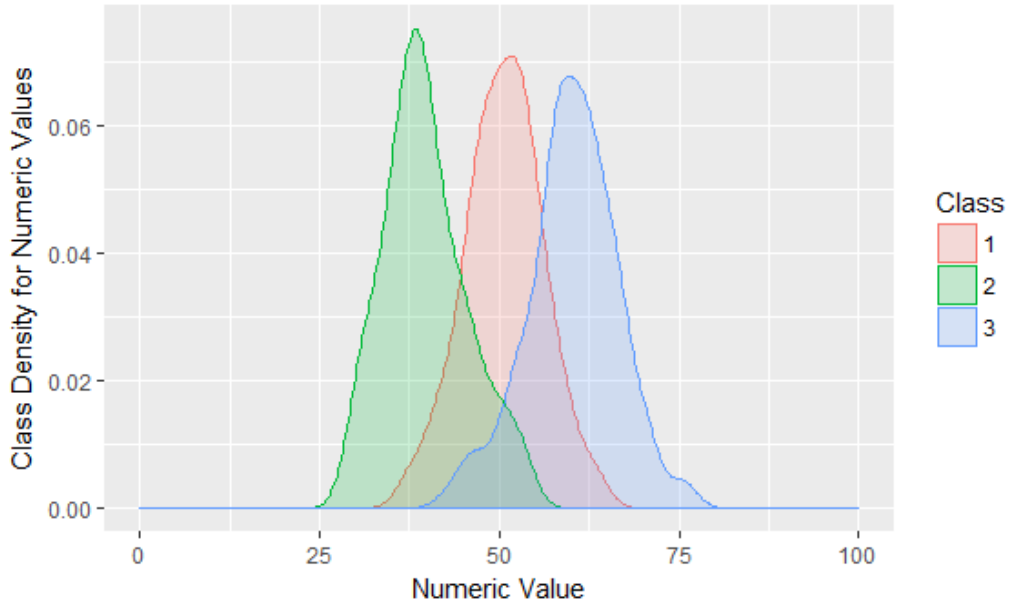


Figure 3.7: Class density of a numeric feature.

As shown in **Figure 3.7**, this is a fictitious example that shows how values of a numeric feature spread for each class label. The example below will consider the process to create a rule term for the feature in **Figure 3.7**, with the approach as proposed in **Section 3.4**, and the well-known binary tree split approach used in the eRules [115] and VFDR [57] algorithms.

3.4.1.1 Binary-Split Rule Term

Depending on used heuristic, by using binary split, an algorithm can select an optimal split value of a numeric feature to create a rule term of the form $(\alpha \leq v)$ or $(\alpha > v)$. However, the binary split approach is unlikely to distinguish a class from others if the class distributions overlap, as shown **Figure 3.7**.

For instance, when selecting the first quantile for the distribution of class 1 in **Figure 3.8** as the split point, in this example, the value of the first quantile for class 1 is 47.

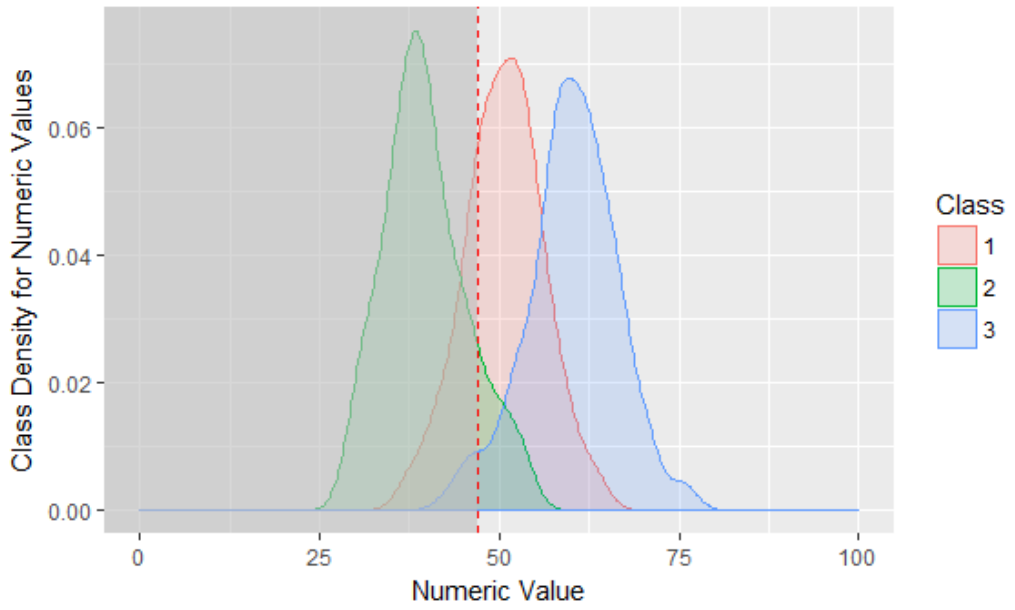


Figure 3.8: Split value for class 1 at first quantile.

Considering the split value of the feature for class 1 as shown in **Figure 3.8**, if the split is at the first quantile of the distribution of class 1, then there are two possible rule terms for this feature, $(\alpha \leq 47)$ or $(\alpha > 47v)$. In

either term, it is very difficult to separate class 1 from other classes. The non shaded area annotates the coverage for ($\alpha > 47$). As shown, the ($\alpha > 47$) rule term well covers the data examples for class 1. However, because of the structure of the rule term, ($\alpha > 47$) also covers a major number of data examples from class 3.

The example for binary split based rule term in this section clearly shows the limitation of binary split approach, and how difficult it is to separate data examples when the underlying distributions of the classes overlap. In particular, in streaming environments, the distributions will be dynamic and unstable as the underlying distributions from seen data examples can only be estimated.

3.4.1.2 Gaussian Based Rule Term

Following the processes described in **Algorithm 7**, the rule term ($51 < \alpha \leq 53$), is created for class 1 as shown in **Figure 3.9**. While it is impossible to create a rule term that can totally separate class 1 from other classes, the false coverage is minimised compared with a binary split approach.

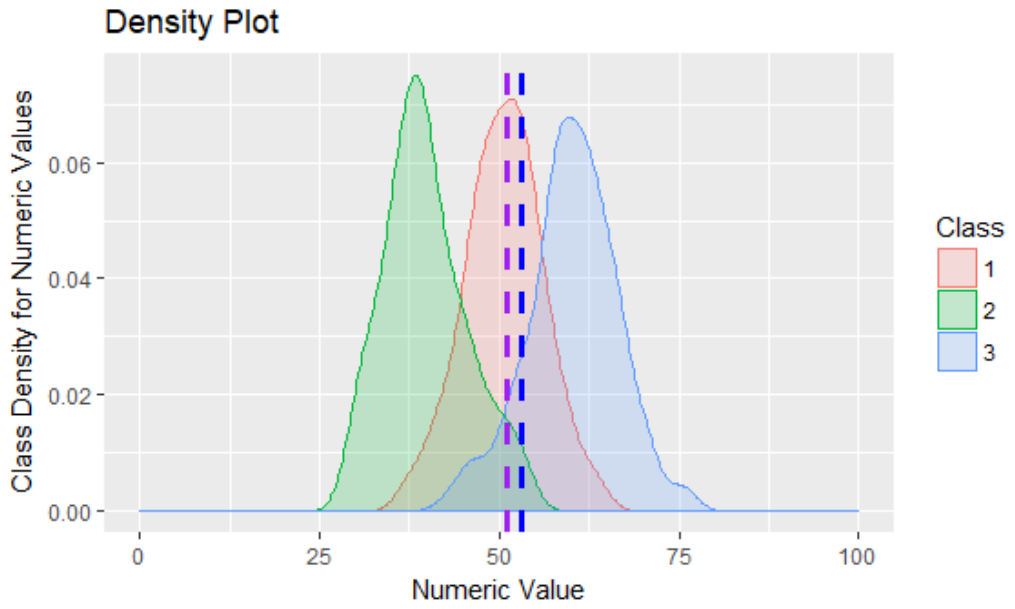


Figure 3.9: Rule term based on Gaussian distribution as described in **Algorithm 7**.

The above example assumes that all numeric values of the feature are available to create a rule term, but in practice, only valid numeric values from the feature at the point of assessment are considered. Iteratively, the training set will be reduced, but the rule term from the proposed approach will be generated with the greatest relevance to the target class, as shown in **Figure 3.9**.

3.5 G-eRules: eRules Algorithm and an Improved Approach for Processing Numeric Features.

An extended algorithm based on e-Rules is termed G-eRules, which is based on the method as proposed in **Section 3.4** to create heuristic for numeric features in order to make the eRules algorithm computationally more efficient. While the proposed method for numeric features was integrated with the eRules algorithm, it could also be adopted as the mechanism to deal with numeric features in other rule-based and tree-based streaming classifiers.

In G-eRules, the algorithm still utilises the "Separate-and-Conquer" search strategy by expanding one single successor feature-value at a time, thereby learning one complete rule from the training data at a time. After each complete rule is introduced, all data examples that are covered by this rule are removed from the training set, and the procedure is repeated. The key innovation compared with the original eRules algorithm is that the algorithm will create a distribution for each possible class label for each numeric feature at the beginning the of the learning process. Subsequently, the conditional probability of a value from a numeric feature for a given class label can be quickly calculated during the rule term selection process.

Algorithm 8 outlines the process in creating heuristics for numeric features, which is integrated in the eRules algorithm, termed G-eRules.

Algorithm 8: G-eRules induction approach for numeric features.

```
1  for i = 1 → C do
2      D ← original Dataset
3      while D contains classes other than ω_i do
4          forall α in D do
5              calculate mean μ and variance σ^2 of numeric feature α for
                class ω_i;
6              foreach value α_j of feature α do
7                  Calculate p(α_j|ω_i);
               end
9              Select α_j of feature α, which has highest value of p(α_j|ω_i);
10             For values < α_j select the value x that has the highest
                probability density in this range, and for values ≥ α_j select
                the value y that has the highest probability density in this
                range;
11             Calculate p(x < α ≤ y|ω_i);
           end
13         Select (x < α ≤ y) for which p(x < α ≤ y|ω_i) is a maximum;
14         Create subset S of D containing all the instances which has
            (x < α ≤ y);
15         Build a rule term describing S;
16         D ← S;
       end
18     The induced rule, R is a conjunction of all the rule terms built at line
        15;
19     Remove all instances covered by rule R from original Dataset.;
20     repeat
21         lines 2 to 19;
       until all instances of ω_i have been removed;
23     Reset original Dataset to its initial state;
   end
```

The principal objective of the presented G-eRules algorithm in this section is to show the effectiveness of the proposed approach in dealing with numeric features. Computational performance and competitiveness in terms of the accuracy compared with other established data stream classifiers are evaluated with different settings as well as both synthetic and real datasets. The implementation of G-eRules was coded with Java so that it could be used within the Massive Online Analysis (MOA) framework [21], a workbench for evaluating data stream mining algorithms. MOA was chosen as it natively contains the implementations of data stream algorithms, such as VFDT (Very Fast Decision Tree) or VFDR (Very Fast Decision Rules). The analysis of the empirical experiments is presented in **Chapter 6**.

3.6 Normality Test

The described approach in dealing with numeric features is however based on an assumption that the values from numeric features come from a normal distribution or Gaussian distribution. However, if the assumption is not taken seriously, then the proposed approach in this chapter should not be used instinctively, as this learnt model will not draw accurate and reliable conclusions about the actual distribution of numeric features. In other words, if the user is certain about the true distribution of a numeric feature that is not normal, then different approaches should be considered.

On the other hand, the normality assumption is more valid in streaming settings as the assumption of an unbounded arriving data example is one of the main characteristics for streaming data, as stated in **Section 2.3**. As

shown in [99], it is common that many statistical techniques assume that the distribution of real values is 'normal' if large enough sample sizes are used ($>$ 30 or 40), and the true distribution is uncertain. Furthermore, Altman and Balnd [9] showed that the distribution of data can be ignored if the samples consists of hundreds of observations as well as samples from a true normal distribution, although these may not necessarily look normal themselves. Additionally, there are few notable points from the central limit theorem [9, 43] regarding the normality assumption as follows:

- If the sample is approximately normal, then the sampling distribution will also be normal.
- If the sample size is reasonably large enough, then the sampling distribution tends to be normal, regardless of the actual underlying distribution of data.
- Means of random samples from any distribution will themselves have normal distribution.

From the stated points above, in a streaming environment in particular, true normality is considered to be a myth, but a good estimation of normality can be confirmed by using visual plots or significant tests. The main idea behind these methods is to show whether data seriously deviates from normality, then confirm the normality [99, 9, 43].

In most cases, the buffer for arriving data examples contains more than 30 or 40 data examples, and it is acceptable to assume the that values for numeric feature come from a normal distribution unless proven otherwise.

3.7 Summary

This chapter studies the challenges in dealing with numeric features in data streams for the purposes of inducing rules from algorithms based on the "Separate-and-Conquer" approach in **Section 3.3**, as well as develops an approach to effectively and efficiently dealing with numeric features in streaming settings. This is the primary concern in one of the raised research questions in **Section 1.2**:

*"**Research Question 2**: What are the most efficient approaches to improve the efficiency in learning heuristic for numeric features from a high-volume and high-velocity streaming data source?"*

The difficulties in dealing with numeric features were discussed, and how the existing methods for batch settings are not applicable in a streaming environment any more. Approaches in eRules [115] and VFDR [57] were also investigated with some practical examples to show inefficiency, drawbacks with a binary-split approach, and the need for a better approach. As shown, it is a very expensive process to search for best split from numeric features if the main algorithm employs the "Separate-and-Conquer" search strategy, because the calculation for all values will need to be repeated after each iteration until the learning of the rule has been completed or the stopping criteria is reached.

Subsequently, a new computationally efficient method of extracting rule terms in the form $(x < \alpha \leq y)$ from numeric features in the data stream for classification applications was proposed. The proposed approach utilizes the density probability values from Gaussian distributions of the class labels to

extract the rule terms. This approach is generic and independent algorithms can adapt and incorporate it into the learning process easier. **Section 3.5** shows the example in incorporating the proposed approach in dealing with numeric features in the eRules algorithm, a simple yet competitive rule-based data stream classifier, which allows abstaining from a classification, but is computationally inefficient when dealing with numeric features. The new version with the method as proposed in **Section 3.5** is termed G-eRules.

Chapter 4

A new Dynamic Sliding Window Technique with Hoeffding's Inequality

A more dynamic and robust method for buffering recent data examples is developed in this chapter by incorporating the property of Hoeffing's Inequality with the sliding window technique.

The sliding window technique in learning from recent data examples has been discussed in **Section 2.4**. While this approach shows a good way to work with data examples in streaming settings, it is not clear how much recent data to learn from is the main topic in this chapter. This is the primary concern in two of the raised research questions in **Section 1.2**:

*"**Research Question 3**: What impact have the properties of streaming data sources had on the faith and accuracy of the predictions for classification*

tasks?"

*"**Research Question 4**: How have concept drifts and dynamic behaviours in streaming data been posing challenges in learning a reliable and accurate classification model?"*

4.1 Issues with Fixed Sliding Windows Technique in Streaming Environment

Typically, the size of a window or how much data to buffer before learning is set by the user before the learning process starts. The size of a window is fixed through the learning process, and there is unlikely to be a global optimal window size that will work well on all streaming sources. In other words, a fixed size sliding window contains 'n - pre-defined' number of data examples or those data examples that have arrived during the last 't - pre-defined' time window. This approach could work well if the occurring concept drift is known in advance, but practically, this is rarely the case.

If the window size is not optimal such as:

- **Too Small**: The algorithm will not be able to learn the underlying concept of the data.
- **Too Large**: The learning time for each window can increase and the outdated concept (in case of concept drift) will also be learnt.

Data examples from a sliding window are used to induce classification but

this process does not take into account the possibility of concept drift. The problems with fixed-size sliding window can be illustrated as in **Figure 4.1**.

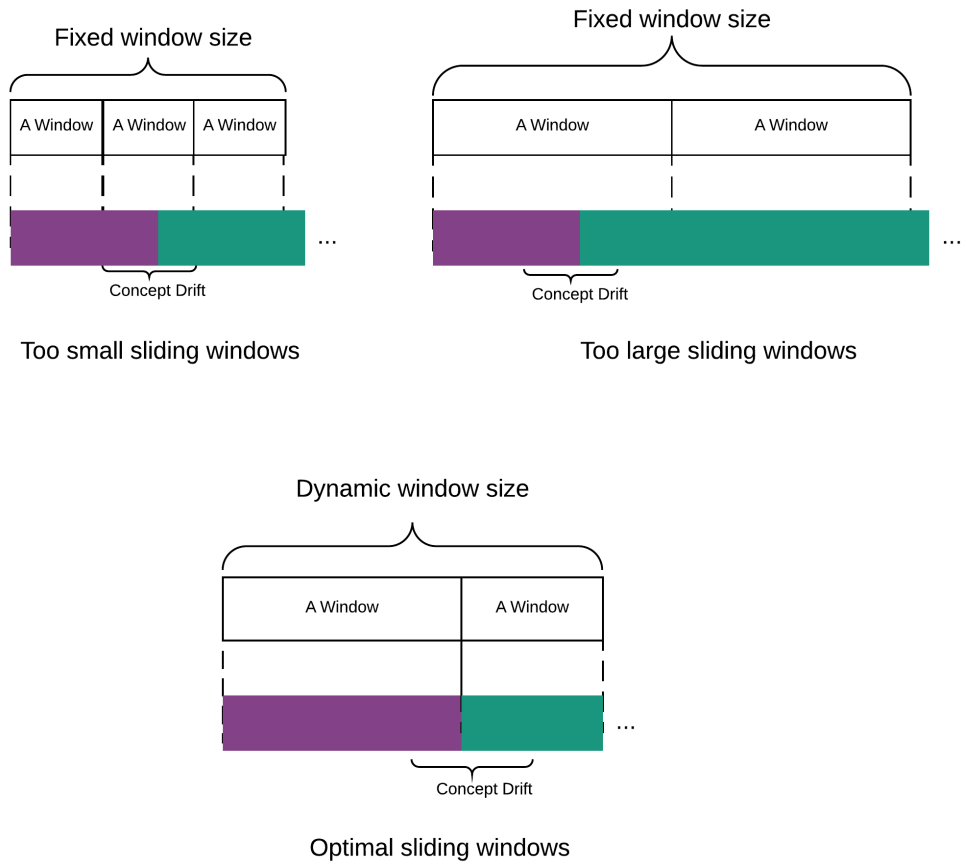


Figure 4.1: Different scenarios with a fixed sliding window when encountering concept drifts.

In a streaming environment, it is important to learn and maintain the concepts that reflect the current state of the data, while not including that which happened in the past and is no longer valid. One way to achieve this objective is to calculate a certain level of confidence in the number of

observed data examples.

The Hoeffding Trees algorithm [40] also has the mechanism to decide how many data examples are needed before splitting at an estimated best feature at any given moment in time.

This chapter looks at the applications of Hoeffding's Inequality [67] in general, and how it can be used to estimate the confidence in whether adding a rule term to a rule currently being induced or stopping the induction process is appropriate.

4.2 A new Dynamic Sliding Windows Technique with Hoeffding's Inequality

The idea to use Hoeffding's Inequality was inspired from [67, 91, 40]. Hoeffding's Inequality provides a statistical measurement in confidence of the sample mean of n independent data examples $x_1, x_1..., x_n$. If E_{true} is the true mean and E_{est} is the estimation of true mean from an independent sample then the difference in probability between E_{true} and E_{est} is bounded by:

$$\mathbb{P}[|E_{true} - E_{est}| > \epsilon] < 2e^{-2n\epsilon^2/R^2} \tag{4.1}$$

Where R is the possible range of the difference between E_{true} and E_{est}. From the bounds of the Hoeffding's Inequality, it is assumed that with the confidence of $1 - \delta$, the estimation of the mean is within ϵ of the true mean:

$$\mathbb{P}[|E_{true} - E_{est}| > \epsilon] < \delta \tag{4.2}$$

From **Equations 4.1** and **4.2** and solving for ϵ, a bound on how close the estimated mean is to the true mean after n observations with the confidence of at least $1 - \delta$ is illustrated as follows:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{4.3}$$

By using the Hoeffding Bound as an independent metric to verify the true likeness of a rule term , the rules satisfy the Hoeffding Bound then they are likely to be as good as the rules learnt from an infinite data stream.

ϵ is calculated after a rule term with best conditional probability for class ω_i selected. However, the rule term will be added to the current rule unless the difference of the conditional probabilities between the selected best and the second best rule term is greater than ϵ. Otherwise, the rule's induction process is completed and the rule is added to the rule set. A new iteration for a new rule is started again, with data examples covered by the previous rule removed.

If $G(t_\alpha)$ is the heuristic measurement that is used to test rule term t_α, then R in **Equation 4.3** represents the range of $G(t_\alpha)$. $G(t_\alpha)$ in our approach is the conditional probability $\mathbb{P}(class = i|t_\alpha)$ at which rule term t_α covers target class ω_i. Hence, the probability range of rule term R is 1. n is the number of data examples that the rule has covered so far.

Concerning the suitability of the best rule term, let $t_{\alpha_{best}}$ be the rule term with the highest conditional probability from the current iteration, and $t_{\alpha_{j-1}}$ be the rule term with the second highest conditional probability from the current iteration, then:

$$\Delta\overline{G} = \overline{G}(t_{\alpha_j}) - \overline{G}(t_{\alpha_{j-1}}) \tag{4.4}$$

If $\Delta\overline{G} > \epsilon$, then the Hoeffding Bound guarantees that with a probability of $1 - \delta$, the true $\Delta\overline{G} \geq (\Delta\overline{G} - \epsilon)$.

Essentially, the Hoeffding Bound is used to determine a probability with the confidence of $1 - \delta$ that the observed conditional probability, in which the rule term covers the target class in n examples, is the same as would be observed for an infinite number of data examples.

4.3 Illustration of Hoeffding's Inequality

The mathematical proofs of Hoeffding's Inequality are well described in [69, 91]. Additionally, this section demonstrates a very simple scenario to prove the aforementioned characteristics of Hoeffding's Inequality in **Section 4.2**.

4.3.1 Illustration Preparation

Problem: There is a vector of letters 'A' and 'B' and the probability of 'A' is exactly 0.30. In other words, if the vector size is 10,000 then there are 3000 'As' and 7000 'Bs'.

Hypothesis: The concept of **Equation 4.3** in this scenario can then be applied to calculate the probability of 'A' from the given vector. Although it is known that the real probability of 'A' is 0.3, assuming that this value is unknown and the probability of 'A' is calculated from sampling values from the vector.

Defining Parameters: In this example, R = 1.0 because the probability of 'A' can only be between 0.0 and 1.0, therefore, the range, R, should be 1.0. δ and n are 0.05 and 200 respectively. However, n and δ are configurable and different values can be used.

Description: Based on the selected parameters, where n = 200, R = 1.0 and $\delta = 0.05$, then ϵ can be calculated. Contextually, it can be interpreted that the estimated probability of 'A' from any 200 values sample from the initial vector of 10,000 should be smaller than a certainty of 0.95. In other words, if the probability of 'A' is calculated from a sample of 200 random values from the original vector and the process is repeated 100 times, then it should be expected that at least 95 times the difference between the estimated probability of 'A' from the sample and the actual probability of 'A' should be constrained by ϵ.

The code for this experiment can found in **Appendix C** to reproduce the stated experiment.

4.3.2 Confirming the Property of Hoeffding's Inequality in Integrating with Sliding Window Technique

From the experiment setup in **Section 4.3.1**, if Hoeffding's Inequality is valid, then an error rate of a run should not excess 0.05 (5%), and this should be valid for an infinite number of runs if the underlying concepts/distributions are static.

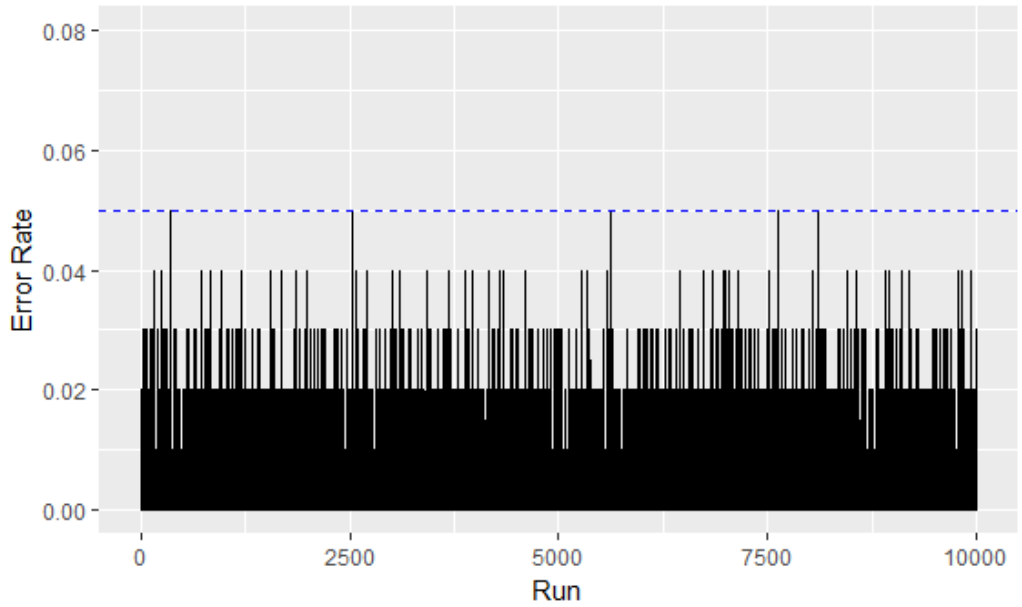


Figure 4.2: Error rate for 10,000 runs of the sample experiment as described in **Section 4.3.1**.

Figure 4.2 showed that the error rate was absolutely respected for $10,000$ runs, which confirmed the stated characteristics of Hoeffding's Inequality.

The empirical experiment in this section provides a concrete foundation to form a more robust and dynamic rule algorithm for streaming in the next

chapter, named Hoeffding Rules.

4.4 Summary

In a streaming environment, new data examples will continuously arrive. Both the computational and storage costs will become impractical over time. It is important to get the right amount of training data examples to ensure the validity of the learning concept over time.

This is the primary concern in two of the raised research questions in **Section 1.2**:

*"**Research Question 3**: What impact have the properties of streaming data sources had on the faith and accuracy of the predictions for classification tasks?"*

*"**Research Question 4**: How have concept drifts and dynamic behaviours in streaming data been posing challenges in learning a reliable and accurate classification model?"*

Therefore, by holding and analysing a small portion of seen data, the aforementioned challenge can be overcome. The sliding window technique is a well-known technique, where only a number of the most recently observed data examples are kept in memory or appropriate secondary storage for learning and validating purposes. Additionally, the learning process is made easier and more substantial by the that fact that the number of data examples will not cause major challenges in computing and storage costs.

Practically, there is no 'magic number' that can be pre-determined to guarantee capturing the underlying concept of data sources. Also, the uniformity of the data sequence can vary between scenarios. Along with the key concepts of the sliding window technique, this chapter introduced an addition to improve the robustness and efficiency of the sliding window technique when learning from a data stream. Hoeffding's Inequality was applied to determine a suitable window size for a given error rate to reflect an underlying concept from a sub-sample set (window size) of data examples. Rather than relying on a static size for the windows throughout the learning process, the window size is determined throughout the learning process. For instance, if Hoeffding's Inequality is not satisfied, then learner would need to buffer more data examples before the learning process can be started. However, as mentioned earlier, it is not practical to have an indefinite window size because of computing cost and storage as well the potential existence of concept drifts in the data.

Subsequently, **Chapter 5** presents the use of a dynamic sliding window along with the G-eRules algorithm described in **Chapter 3** to form a more comprehensive and robust classifier for streaming data, termed *Hoeffding Rules*.

Chapter 5

Hoeffding Rules Algorithm: Expressiveness and Uncertainty Awareness Rule-based Classifier for Data Streams

The advantages and importance of a modular rule-based classifier were discussed in **Chapter 2**, and the approaches to efficiently deal with numeric features and dynamic buffering from a streaming environment were outlined in, **Chapter 3** and **Chapter 4** respectively. This is the primary concern in two of the raised research questions in **Section 1.2**:

*"**Research Question 2**: What are the most efficient approaches to improve the efficiency in learning heuristic for numeric features from a high-volume and high-velocity streaming data source?"*

*"**Research Question 4**: How have concept drifts and dynamic behaviours in streaming data been posing challenges in learning a reliable and accurate classification model?"*

The Expressiveness of decision models in data streams is an area of research that has attracted less attention, despite its practical importance. Moreover, rule-based models are another well-known alternative to the tree-based model because of its modular and expressive structure. The aforementioned properties of rule-based models are even more valuable factors when considering the suitability of model representation for learning from streaming data sources, which can help decision makers with informed predictions (white-box).

Furthering the works in **Chapter 3** and adapting the illustrated concept in **Section 4.3**, this chapter presents a completed rule-based classifier for streaming environment that adopts Hoeffding's Inequality to build decision rules. The proposed algorithm can help decision makers within informed predictions. The algorithm is termed *Hoeffding Rules*. Objectively, this chapter addresses the two remaining raised research questions in **Section 1.2**:

*"**Research Question 1**: How can a trained predictive model from a streaming data source be reliably interpreted and understood by the users?"*

*"**Research Question 3**: What impact have the properties of streaming data sources had on the faith and accuracy of the predictions for classification tasks?"*

In this chapter, the details of Hoeffding Rules algorithm is explained.

Section 5.1 describes the high-level skeleton of Hoeffding Rules algorithm and how different processes connect to others. Subsequently, **Section 5.1.1**, **Section 5.1.2**, **Section 5.1.3** and **Section 5.1.4** provide deep-dives into each component of Hoeffding Rules algorithm.

5.1 Overall Learning Process of Hoeffding Rules

This section highlights the development of Hoeffding Rules algorithm conceptually. It involves the induction of an initial classifier in the form of a set of expressive *"IF... THEN..."* rules. This section first highlights expressive rule sets in general and then discusses the Prism algorithm as a basic approach for inducing such rules on batch data in **Section 5.1.1**. Prism algorithm has been adopted by Hoeffding Rules as the basic process for inducing expressive rules. However, it has been enhanced with a more expressive rule term induction method for numeric features as described in **Section 3.4** based on probability density distribution. **Section 5.1.3** then describes the Hoeffding Inequality adapted by Hoeffding Rules algorithm as a metric to estimate a good dynamic window size of the data stream to induce expressive rules form. Lastly, **Section 5.1.4** illustrates the process of adding newly learned rules from streaming data.

Expressive classification rules are learned from a given set of labelled data examples, which consists of feature values and rule learning algorithms to construct one or more rules of the form:

$$\text{IF } t_1 \text{ AND } t_2 \text{ AND } t_3 \text{ ... AND } t_k \text{ THEN Class } \omega_i$$

The left side of a rule is the conditional part of the rule, which consists of a conjunction of rule terms, where a rule term t is a logical test that determines whether a data example to be classified has the classification ω_i or not. A classification rule can have one up to k rule terms, where k is the number of features in the data.

5.1.1 Inducing the Initial Classifier

The first step of Hoeffding Rules execution is the generation of the initial classifier, which is conducted in batch mode using Prism [26] on the first n seen data examples in the window. For the first window, the window size n is predefined by the user. Subsequently, the number of data example for each window consists of unseen data examples plus the data examples not covered by the rules from the previous window. The overall learning process of the Hoeffding Rules algorithm is as defined in **Algorithm 9**.

Algorithm 9: Hoeffding Rules - Inducing rules from an infinite data stream.

```
   R ← Learnt rule set;
   r ← A classification rule;
   S ← Stream of data examples;
   W_unseen ← Buffer of unseen data example;
   W_HB ← Buffer of data examples not covered by rules from previous
    W_unseen;
   n : pre-defined window size;
 7 while S has more data example do
 8     i → new instance from S ;
 9     if r ∈ R covers i then
10         Validate the rule r and remove if necessary;
       else
12         Add i to W_unseen;
13         if W_unseen = n then
14             W' := W_unseen + W_HB;
15             empty(W_unseen, W_HB);
16             Learn rule set, R', in batch mode as in Algorithm 10
                from W';
17             Add R' to R;
18             W_HB := data examples not covered by r ∈ R' in W';
           end
       end
   end
```

Algorithm 10: Hoeffding Rules - Inducing classification rules in batch mode.

```
1  for i = 1 → C do
2      D ← input Dataset;
3      while D contains classes other than ω_i do
4          forall α in D do
5              if α is categorical then
6                  Calculate the conditional probability, ℙ(ω_i|t_α) for all rule
                     terms t_α from on all feature-value, α_i from α;
                   else if α is numeric then
8                      calculate mean μ and variance σ² of numeric feature α
                         for class ω_i;
9                      foreach value α_j of feature α do
10                         Calculate ℙ(α_j|ω_i) based on created Gaussian
                             distribution created in line 8;
                       end
12                     Select α_j of feature α, which has highest value of
                         ℙ(α_j|ω_i);
13                     Create t_α in form of x < α ≤ y as described in
                         Section 3.4.1.2;
14                     Calculate ℙ(t_α|ω_i), where t_α is in the form of
                         x < α ≤ y;
               end
           end
17         Select t_α for which ℙ(t_α|ω_i) is a maximum;
18         Create subset S of D containing all the instances which covered by
             t_α;
19         D ← S;
       end
21     The induced rule, R is a conjunction of all the rule terms built at line
         17;
22     Remove all instances covered by rule R from input Dataset;
23     repeat
24         lines 2 to 22;
       until all instances of ω_i have been removed;
26     Reset input Dataset to its initial state;
   end
   return induced rules;
```

5.1.2 Evaluating Existing Rules and Removing Obsolete Rules

The evaluation and removal of rules are performed throughout the learning process. Once labelled data examples are available, the rules of the current classifier are applied to these data examples. Each rule remembers how many data examples that the rule has correctly and incorrectly classified in the past. With this information, a rule can actively update its accuracy after each classification attempt. If a rule's classification accuracy drops below a pre-defined threshold (by default 0.8) and the rule has also surpassed a minimum number of classification attempts (by default 5), then the rule is removed. The reason for considering a minimum number of classifications is to avoid the rule being removed too early.

For instance, if the rule's minimum number of classification attempts is only 1, then it would be removed if the first classification attempt failed. However, with the default settings, the rule would 'survive' 5 attempts. Assuming that 1 out of 5 attempts failed, then the rule would be retained, as it has an accuracy of $4 \div 5 = 0.8$, which is the minimum classification accuracy required.

The default settings may be adjusted according to the user requirements. A lower minimum accuracy will cause the classifier to adapt more slowly, however, a high accuracy may result in rules expiring quickly and thus more computing cost occurring to induce new rules. Also, a low number of minimum classification attempts will result in rules expiring quickly and a high number of minimum classification attempts will lead to a slower adaptation.

In the experiments, it was found that the default values worked well in most cases. these were used in all experimental results presented.

5.1.3 Buffering Data Examples that Do Not Satisfy the Hoeffding Bound

One of the notable features of Hoeffding Rules algorithm is the use of Hoeffding's Inequality, which is described in **Chapter 4** to determine the credibility of a rule term. For an algorithm based on the "Separate-and-Conquer" search strategy in batch data, a new rule term is searched and added to a current rule until the rule only covers data examples of the target class. However, the Hoeffding Rules algorithm does not always induce rules that cover only examples of the target class, because Hoeffding Rules will stop inducing further rule terms if the rule does not satisfy the Hoeffding's Bound metric from the current subset of data examples.

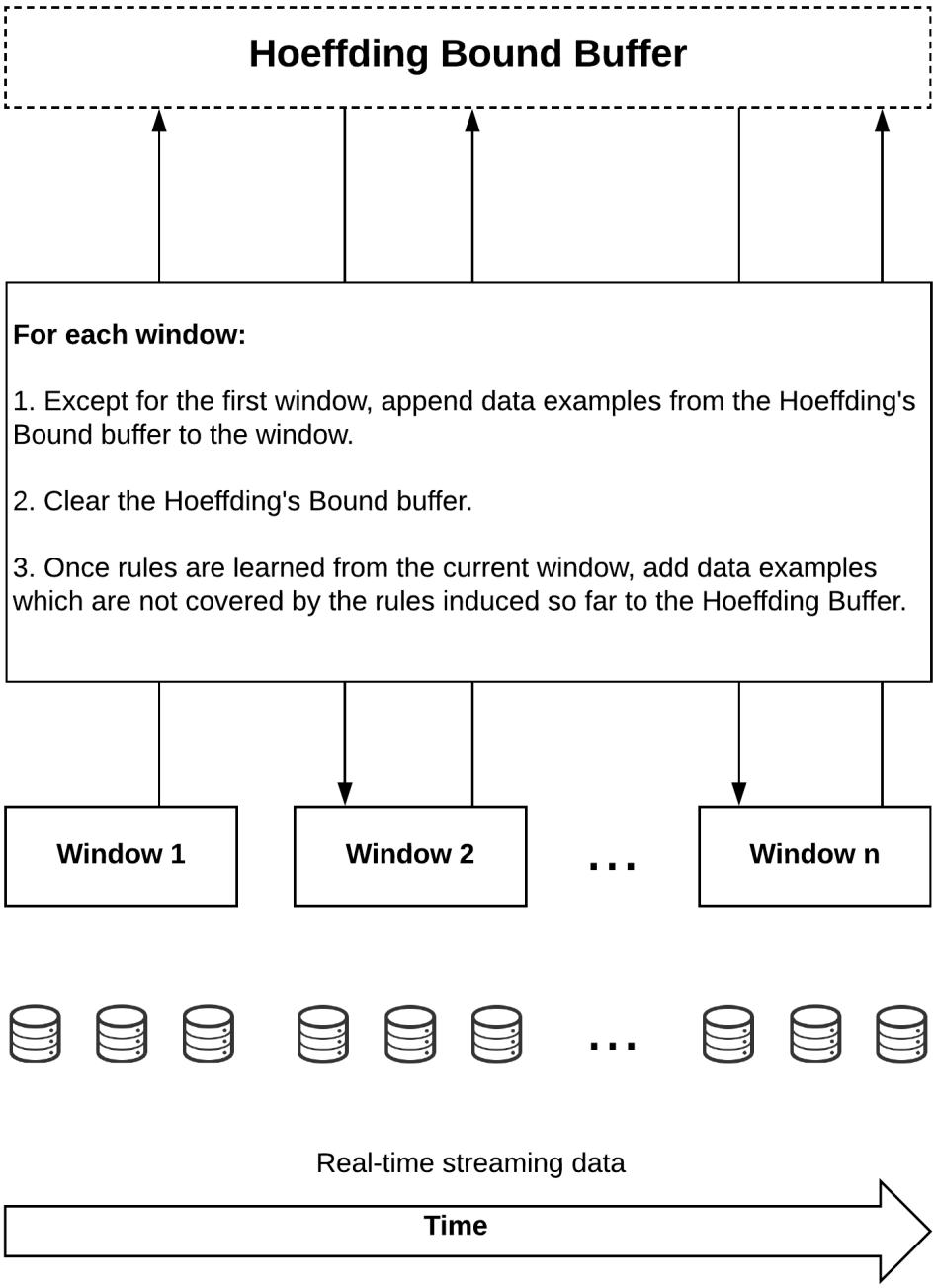


Figure 5.1: Combining data examples that satisfy the Hoeffding's Bound from the previous window with the unseen data examples from the current window.

As illustrated in **Figure 5.1**, once all possible rules are induced from the sliding window, then all data examples that are not covered by the newly created rules are stored in a buffer, which is combined with the next window of unseen data examples from the stream. Hence, after the first window, each sliding window is filled with unseen data examples from the window and the instances from the Hoeffding's Bound buffer from the previous windows. The Hoeffding Bound buffer contains instances that are not covered by the current rule set.

5.1.4 Addition of New Rules

The addition of new rules also takes place online. As outlined in **Algorithm 9**, Hoeffding Rules applies its current rules to new data examples that are already labelled, in order to evaluate the rule set's accuracy. However, if none of the rules applies to a labelled data example, this data example is added to the window. Once the window of unseen instances reaches the defined threshold, data examples are learnt as outlined in **Algorithm 9** to induce new rules, which are then added to the current classifier. Next, the window is reset by removing all data examples.

5.2 Summary

The Hoeffding Rules algorithm is developed in this chapter which focuses on producing an expressive rule set that is still robust with concept drift in real-time, but more expressive to users. Compared with less expressive streaming classifiers, Hoeffding Rules explains how a decision is reached. The algo-

rithm utilised the "Separate-and-Conquer" search strategy and Hoeffding's Inequality to learn new rules and adapt learned ruleset throughout the learning cycle. Inheriting, the feature from eRules and G-eRules, the Hoeffding Rules algorithm may also decide to abstain from classifying a data example if it has low confidence about a true class label. This again is desirable in applications where a false classification label could be very costly.

This chapter is motivated by the fact that rule-based data stream classification models are more expressive than other types of classification models such as the decision tree model, instances-based model, probabilistic model or neural network. Additionally, inducing a classifier from a data stream has some unique challenges compared with data mining from conventional batch data because the pattern encoded in the stream may change over time, which is known as concept drift as described in **Section 2.3**. Typically, most streaming classification techniques focus on achieving a high level of accuracy and quick adaptation to concept drift, and they are often unfriendly, difficult to interpret or too complicated to provide a trustworthy decision to the users, which is undesirable in many domains such as medical applications, finance or surveillance.

Collectively, Hoeffding Rules algorithm addresses the four research questions in **Section 1.2** about a system which can learn expressive rules effectively from streaming data sources.

Chapter 6

Empirical Evaluations and Discussions

This chapter presents the empirical evaluations and discussions of the novelty contributions in previous chapters under controlled and standardized evaluation procedures for classification tasks in a streaming environment.

It is essential to have evaluations to demonstrate the ability to meet the requirements of the end user and their desired application. The improved technique in dealing with numeric features in G-eRules algorithm is evaluated by looking at the integration of the technique to an existing algorithm, eRules. Hoeffding Rules algorithm is evaluated to demonstrate the capability of a new rule-based algorithm for classification tasks being able to answers the challenges of learning from streaming data sources. Results of the evaluation show that the proposed Hoeffding Rules algorithm significantly outperforms the baselines in terms of accuracy and learning time in comparison with other existing algorithms.

This chapter consists of five sections. **Section 6.1** describes the methodology for evaluation procedures for stream mining experiments. **Section 6.2** provides an of logical structure of the evaluations this chapter in in relation to address the raised research questions. **Section 6.3** critically review and discuss the outputs from the empirical experiments for G-eRules algorithm and Hoeffding Rules Algorithm. Finally, **Section 6.5** concludes the chapter.

6.1 Evaluation Methodology

The prequential approach is used to place various learning algorithms under test and systematically compared with others. The experimental methodology in this research is motivated by the requirements of the desire to answer the defined research questions.

6.1.1 Validation and Evaluation Procedures

For a classification task in the streaming environment, there is a particular set of requirements. They are a number of features per data example, a certain volume of data and a velocity at which data examples arrive. The behaviour of data stream algorithm has three dimensions of interest:

The accuracy of classified labels for unseen data examples.

The amount of space required to buffer training data examples for learning tasks at any given time.

The time required to learn from training data examples before the training data examples need to be discarded completely.

However, when the requirements cannot be practically relaxed, then the effectiveness of the learning algorithm can also be considered as a metric–the ability to minimize the overall error rate with limited time and limited space.

The error rate of an algorithm tends to be the typical metric which one wants to be able to control the most, but it also tends to be the most uncontrollable one. The most influential factor of error rate is the *representational power* of an algorithm, in other words, it is the capability of an algorithm to capture the true underlying concepts in a data stream, and the ability to generalize to ignore the noise and isolate useful concepts in the data.

Controlling the time and space required by an algorithm, one can influence the error rate. Space and time are independent. By keeping more pre-computed information, such as statistical meta-data or lookup tables, an algorithm can learn faster at the cost of space. On the other hand, an algorithm can also run faster by processing less information, either by storing less or stopping early, thus having less data to process. In short, the more time an algorithm has particularly, the more has to compute, or more information that is processed, the more likely it is that the error rate can be reduced.

Streaming environment has different requirements from the batch settings. Instead of maximizing data use, the attention focuses on the trends over time–where for static data, a single model is the desired final output of learning, whereas, in the streaming environment, the model evolves over time and can be employed at different stages of growth.

Prequential or 'Interleaved Test-Then-Train' procedures are selected for evaluating the developed algorithms in the research. Each data example can be used to test the model before it is used for training, and from this, the

error rate can be incrementally minimised and monitored over time. This scheme is particularly suitable for streaming algorithms evaluation because it makes maximum use of available data as well as measuring both time and accuracy. It also provides the capability to monitor a plot of the accuracy of over time, as data individual data example will become increasingly less important to the overall average.

6.1.2 Data Sources

Both real time and synthetic data generators are used for the experiments in this chapter. Whether a real life dataset or a synthetic data generator, the data examples are always generated on-the-fly, and the data examples arrive sequentially to the learning algorithm. This behaviour influences the amount of learning data examples that can be supplied.

In addition, the use of synthetic data generators has several advantages– it is easier to reproduce and controlled (speed, number of features, volume, function switch) and the cost in terms of storage and transmission is minimised. For the evaluation of this research, the data generators commonly found in the literature have been considered along with real dataset from the UCI [12] repository. The details of datasets and the data specification for each set of experiments are described in more details in the corresponding evaluations.

6.1.3 Evolving Stream Experimental Settings

Massive **O**nline **A**nalyais (MOA) [21] is used as a comprehensive workbench to execute the experiments for evaluating the developed algorithms in this research. As a streaming environment has settings/constraints which do not exist in a traditional data mining environment. MOA can help to deliver standardised mechanisms for empirical evaluation of these methods. In a streaming environment, there are three primary dimensions:

- Accuracy.
- Amount of space required.
- The time required to learning from training data examples and make predictions.

The above properties can be independent or related to others because a change in the time and space used by a learning algorithm can influence the accuracy/error rate. By holding more pre-computed metadata, such as look-up tables, a learning algorithm can execute faster at the cost of space. A learning algorithm can also run faster by processing less information, either by holding less information or stopping early, thus needing to process less data. Typically, the more time is available to a learning algorithm, the more likely it is has a better accuracy.

To summarize, the main criteria to assess the performance of a learning algorithm in a streaming environment are the following: accuracy, the computational cost in both space and time, adaptability, theoretical performance guarantee, and a minimal number of parameters.

6.2 The Organisation of the Evaluations

Two sets of experiment were conducted to evaluate the developed algorithms in the research, G-eRules and Hoeffding Rules. The logical view of how each algorithm is related to the raised research questions and it's key contributions are illustrated in **Figure 6.1**.

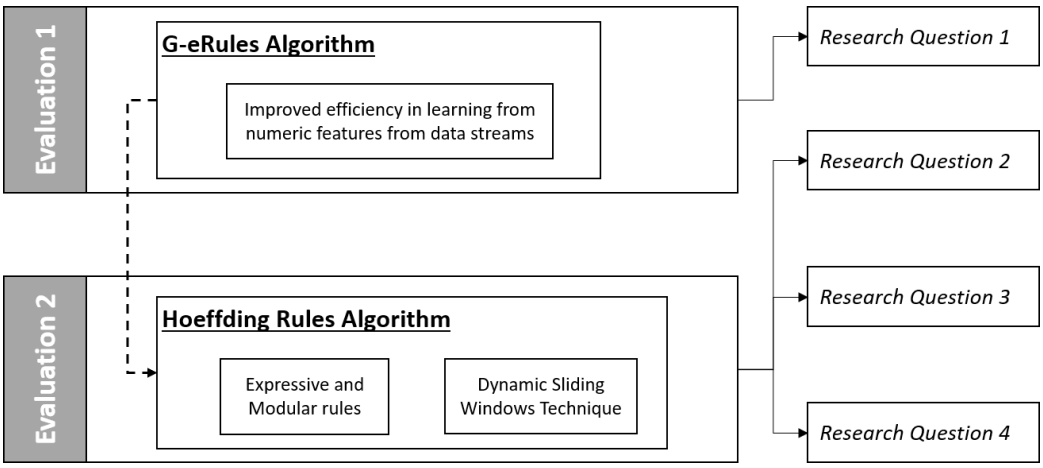


Figure 6.1: Logical structure of the evaluations in relation to the research questions.

G-eRules Algorithm

The new form of representation for a numeric rule term improves the computational cost while maintaining accuracy. The proposed method can be integrated with an existing algorithm which uses "Separate-and-Conquer" search for learning a classification rule to improve the computational cost/learning time which is one of three important metrics for a good classifier in a streaming environment. G-eRules algorithm was developed as an integration of the developed method into an existing rule-based algorithm, eRules. The in-

tegrated algorithm termed G-eRules, which should be more efficient (less learning time) compared with the original eRules algorithm and be competitive with other algorithms for classification tasks in streaming mining.

Dynamic Sliding Windows Technique

This technique addresses the current drawbacks of fixed sliding technique with the use of Hoeffding's Inequality. Hoeffding's Inequality was applied to determine a suitable window size for a given error rate to reflect an underlying concept from a sub-sample set (window size) of data examples. Rather than relying on a static size for the windows throughout the learning process, the window size is determined dynamically throughout the learning process Concept drifts have always been the challenges for classification tasks in a streaming environment.

Hoeffding Rules Algorithm

Hoeffding Rules algorithm is coined as a collective system of improvements for classification tasks in a streaming environment. There are three main dimensions to assess the performance of a learning algorithm (accuracy, learning time and space). Usually, in a streaming environment, the gain in one metric is at the cost of one or more metrics. G-eRules algorithm was developed to show the improvement in efficiency (learning time) while maintaining the competitiveness in accuracy, and do not affect space. The developed dynamic sliding windows technique is supposed to improve the dimensions of space and accuracy in a streaming environment. Altogether, these techniques are

incorporated to form the Hoeffding Rules Algorithm. The empirical evaluation of Hoeffding Rules algorithm demonstrates the mutually improvements in accuracy and learning time for classification tasks in a streaming environment while the requirement for space is maintained for Hoeffding Rules algorithm.

6.3 Critical Views and Discussion

This section provides critical views for the conducted experiments for the proposed technique in dealing with numeric features by evaluating the developed G-eRules algorithm plus a comprehensive view of performance for the developed Hoeffding Rules algorithm.

The main challenge in learning classification rules in a streaming environment is to maintain an accurate set of rules over time and the ability to deal with concept drifts. Hoeffding Rules algorithm has substantially addressed this challenge with dynamic sliding windows and the foundation from G-eRules algorithm.

6.3.1 Evaluation 1: G-eRules Algorithm

An extended algorithm based on eRules algorithm is termed G-eRules, which is based on the method as proposed in **Section 3.4** to create heuristic for numeric features in order to make the eRules algorithm computationally more efficient. This section shows empirical experimental results to confirm the proposed algorithm improves e-Rules' processing time and competes well with other well-known data stream classifiers, as stated in the background work in

Chapter 2. While the proposed method for numeric features was integrated with the eRules algorithm, it could also be adopted as the mechanism to deal with numeric features in other rule-based and tree-based streaming classifiers.

MOA was used as an underlying and independent base to evaluate the proposed algorithm. Particularly, the following algorithms were used for this comparative analysis, which also come with MOA natively.

1. *VFDR (Very Fast Decision Rules)*–[57] a rule-based data stream classifier.

2. *Hoeffding Trees*–[40] is a state-of-art decision tree-based classifier.

3. *eRules* was outlined in **Section 2.4.3**. It is the only data stream classifier that is able to abstain from classifying when uncertain.

4. *G-eRules* inherited from original eRules but this version of the classifier uses the proposed numeric rule term format and induction approach, as proposed in **Section 3.5**.

The reason for these choices is that the Hoeffding Trees algorithm has been studied by the community [70] for the last decade as the state-of-the-art in data stream classification and VFDR is one of a few rule-based classifiers which can generate rules directly from a data stream, and thus shares similarities with eRules. All experiments in this section used the default settings for eRules and G-eRules, which are: a sliding window of size 500, and a particular rule being removed if it falls below an individual accuracy 0.8 and has had a minimum number of 5 classification attempts.

6.3.1.1 Datasets and Data Settings

Artificial (stream generators in MOA) and real datasets are both used in the experiments:

SEA Generator–This artificial dataset is introduced in [116]. It generates an artificial data stream with 2 classes and 3 numeric features, whereas one feature is irrelevant for distinguishing between the 2 classes. In other words, one numeric feature does not have any influence on the class label for give data example and it should be ignored by learning algorithm. This data generator has been used in empirical studies in [20, 57, 115]. For all experiments, the default generator settings are used, which are concept function 1, a random instance seed of 1, allowing unbalanced classes and a noise level of 10 %. There are 500,000 data examples generated in the experiments.

Random Tree Generator–This generator is introduced in [40] and is based on a randomly generated decision tree in which the user can specify the number of features and classes. New instances are generated by assigning uniformly distributed random values to features and the class label is determined using the tree. Because of the underlying tree structure, decision tree-based classifiers should perform better on this stream. There are two versions of this data stream, one with 5 categorical and 5 numeric features, and 3 classifications called RT Generator 5-5-3; and the other one with 3 classifications but no categorical and only 4 numeric features called RT Generator 0-4-3. Both versions comprised 500,000 instances. The remaining settings were left at their default values, which are a tree random seed of 1, instance random seed option of 1, 5 possible values for each categorical feature, a maximum

tree depth of 5; minimum tree level for leaf nodes of 3, and the fraction of leaves per level from first leaf level onwards being 0.15.

Covertype–is a dataset from US Forest Service (USFS) Region 2 Resource Information System(RIS) which contains the forest cover type for 30x30 meter cells. There are 581,012 instances with 54 features, 10 of which are numeric features. This dataset has been used in several papers on data stream classification, i.e. in [3, 59, 98]. This real dataset was downloaded from the MOA website [1] and used without any modifications.

Airlines–this data source was created based on the regression dataset from Elena Ikonomovska, which was extracted from Data Expo 2009, which consists of about 500,000 flight records. The task is to use the information of scheduled departure to predict whether a given flight will be delayed. The dataset comprises 3 numeric and 4 categorical features. The dataset was also used in one of Elena's studies about data streams [73]. This real dataset was downloaded from the MOA website [1] and used without any modifications.

Instead of initialising an artificial stream for each experiment anew, synthetic generators only created artificial datasets once into static files, then these datasets were subsequently streamed in as data sources in all experiments. This way, each classifier would be presented with exactly the data examples in an identical order within the stream.

6.3.1.2 Overall Accuracy, Tentative Accuracy and Learning Time

The experiments in this section focus on the scalability of the proposed G-eRules algorithm to fast data streams but also, show its competitiveness in terms of classification accuracy and learning time. While traditional learning

problems are typically evaluated with well-known methods such as holdout and cross-validation, both of these methods were not designed for streaming settings, nor can they be directly applied to evaluate the continuously arrival of new data examples. As discussed in **Section 2.5**, there are a few different approaches that have been used in literature, such as 'Holdout' and 'Prequential', but none of these are widely and universally used, nor accepted within the data stream mining community or by the literature.

Nevertheless, 'Interleaved Test-then-Train' strategy [21], which was created specifically for the streaming domain, and it was used in [72, 57, 115]. The experiments in this section also used an 'Interleaved Test-then-Train' approach to calculate the mean accuracy and learning time from all candidate classifiers.

One special feature of the eRules algorithm is the ability to abstain from a classification if there is no realised rule that covers the unseen data example, which G-eRules also inherited. For this reason, tentative accuracy is also recorded for eRules and G-eRules, which is the accuracy for instances where the classifier is confident and actually makes a prediction.

Table 6.1: G-eRules compared with eRules, Hoeffding Trees and VFDR. The abbreviation *Ab* stands for abstain rate, *T.Ac* stands for tentative accuracy, *T* stands for execution time and *Ac* stands for accuracy.

Dataset	**Classifier**									
	eRules			**G-eRules**			**Hoeffding Trees**		**VFDR**	
	Ab (%)	**T.Ac (%)**	**T (ms)**	**Ab (%)**	**T.Ac (%)**	**T (ms)**	**Ac (%)**	**T (ms)**	**Ac (%)**	**T (ms)**
Covertype	27.993	81.12	90,098	35.77	80.68	12,484	80.61	10,631	61.29	20,722
Airlines	9.63	61.76	366,859	16.68	62.55	116,975	65.20	4,289	61.02	4,818
RT Generator 5-5-3	10.52	69.28	333,163	26.47	64.77	36,486	89.78	2,422	55.51	79,153
RT Generator 0-4-3	96.35	34.94	47,790	65.65	81.21	4,283	96.39	2,391	90.10	3,883,291
SEA Generator	94.06	68.58	22,043	40.84	95.87	3,227	99.9.3	1,265	62.70	176,705

As shown in **Table 6.1**, G-eRules achieved a very similar classification accuracy when compared with the original eRules algorithm, however, it was significantly faster. Both eRules and G-eRules suffered from higher abstaining rates when confronted with with data stream with many numeric features(i.e., Random Tree Generator datasets and SEA). However, G-eRules had a lower abstaining rate on numeric data. Nevertheless, G-eRules not only had a lower abstain rate when confronted with numeric features, but also a high accuracy.

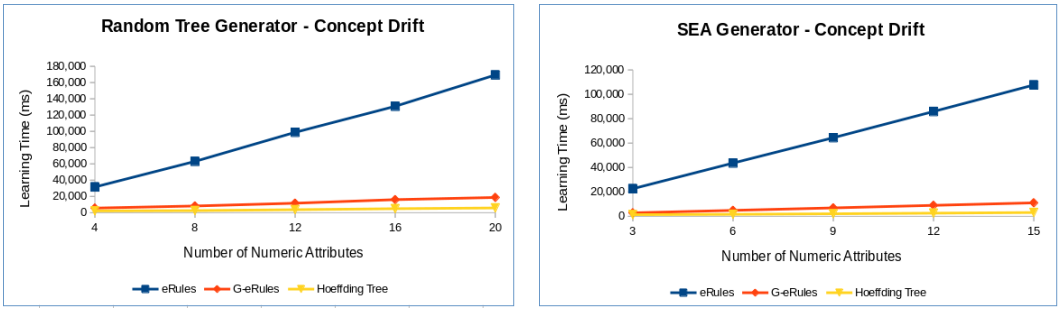
G-eRules generally outperformed existing rule-based classifier, VFDR, in all experiments, in terms of accuracy and running time. For accuracy, G-eRules outperformed VFDR in 3 out of 4 cases, and in general, processed the data streams much faster. For 'RT-Generator 0-4-3' synthetic generator, the VFDR algorithm was unexpectedly slow. This could be explained by the VFDR algorithm not reaching a stable rule set that keeps changing over

time. Regarding the Airlines dataset, eRules and G-eRules needed much longer to process the stream, however, again, this could be due to the fact that both algorithms are unstable on this particular dataset due to a fixed sliding window size. However, this problem could be potentially addressed by a sliding window that adapts its size dynamically to concept changes.

In comparison with the Hoeffding Trees algorithm, G-eRules achieves a comparable accuracy on the two real datasets, Covertype and Airlines. On the synthetic generators, Hoeffding Trees performs better. However, one needs to note that the Hoeffding Trees algorithm is less expressive compared with G-eRules, eRules, and other rule-based classifiers. The rule set generated by algorithms such as eRules or G-eRules can easily be interpreted and examined by the users. On the other hand, the Hoeffding Trees algorithm needs a further processing step to explain the single path that led to a particular decision, which may well be quite long for interpretation by decision takers. As such, the G-eRules algorithm is an expressive rule-based technique for data stream classification.

In order to examine G-eRules' computational advantage within numeric features, there is an experiment comparing eRules and G-eRules on the same base datasets as in **Table 6.1**, but with increasing numbers of numeric features. SEA and Random Tree generators (with numeric features only) were used, both with gradual concept drift lasting for 1000 data examples starting at the $250,000^{th}$ data example in order to trigger adaptation and thus make the problem computationally more challenging. The two real datasets, Coverytpe and Airlines, were also chosen. For all datasets, the numeric features were gradually increased by duplicating the existing features. The reason for

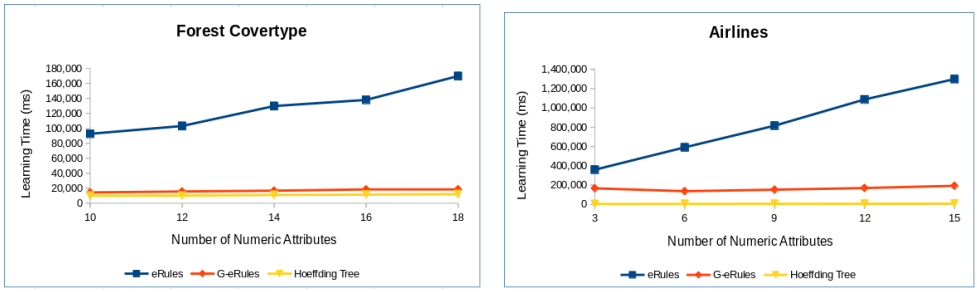
duplication is that the concept encoded in the stream stays the same, but the computational effort needed to find the rules is increases.



(a) Random Tree Generator **(b)** SEA Generator

Figure 6.2: Learning time of Random Tree and SEA generators with concept drift.

As expected, **Figures 6.2a**, **6.2b**, **6.3a** and **6.3b** show that the execution times for G-eRules increased at a much smaller rate compared with the execution times of eRules. In fact, G-eRules increased in execution time when it was confronted with an increasing number of numeric features, and the increasing rate seemed to be very close to that of Hoeffding Trees. The reason for this is that the eRules algorithm has to calculate $p(\omega_i|\alpha < v)$ and $p(\omega_i|\alpha \geq v)$ for each value of a numeric feature, then this process is repeated during the learning.



(a) Forest Covertype **(b)** Airlines

Figure 6.3: Learning time of real datasets, Covertype and Airlines.

On the other hand, for each numeric feature, G-eRules only has to calculate Gaussian distribution for each class label once and then the classifier can update and look up $p(\omega_i|x < \alpha \leq y)$ value from the realised distributions. Note that the execution times for VFDR were omitted in **Figures 6.2a**, **6.2b**, **6.3a** and **6.3b**, as they were generally much greater compared with eRules and G-eRules on these particular cases, as shown in **Table 6.1**.

Moreover, eRules tends to produce rules with irrelevant terms for numeric features, resulting in some of the rules produced by eRules being too specific. In other words, the eRules algorithm encounters the problem of overfitting in dealing datasets with many numeric features. However, rule terms in the form of $(x < \alpha \leq y)$ produced from G-eRules for numeric features tend to cover more data examples matching the target class. Thus, it needs to produce less rule terms. Although G-eRules may not dramatically improve the levels of accuracy from the original eRules algorithm, the processing time is clearly improved, hence the proposed technique in dealing numeric feature is shown to be a viable approach to learning from datasets with numeric features.

6.3.1.3 Experimental Conclusion and Remarks

The conducted experiments of G-eRules algorithm showed that in terms of overall classification accuracy and adaptivity to concept drifts, G-eRules algorithm exhibits a similar accuracy to other competitors or outperforms them. Moreover, the evaluation also showed that G-eRules outperforms all other competing rule-based data stream classifiers.

Although the efficiency in learning from data streams has been improved in G-eRules algorithm, the algorithm inherently uses a static sliding window technique to buffer unseen data examples from the original eRules algorithm. It has been observed that both eRules and G-eRules were able to detect and adapt to concepts drifts, but the sensitivity and latency in detecting and recovering vary. In data stream mining, concept drift can appear gradually and abruptly, and there could also be a transitional period where data examples from more than one concepts arrive together. Therefore, static buffering windows throughout the learning process cannot be guaranteed to adapt to concept drift with minimal delay. Practically, data streams in real-time are not static, and the ability to adapt to changes and seasonality is essential for any data mining solutions.

6.3.2 Evaluation 2: Hoeffding Rules Algorithm

An empirical evaluation has been conducted to evaluate Hoeffding Rules in terms of accuracy, adaptivity to concept drift and the trade-off accuracy for a (white-box) model such as Hoeffding Rules algorithm compared with (black-box) or complex rules to interpret a model such as the Hoeffding Trees

algorithm.

Two different classifiers were used for comparing and analysing the Hoeffding Rules approach.

- **VFDR** (Very Fast Decision Rules) [57], is rule-based algorithm which learns classification from data streams.
- **Hoeffding Trees** [40], the state-of-the-art in decision tree learning algorithm from data streams.

The rationale behind this choice is twofold: (1) Hoeffding Trees has established itself for the last two decades as the state-of-the-art in data stream classification [70, 82, 100, 56], with a long history of success; and (2) both techniques use the Hoeffding Bound for approximation in learning, which has also been adopted in the proposed technique for dynamic sliding window to learn from a data stream.

Table 6.2 shows the default parameters for the classifiers that were used in all the experiments, unless stated otherwise.

Table 6.2: Parameter settings for classifiers used in the experiments.

Hoeffding Rules	**VFDR (Very Fast Decision Rules)**	**Hoeffding Trees**
HR.SW: 200 HR.MRT: 3 HR.RVT: 0.7 HR.HBT: 0.01 HR.APT: 0.5	VF.P: 0.1 VF.SC: 0.0 VF.TT: 0.05 VF.AAP: 0.99 VF.PT: 0.1 VF.AT: 15 VF.GP: 200 VF.PF: First hit VF.OR: false VF.AD: false	HT.NUE: Gaussian Observer HT.NOE: Nominal Observer HT.GP: 200 HT.SC: Information Gain HT.SCON: 0.0 HT.TTH: 0.05 HT.BS: false HT.RPA: false HT.NPP: false HT.LP: NBAdaptive
HR.SW: Sliding window size HR.MRT: Minimum rule tries HR.RVT: Rule validation threshold HR.HBT: Hoeffding Bound threshold HR.APT: Adaptation threshold	VF.P: % of total samples seen in the node VF.SC: Split Confidence VF.TT: Tie threshold VF.AAP: Anomaly probability threshold VF.PT: Probability threshold VF.AT: Anomaly threshold VF.GP: Grace period VF.PF: Prediction function VF.OR: Ordered rules VF.AD: Anomaly detection	HT.NUE: Numeric feature estimator HT.NOE: Categorical feature estimator HT.GP: Grace period HT.SC: Split criterion HT.SCON: Split confidence HT.TTH: Tie threshold HT.BS: Binary split HT.RPA: Remove poor feature HT.NPP: No pre-prune HT.LP:Leaf predictieve

6.3.2.1 Datasets and Data Settings

Different synthetic and real-world datasets were used in the experiments. As for synthetic datasets, available stream generators in MOA were used. The real world datasets 'Airlines' and 'Forest Covertype' are known and used for batch learning, in which case all data examples from datasets are read and learnt in one pass. However, these datasets were simulated into data streams by reading data examples from these datasets in ordered sequence over time.

Table 6.3: Setting parameters for synthetic stream generators are used in the experiments.

Random Tree	**Random Tree with Drift**		**SEA**	**SEA with Drift**		**STAGGERS**	**STAGGER with Drift**	
	Before Drift	**After Drift**		**Before Drift**	**After Drift**		**Before Drift**	**After Drift**
RT.TRSV: 1 RT.ISV: 1 RT.NCL: 4 RT.NCA: 5 RT.NNA: 5 RT.NVPCA: 5 RT.MTD: 5 RT.FLL: 3 RT.LF: 15%	RT.TRSV: 1 RT.ISV: 1 RT.NCL: 4 RT.NCA: 5 RT.NNA: 5 RT.NVPCA: 5 RT.MTD: 5 RT.FLL: 3 RT.LF: 15%	RT.TRSV: 5 RT.ISV: 5 RT.NCL: 4 RT.NCA: 5 RT.NNA: 5 RT.NVPCA: 5 RT.MTD: 5 RT.FLL: 3 RT.LF: 15%	SEA.F: 1 SEA.IRS: 1 SEA.BC: true SEA.NP: 10%	SEA.F: 1 SEA.IRS: 1 SEA.BC: true SEA.NP: 10%	SEA.F: 2 SEA.IRS: 1 SEA.BC: true SEA.NP: 10%	ST.IRS: 1 ST.F: 1 ST.BC: true	ST.IRS: 1 ST.F: 1 ST.BC: true	ST.IRS: 1 ST.F: 2 ST.BC: true
	Drift at: 150,000 Drift Width: 10,000			Drift at: 150,000 Drift Width: 10,000			Drift at: 150,000 Drift Width: 10,000	
RT.TRSV: Tree random seed value RT.ISV: Instance seed value RT.NCL: Number of class labels RT.NCA: Number of categorial feature(s) RT.NNA: Number of numerical feature(s) RT.NVPCA: Number of values per categorical feature RT.MTD: Max tree depth RT.FLL: First leaf level RT.LF: Leaf fraction			SEA.F: Classification function as defined in paper SEA.IRS: Seed for random generation of instances SEA.BC: Balanced class SEA.NP: Noise Percentage			ST.IRS: Instance random seed ST.F: Classification function ST.BC: Balanced class		
∗ 400,000 data examples are generated for each experiment. ∗ In each experiment, all classifiers are given identical data examples and same sequenced order.								

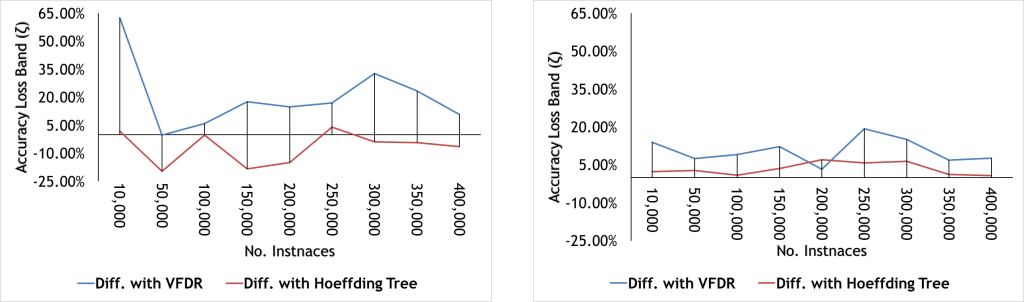
All synthetic data stream generators are controllable by parameters and **Table 6.3** shows the settings used for all synthetic streams in the evaluation.

6.3.2.2 Utility of Expressiveness

The empirical evaluation is focused on the cost of expressiveness when comparing the accuracy and performance between classifiers for data streams.

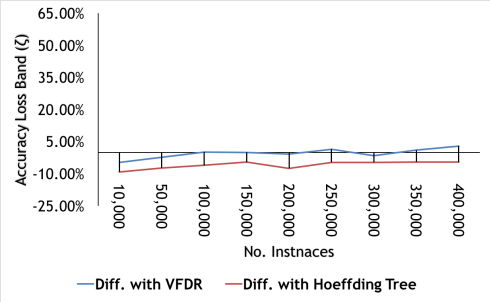
As mentioned in [127], a learnt model from the labelled data examples may produce a high predictive accuracy for unlabelled data, however, the learnt model can be hard and complex to understand for laypersons or even domain experts. To examine the trade-off between rule-based classifiers such as VFDR and Hoeffding Rules with a decision tree-based algorithm such as the Hoeffding Trees algorithm, all classifiers were evaluated with the same base datasets. Concept drift was also simulated in all synthetic datasets from 150,000 data examples onwards for approximately 1,000 further instances, where both concepts were presented before switching completely to the new

concept. The accuracy loss band ζ can be either positive or negative, in which positive values indicate the Hoeffding Rules algorithm has a better accuracy and negative values indicate the shortfall in accuracy compared with its competitor.

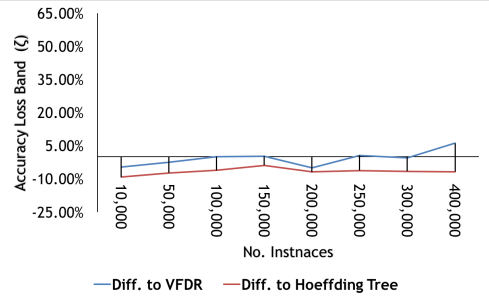


(a) Forest Covertype - Real Dataset **(b)** Airlines - Real Dataset

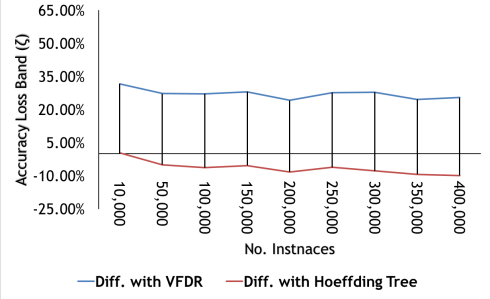
Figure 6.4: Difference in accuracy compared with other classifiers for real data streams.



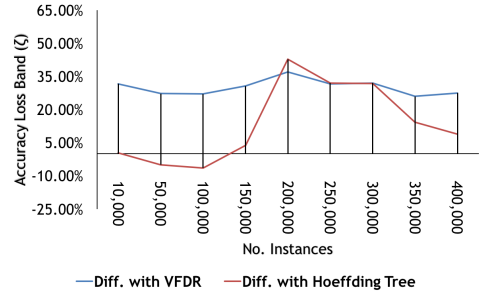
(a) SEA Dataset with No Concept Drift



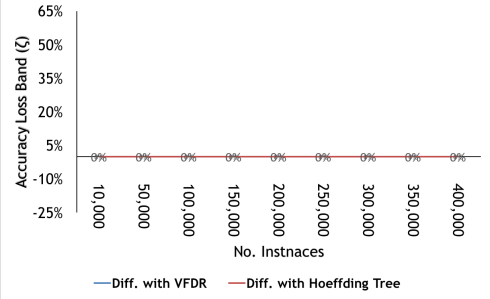
(b) SEA Dataset with Concept Drift at 150,000



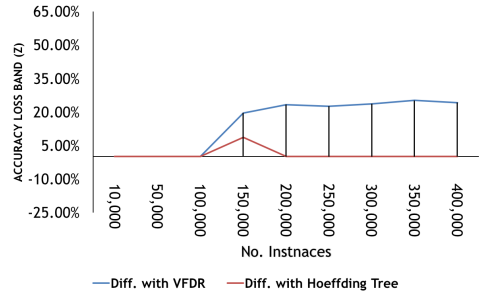
(c) Random Tree Dataset with no Concept Drift



(d) Random Tree Dataset with Concept Drift at 150,000



(e) STAGGER without Concept Drift



(f) STAGGER with concept drift at 150,000

Figure 6.5: Difference in accuracy compared with other classifiers for synthetic data streams.

Figures 6.5a, **6.5b**, **6.5c**, **6.5d**, **6.5e**, **6.5f**, **6.4a** and **6.4b** show that

accuracy loss band ζ of Hoeffding Rules is very competitive compared with the Hoeffding Trees, while clearly outperforming VFDR in most cases. The reader should note that the existing implementations of the Hoeffding Trees, VFDR and synthetic data generators in MOA were used. These MOA-implemented classifiers may have been optimised to work well on these synthetic datasets. Two real datasets Airlines and Covertype, have been chosen and included for an unbiased evaluation. VFDR is the closest algorithm to Hoeffding Rules because it is a native rule-based classifier with the ability to produce rules directly from the seen labelled data examples. However, VFDR does not offer abstaining and forces a classification. Evidently, it has a positive loss band compared with VDFR on both real and synthetic datasets and outperforms the Hoeffding Trees in the Airlines dataset, while suffers a minor negative loss band on a few occasions on the Covertype dataset.

6.3.2.3 Abstaining from Classification

The result in **Section 6.3.2.2** shows the tolerance of rule-based classifiers in term of the utility of expressiveness compared to decision tree-based classifiers. In addition, another important feature of Hoeffding Rules is the ability to abstain. In **Figure 6.6**, can be seen that the abstaining rate of Hoeffding Rules decreases as it processes more data examples, except with the Random Tree dataset. For synthetic datasets with concept drift, at the point of simulated concept drift (150,000), the abstaining rate spiked up but quickly recovered to adapt to the new concept. This indicates one of the major benefits of abstaining instances from classification; when Hoeffding Rules algorithm is uncertain about a classification, it does not risk classifying unseen

data examples. This feature is desirable and indeed crucial in domains and applications, where miss-classification is costly or irreversible.

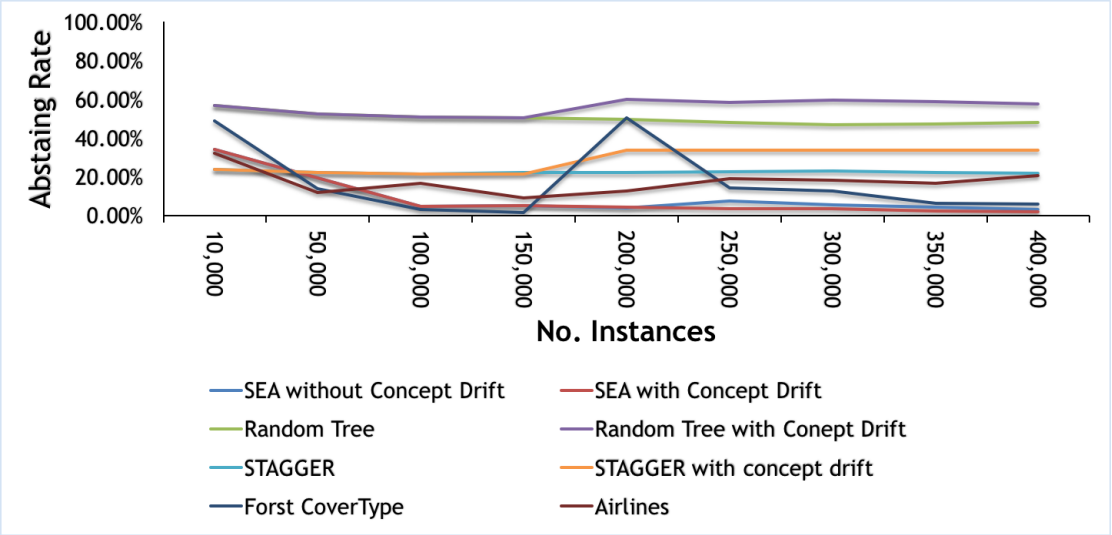


Figure 6.6: Abstaining rates of Hoeffding Rules.

Table 6.4 shows the accuracy evaluation between the Hoeffding Trees, VFDR and Hoeffding Rules. One unique feature of Hoeffding Rules is its ability to refuse a classification if the classifier is not confident of outputting a decisive prediction from its rule set. The tentative accuracy, abstaining rate for Hoeffding Rules and overall accuracy for the Hoeffding Trees and VFDR were recorded. The tentative accuracy for Hoeffding Rules is the accuracy for instances where the classifier is confident of producing a reliable prediction. The tentative accuracy was also used for estimating the utility of expressiveness shown in **Section 6.3.2.2**.

Table 6.4: Accuracy evaluation between the Hoeffding Trees, VFDR and Hoeffding Rules.

		Dataset							
Algorithm	**Measure**	**SEA**		**Random Tree**		**STAGGER**		**Covertype**	**Airlines**
		No Drift	With Drift	No Drift	With Drift	No Drift	With Drift		
Hoeffding Rules	Tentative Accruacy (%)	82.6	84.30	81.86	75.62	100	98.50	74.24	66.74
	Abstaining Rate (%)	8.07	6.43	49.51	56.02	22.27	28.52	10.04	16.47
VFDR	Overall Accruacy (%)	81.3	82.26	52.28	47.07	100	86.20	61.32	62.50
Hoeffding Trees	Overall Accuracy (%)	88.08	89.23	88.98	61.08	100	99.75	82.04	66.04

As shown in **Table 6.4**, Hoeffding Rules outperforms VFDR in all cases and is very competitive compared with the Hoeffding Trees, both on synthetic and real data streams. Hoeffding Rules is also competitive with the Hoeffding Trees; the accuracy of both classifiers is very close, with Hoeffding Rules outperforming the Hoeffding Trees in 3 cases. However, compared with the Hoeffding Trees, Hoeffding Rules produces a more expressive rule set and does not fundamentally suffer from the replicated subtree problem. Also, the classification rules generated by Hoeffding Rules can be easily interpreted and examined by everyday users or domain experts. Decision trees would first need to undergo a further processing step before a human user could interpret the rules. This would translate the tree into rules, starting with single passes from the root node down to each leaf. This may well be too cumbersome and too time-consuming a task for the decision maker.

Automated tree traversal to increase expansiveness is an additional linear process with respect to the number of tree nodes, or in its best case for balanced trees, it can be $O(log(t_n))$ where t_n is the number of nodes in the

tree [62].

6.3.2.4 Computational Efficiency

To examine Hoeffding Rules' computational efficiency, Hoeffding Rules, VFDR and Hoeffding Trees are compared on the same data streams as in **Table 6.4**. As shown in **Figure 6.7**, the execution time of Hoeffding Rules outperforms that of VFDR by far and is also very close to that of the Hoeffding Trees.

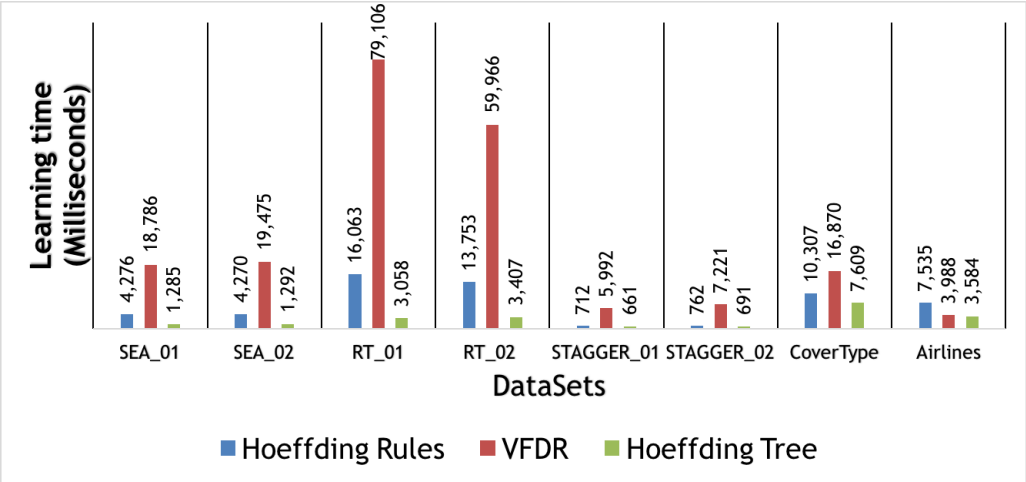


Figure 6.7: Benchmark of learning time for different datasets.

Generally speaking, Hoeffding Rules algorithm shows a much better performance in terms of utility of expressiveness compared with its direct competitor VFDR algorithm and is also competitive compared with its less expressive competitor the Hoeffding Trees algorithm. The same is true for the evaluation with respect to computational efficiency, Hoeffding Rules' runtime is much shorter than VFDR and only slightly longer than the Hoeffding Trees.

6.3.2.5 Experimental Conclusion and Remarks

The conducted experiments of Hoeffding Rules algorithm showed the utility of expressiveness when computing the accuracy and performance between classifiers for the data stream. In certain datasets, tree-based algorithms may marginally produce higher predictive accuracy for unlabelled data examples compared with rule-based algorithms such as Hoeffding Rules algorithm. However, the trained model/tree-structure can be difficult and complicated to understand for humans due to the way the tree is constructed, and from training examples.

Within the Hoeffding Rules Algorithm, the Hoeffding Bound is a metric that calculates an upper bound of difference between the best and second-best rule term within an iteration, to verify the true likeness of a rule term. If the rules satisfy the Hoeffding Bound from a number of buffered data examples, then these rules are more likely to be as effective as the rules that learn in an infinite data stream. There is no 'magic number' that can be predetermined to guarantee correctly capturing the underlying concept from a streaming data source. Also, the uniformity of the data sequence tends to vary between scenarios. Rather than relying on a static size of buffering data examples, an adequate number of unseen data examples that is required for inducing classification rules should be dynamically determined throughout the learning process. During the learning process, at any given moment, if the Hoeffding Bound is not satisfied, then the algorithm would need to buffer more data examples before a rule can be induced. Limitless buffering data examples is not a practical approach in a streaming environment due

to constraints in computing cost and physical storage.

Lastly, in addition to the ability to provide more clarity of learned models in the form of expressive modular rules, Hoeffding Rules algorithm also has the ability to abstain from a classification decision when the algorithm is uncertain about a prediction. In many scenarios, abstaining from classification and leaving the decision to the users may be necessary for critical applications where a misclassification would be costly, such as in medical or financial domains.

6.4 Identified Limitations of Hoeffding Rules Algorithm

Hoeffding Rules algorithm has substantially addressed the challenge in dealing with concept drifts with dynamic sliding windows. However, the algorithm is still far from perfect. Although, the capability in adapting to concept drifts Hoeffding Rules algorithm was shown, but a suitable stopping criterion was revealed as on one of the aspects which one can be improved to increase to performance of the algorithm.

Due to the nature of a covering algorithm, the rule inducing process typically stops when there are no more negative examples (or not many of them). However, this is also the main cause of overfitting; where a rule specifically covers a small subset of data examples, and will not perform well with unseen data examples. Therefore, the main issue that needs to be solved for achieving completeness in covering the underlying concept is to

ensure that a learned rule is diverse enough, so that it generally represents a complete piece of information.

6.5 Summary

This chapter provides a comprehensive evaluation for the two developed algorithms in the research, G-eRules and Hoeffding Rules. There are three primary dimensions of interest–the time required to learning, the amount of space and the error of predictions. These criteria were used to validate the developed algorithms with other existing algorithms with different combinations of selected metrics to conclude on a particular dimension of interest such as speed, accuracy or adaptability.

The experiments for G-eRules algorithm confirms the effectiveness of the proposed technique based on Gaussian distribution for learning heuristic from numeric features. The technique can be practically integrated/used in an existing framework and G-eRules is an example of such an integration.

The developed Hoeffding Rules algorithm represents all the findings of the research in a collective system. It addresses the challenges in dealing with numeric features and dynamically buffering data examples in a streaming environment. Evidently, it is not always the best in terms of accuracy and learning time compared with other algorithms. In a streaming environment, adjusting the time and space used by the algorithm can influence the error rate. Hoeffding Rules algorithm is very competitive with state-of-the-art, Hoeffding Trees algorithm as well as outperforms in several cases, especially with real-life datasets. Additionally. Hoeffding Rules algorithm can produce

modular express rules directly from the data examples, and it is more robust with concept drifts, which is not the case with the Hoeffding Trees algorithm.

Lastly, due to the nature of a covering algorithm, there is a number of limitations was identified in Hoeffding Rules and these limitations open the way forward for potential improvements for Hoeffding Rules algorithm in the future.

Chapter 7

Conclusion and Future Work

This research focuses on approaches and algorithms to mining valuable and expressive insights from data streams. The advances in network communication and rapid increase in the number of connected devices lead the needs for a system and dedicated approaches to learn from data streams which are unbounded, as well as having the ability to fit all the learning data into secondary storage.

Chapter 3 introduces G-eRules algorithm which is developed to improve the computing efficiency in dealing with numeric features for classification tasks from dynamic data streams. The dynamic sliding windows technique in **Chapter 4** is developed to address the challenges in dealing with concept drifts in a streaming environment. In **Chapter 5**, Hoeffding Rules, a rule-based algorithm for inducing highly expressive and modular classification rules from data streams, was developed. Finally, the empirical evaluations of the developed algorithm are described and critically discussed in **Chapter 6**.

In this chapter, the main research proposition and research questions are

revisited in **Section 7.1**. **Section 7.2** highlights the novelty contribution of the research. This chapter is concluded with suggestions for future direction in **Section 7.3**.

7.1 Research Questions Revisited

The primary proposition of the research was:

> "*This research investigates common issues across most rule-based techniques in data stream mining and introduces and discusses a set of techniques as a system to improve the efficiency and feasibility of learning from streaming data sources.*"

This research is located in the *Knowledge Discovery* domain and classification tasks in a streaming environment specifically. To address the above proposition, the following four questions were identified.

Research Question 1*: How can a trained predictive model from a streaming data source be reliably interpreted and understood by the users?*

While this research aims to establish a technique to reliably and accurately learn from streaming data sources, tree-based models may offer promising accuracy, but suffer from a lack of expressiveness. As accuracy has been the dominating measurement of interest in comparing classifiers in both static and streaming environments, it is evident that real-time decision making based streaming models still suffer from the issue of trust.

By looking to develop a rule-based technique, a user can determine an accuracy loss, such that a model can be expressive enough to grant trust, and at the same time, the shortfall in accuracy can be tolerated compared with any other best performing classifiers which are less expressive or can even be completely black box.

***Research Question 2**: What are the most efficient approaches to improve the efficiency in learning heuristic for numeric features from a high-volume and high-velocity streaming data source?*

To date, there is no single optimised rule-based method to deal with numeric features. However, methods for working with numeric features have been studied extensively in the past two decades for static/batch data environments.

This research investigates the ability to learn from numeric features in a streaming environment. In particular, the computational efficiency is investigated by comparing established methods with the proposed works because the runtime when processing numeric features in the streaming environment is even more of a concern.

***Research Question 3**: What impact have the properties of streaming data sources had on the faith and accuracy of the predictions for classification tasks?*

Abstaining is desirable and necessary in critical applications where misclassification is very costly, such as in medical or financial applications. Tree-based techniques always return a prediction for a given data example because of its hierarchical structure. However, a model that can draw a clear line be-

tween objective and random subjective prediction is highly desirable.

Research Question 4*: How have concept drifts and dynamic behaviours in streaming data been posing challenges in learning a reliable and accurate classification model?*

The key challenge of streaming data classification lies in the need of the classifier to adapt in real-time to concept drifts, which is significantly more challenging if the data stream is high velocity.

By looking at classification in data stream environments, this research does not stop at the assumption of early machine learning techniques which typically expect to work with a complete set of training data examples, but even more demanding and infinite data sources. An example of such an application is the monitoring of a network of sensors to determine whether the unbounded and sequential flow of data may be too large to consider storing and revisiting as being impractical.

7.2 Contributions to Knowledge

This research presents various topics that are related to the construction of a rule-based algorithm for streaming data classification. In other words, the proposed rule-based algorithm addresses the raised concerns in the research questions as described in **Section 1.2**, keeping in view the drawbacks of decision tree-based techniques as a basis for a classification model in both static and streaming environments.

By testing the raised research questions, this work introduces novel methods and an algorithm in dealing with the challenges mentioned in the pre-

vious section for learning expressive classification rules from streaming data sources. The critical contributions are summarized in the following points:

- Develop a new method in learning heuristics for numeric features from streaming sources by using Gaussian distribution, which is proven to be more efficient compared with existing method from other rule-based classification algorithms for streaming data.

- Address the challenges in buffering recent data examples by developing the dynamic sliding window technique, which is based on the well-known Hoeffding's Inequality. The empirical illustration showed that the proposed sliding window method performs better than the previously used method.

- A further and the most significant contribution of this research is that Hoeffding Rules algorithm was formed to address challenges and identified shortfalls from other methods in learning classification model from streaming data sources as set out in *Research Question 2, 3 and 4*. Especially, Hoeffding Rules is designed to introduce to modular, expressive and interpretable classification rules directly by employing the "Separate-and-Conquer" searching approach which is the primary concern in *Research Question 1*.

  The empirical evaluations showed the Hoeffding Rules algorithm is able to adapt to concept drifts and is significantly faster when compared with other rule-based algorithms for data streams, so that these are the goals. In addition, Hoeffding Rules algorithm is very competitive

or even outperformed in certain cases compared with the state-of-the-art tree-based algorithm, Hoeffding Trees. Collectively, the Hoeffding Rules algorithm delivers the primary proposition of the research:

> "*Develop a set of techniques as a system to improve the efficiency and feasibility of predictive learning from streaming data sources.*"

Finally, there is software code for the listed contribution above, which were developed to be used in the common framework, Massive Online Analysis (MOA), for empirical evaluations. These components can be integrated into other research projects.

7.3 Future Directions

This thesis has created up a number of avenues for future work. Possible future work on Hoeffding Rules would comprise the development of more advanced stopping criteria or filtering criteria to improve accuracy for the core learner in Hoeffding Rules, this is Prism algorithm. Additionally, the Hoeffding Rules algorithm can be extended further to learn more generalised rules instead of only classification predictive rules for one target class feature.

Stopping Criteria for Rule Induction

Pre-pruning is one of the approaches to controlling the complexity of the resulting target theory and avoiding overfitting during the rule construction. Stopping criteria is one pruning method that automatically decides when to

stop inducing a rule or adding an extra rule term to a rule. In other words, a stopping criteria in rule induction stops the refinement of a rule, even though it may not cover all training examples positively. In many cases, a rule should not have more refinements because excluding the remaining covered negative data examples is estimated to be too costly. There are several prominent stopping criteria which can be considered, such as minimum coverage constraints, support and confidence, significant tests, encoding length restrictions and correlation cut-offs.

The simplest form of stopping criteria is discarding rules with low coverage. For instance, a rule should cover a certain minimum number of data examples or a minimum number of positive examples. For the case of minimum total coverage, if a rule only covers data examples in the shaded area, then it should be discarded from consideration. Examples need to be covered by the rule, regardless of whether they are positive or negative. However, in both cases, a deeper look into a stopping criterion-based coverage is needed because the objective is to avoid overfitting by filtering out rules whose quality cannot be reliably estimated due to the small number of training data examples they cover.

Similar to minimum coverage criteria, some rule-based algorithms also use a minimum precision or minimum rule accuracy constraint. Although minimum accuracy is already used in the Hoeffding Rules algorithm, metrics such as recall and precision are considered to maximise the generalisation. For example, FOIL [103] has a theoretical stopping criteria; when the best rule has a value of purity below a certain threshold such as 80%, the algorithm rejects the rule and the learned theory is considered complete. Moreover, in SFOIL

algorithm [101] used minimum coverage constraints based on minimum precision and minimum accuracy as a termination criterion for the stochastic search.

There is also no reason why only a single metric should be used as a stopping criterion to filter out unpromising rules. Association rule mining algorithms are a notable example of combining multiple criteria with the use of support and confidence. This is also extended to classification algorithms, where the algorithms obtain candidate rules for covering loop in association rules learning framework [78, 89]. Confidence can indicate the quality of the rules, where support aims to ensure a minimum certainty by discarding the rules with high confidence but with too few positive examples covered.

Filtering Criteria with Likelihood Ratio Significant Test

Although rules are already evaluated with learning data examples, the performance of the induced rules can also be estimated. In particular, when induced rules only cover a few data examples, their evaluation metrics may not represent the entire underlying concept. The CN2 [27, 28] algorithm makes use of significance testing, the *likelihood ratio statistic* [28], which compares the probability of class distribution covered by the rule, with the prior distribution of data examples in the training dataset. A robust rule would have a significant difference in these two distributions. The use of significance testing was used as early as 1989 within CN2, and later, in works such as the relational learner FOIL [110] and BEXA [118]. Unlike stopping criteria, the shape of the likelihood isometric does not depend on the number

of data examples in the learning set but the actual class distribution. Therefore, a rule may not be significant for a small dataset, but it may become more significant if there are a lot of training data examples. Hence, the significant test tends to lead to purer rules. However, some of these pure rules are often the outcome of overfitting, and this raise doubt as to whether this strategy can effectively counter overfitting.

Initially, stopping criteria and filtering appear to very similar. Nonetheless, the filtering out strategy and stopping criteria can distinguished as follows:

- Stopping criteria determine when the inducing process should stop and the current best rule returned.
- Filtering out determines regions of acceptance performance.

In any case, both strategies are closely related. In particular, filtering criteria can also be used as stopping criteria if necessary; where no further rule can be found within acceptance region filtering criteria, the learning process is considered to be complete. Equivalently, if no further refinement of a rule can be found within the acceptable region, then the rule is considered to be complete, and the rule's inducing process stops.

Generalised Rules Induction

A common approach to descriptive learning in data mining is frequent item-set mining, which expresses many-to-many relationships between the items from a given number of item-sets. Association Rules learning is one the

techniques based on the concept of '*Frequent Itemset Mining*' to discover actionable insights. Associations Rules have a general form '**IF** *antecedent* **THEN** *output*'. For instance, when there is a rule in the form of 'IF a **AND** b **THEN** c', the presence of a and b is confirmed, so the presence of c is high with a minimum probability, which is defined by the user in terms of support and confidence values of the rules. Further information about frequent itemsets mining as well some algorithms in this area are reviewed and discussed in [8, 22].

For a predictive technique such as classification, the induced rules always produce insights and information related to the target class feature. Contrary to predictive techniques, descriptive techniques can capture a greater relationship between all features to explain the correlated links, in order to produce insights and patterns. Descriptive methods are designed for unlabelled data examples, as is the case with Association Rules. However, descriptive rules can express a wider range of feature values (not only binary, as with Association Rules). In this context, if there is a special target feature which that is always in the *output* part of the rules, this is referred to as classification. On the other hand, generalised descriptive rules are in a more general context, where the *output* part of the rule can be the arbitrary conjunction of any 'feature-value' pairs.

References

[1] Datasets from moa (massive online analysis) website. [Online; accessed Apirl-2014].

[2] High-Frequency Firm Tradebot Swam in Barclays Pool, N.Y. Says.

[3] Hanady Abdulsalam, David B. Skillicorn, and Patrick Martin. Streaming Random Forests. In *Proceedings of the International Database Engineering and Applications Symposium, IDEAS*, pages 225–232, 2007.

[4] C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu. On demand classification of data streams. *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 503–508, 2004.

[5] Charu Aggarwal. *Data Streams: Models and Algorithms*. Springer Science & Business Media, 2007.

[6] Charu C. Aggarwal, T. J. Watson, Resch Ctr, Jiawei Han, Jianyong Wang, and Philip S. Yu. A Framework for Clustering Evolving Data Streams. *Proc. of the 29th int. conf. on Very large data bases*, pages 81–92, 2003.

[7] Charu C. Aggarwal and Philip S. Yu. On clustering massive text and categorical data streams. *Knowledge and Information Systems*, 24(2):171–196, 2010.

[8] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *Journal of Computer Science and Technology*, 15(6):487–499, 1994.

[9] D G Altman and J M Bland. Statistics notes: the normal distribution. *BMJ (Clinical research ed.)*, 310:298, 1995.

[10] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *American Statistician*, 46(3):175–185, 1992.

[11] Arvind Arasu, Brian Babcock, Shivnath Babu, Jon McAlister, and Jennifer Widom. Characterizing memory requirements for queries over continuous data streams, 2004.

[12] a Asuncion and D J Newman. UCI Machine Learning Repository: Data Sets, 2015.

[13] Mohannad M. Ayash. Research Methodologies in Computer Science and Information Systems. *Computer Science*, 2014:1–4, 2014.

[14] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. *Proceedings of the twentyfirst ACM SIGMODSIGACTSIGART symposium on Principles of database systems PODS 02*, pages(2002-19):1, 2002.

[15] Enzo Baccarelli, Nicola Cordeschi, Alessandro Mei, Massimo Panella, Mohammad Shojafar, and Julinda Stefa. Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: Review, challenges, and a case study, 2016.

[16] Stephen H. Bach and Marcus A. Maloof. A Bayesian Approach to Concept Drift. *NIPS*, pages 1–9, 2008.

[17] Stephen H. Bach and Marcus A. Maloof. Paired learners for concept drift. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 23–32, 2008.

[18] Anand Bahety. Extension and Evaluation of ID3–Decision Tree Algorithm. *Entropy (S)*, 2:1.

[19] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 139, 2009.

[20] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. New ensemble methods for evolving data streams. *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, page 139, 2009.

[21] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. MOA: Massive Online Analysis, a framework for stream classification and clustering. 2010.

[22] Christian Borgelt. Frequent item set mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):437–456, 2012.

[23] Henrik Boström and Peter Idestam-Almquist. Induction of logic programs by example-guided unfolding. *Journal of Logic Programming*, 1999.

[24] Nigel Bruce, Daniel Pope, and Debbi Stanistreet. *Quantitative Methods for Health Research: A Practical Interactive Guide to Epidemiology and Statistics*. 2008.

[25] Don Carney, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, and Stan Zdonik. Monitoring Streams – A New Class of Data Management Applications. *Vldb*, pages 215–226, 2002.

[26] Jadzia Cendrowska. PRISM: An algorithm for inducing modular rules, 1987.

[27] Peter Clark and Robin Boswell. Rule induction with CN2: Some recent improvements. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 482 LNAI, pages 151–163, 1991.

[28] Peter Clark and Tim Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3(4):261–283, 1989.

[29] Lior Cohen, Gil Avrahami, Mark Last, and Abraham Kandel. Info-fuzzy algorithms for mining dynamic data streams. *Applied Soft Computing*, 8(4):1283–1294, 2008.

[30] William W. Cohen. Fast effective rule induction. *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[31] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. Approximate aggregation techniques for sensor databases. In *Proceedings - International Conference on Data Engineering*, volume 20, pages 449–460, 2004.

[32] D. Coomans and D. L. Massart. Alternative k-nearest neighbour rules in supervised pattern recognition. Part 1. k-Nearest neighbour classification by using alternative voting rules. *Analytica Chimica Acta*, 136(C):15–27, 1982.

[33] C Cortes and V Vapnik. Support vector machine. *Machine learning*, pages 1303–1308, 1995.

[34] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[35] A Criminisi, J Shotton, and E Konukoglu. Decision Forests for Classification , Regression , Density Estimation , Manifold Learning and Semi-Supervised Learning. *Learning*, 2011.

[36] Michael Crotty. The foundations of social research: Meaning and perspective in the research process. *SAGE*, 6(4):449, 1998.

[37] Gianmarco De Francisci Morales, Albert Bifet, Latifur Khan, Joao Gama, and Wei Fan. IoT Big Data Stream Mining. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 2016.

[38] Magdalena Deckert. Incremental rule-based learners for handling concept drift: an overview. *Foundations of Computing and Decision Sciences*, 38(1):35–65, 2013.

[39] Pam Denicolo and Lucinda Becker. *Developing Research Proposals*. 2017.

[40] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, pages 71–80, 2000.

[41] Pedro Domingos and Geoff Hulten. A general framework for mining massive data streams. *Journal of Computational and ...*, pages 1–5, 2003.

[42] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and Unsupervised Discretization of Continuous Features. In *Machine Learning Proceedings 1995*, pages 194–202. 1995.

[43] Alan C Elliott and Wayne A Woodward. Statistical Analysis Quick Reference Guidebook: With SPSS Example, 2007.

[44] Ericsson. The cellular Internet of Things – technologies, standards and performance, 2017.

[45] Alan Fern and Robert Givan. Online ensemble learning: An empirical study. *Machine Learning*, 53(1-2):71–109, 2003.

[46] Francisco Ferrer-Troyano, Jesús S Aguilar-Ruiz, and José C Riquelme. Discovering Decision Rules from Numerical Data Streams. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 649–653, 2004.

[47] Ellie Fossey, Carol Harvey, Fiona McDermott, and Larry Davidson. Understanding and evaluating qualitative research. *Australian and New Zealand Journal of Psychiatry*, 2002.

[48] Jerome H. Friedman and Nicholas I. Fisher. Bump Hunting in High-Dimensional Data. *Statics and Computing*, 9:123–143, 1999.

[49] Johannes Fuernkranz, Dragan Gamberger, and Nada Lavrač. *Rule Learning in a Nutshell*. 2012.

[50] J Fürnkranz, D Gamberger, and N Lavrač. *Foundations of rule learning*. Number 2003. 2012.

[51] Mohamed Medhat Gaber. Data Stream Mining Using Granularity-Based Approach. *Foundations of Computa- tional Intelligence*, 6:47–66, 2009.

[52] Mohamed Medhat Gaber. Advances in data stream mining, 2012.

[53] Mohamed Medhat Gaber, Shonali Krishnaswamy, and Arkady Zaslavsky. On-board Mining of Data Streams in Sensor Networks. *Advanced Methods for Knowledge Discovery from Complex Data*, pages 307–335, 2005.

[54] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Towards an Adaptive Approach for Mining Data Streams. *LNCS*, 3181:189–198, 2004.

[55] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26, 2005.

[56] J Gama and C Pinto. Discretization from Data Streams: Applications to Histograms and Data Mining. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 662–667, 2006.

[57] João Gama and Petr Kosina. Learning decision rules from data streams. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 1255–1260, 2011.

[58] João Gama, Pedro Medas, and Ricardo Rocha. Forest trees for online data. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 632–636. ACM, 2004.

[59] João Gama, Ricardo Rocha, and Pedro Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 523–528. ACM, 2003.

[60] Deepak Ganesan, Deborah Estrin, and John Heidemann. Dimensions: Why Do We Need a New Data Handling Architecture for Sensor Networks? *SIGCOMM Comput. Commun. Rev.*, 33(1):143–148, 2003.

[61] J Gehrke, R Ramakrishnan, and V Ganti. RainForest–A Framework for Fast Decision Tree Construction of Large Datasets. *Data Mining and Knowledge Discovery*, 4:127–162, 2000.

[62] E Gossett. *Discrete Mathematics with Proof*. Wiley, 2009.

[63] Carol Grbich. *Qualitative data analysis: An Introduction*. 2012.

[64] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[65] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2012.

[66] Hossein Hassani. Research methods in computer science: The challenges and issues. *arXiv preprint arXiv:1703.04080*, 2017.

[67] David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

[68] S Hettich and S D Bay. The UCI KDD Archive [http://kdd.ics.uci.edu]. *University of California, Department of Information and Computer Science*, 1999.

[69] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[70] Stefan Hoeglinger and Russel Pears. Use of Hoeffding trees in concept based data stream mining. *2007 Third International Conference on Information and Automation for Sustainability*, pages 57–62, 2007.

[71] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE, 1994.

[72] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, 18:97–106, 2001.

[73] Elena Ikonomovska, João Gama, and Sašo Džeroski. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23:128–168, 2010.

[74] Keki B Irani. Multi-interval discretization of continuous-valued attributes for classification learning. 1993.

[75] Ronald L. Jackson, Darlene K. Drummond, and Sakile Camara. What is qualitative research? *Qualitative Research Reports in Communication*, 8(1):21–28, 2007.

[76] Ruoming Jin and Gagan Agrawal. Efficient decision tree construction on streaming data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 571–576. ACM, 2003.

[77] Saint John Walker. Big Data: A Revolution That Will Transform How We Live, Work, and Think. *International Journal of Advertising*, 2014.

[78] Viktor Jovanoski and N. Lavrač. Classification rule learning with APRIORI-C. *Progress in artificial intelligence*, pages 111–135, 2001.

[79] Petr Kadlec, Bogdan Gabrys, and Sibylle Strandt. Data-driven Soft Sensors in the process industry, 2009.

[80] Hillol Kargupta, Ruchita Bhargava, Kun Liu, Michael Powers, Patrick Blair, Samuel Bushra, James Dull, Kakali Sarkar, Martin Klein, Mitesh Vasa, and David Handy. VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring. In *International SIAM Data Mining Conference*, volume 23, pages 300–312, 2004.

[81] Gary King. Preface: Big data is not about the data.

[82] Richard Brendon Kirkby. *Improving hoeffding trees*. PhD thesis, The University of Waikato, 2007.

[83] Sotiris B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, 2007.

[84] David Lane. History of Normal Distribution.

[85] David Lane. Introduction to Normal Distributions.

[86] Mark Last and Mark Last. Online classification of nonstationary data streams. *Intelligent Data Analysis*, 6(2):129–147, 2002.

[87] Yan-Nei Law and Carlo Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 108–120, 2005.

[88] F A Lees. *Financial Exchanges: A Comparative Approach*. Taylor & Francis, 2012.

[89] Bing Liu, Wynne Hsu, Yiming Ma, and Blwhy Ma. Integrating classification and association rule mining. In *4th International Conference on Knowledge Discovery and Data Mining*, pages 80–86, 1998.

[90] S. Madden and M.J. Franklin. Fjording the stream: an architecture for queries over streaming sensor data. *Proceedings 18th International Conference on Data Engineering*, 2002.

[91] Oded Maron and Aw Moore. Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. *Advances in Neural Information Processing Systems*, pages 59–66, 1993.

[92] Dayrelis Mena-Torres and Jesús S. Aguilar-Ruiz. A similarity-based approach for data stream classification. *Expert Systems with Applications*, 41(9):4224–4234, 2014.

[93] R S Mickalstd. LEARNING BY BEING TOLD AND LEARNING FROM EXAMPLES: AN EXPERIMENTAL COMPARISON OF THE TWO METHODS OF KNOWLEDGE ACQUISITION.

[94] S Muthukrishnan. Data Streams: Algorithms and Applications, 2005.

[95] Allen Newell and Herbert A. Simon. Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 1976.

[96] Nokia. Igniting the new businesses of the Internet of Things, 2017.

[97] Dan Noyes. The Top 20 Valuable Facebook Statistics - Updated February 2014, 2014.

[98] Nikunj C. Oza and Stuart Russell. Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, pages 359–364, 2001.

[99] Julie Pallant. *SPSS Survival Manual: A step by step guide to data analysis using SPSS*, volume 3rd. 2010.

[100] Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. New Options for Hoeffding Trees. In *AI 2007: Advances in Artifical Intelligence*, pages 90–99. 2007.

[101] Uros Pompe, Matevz Kovacic, and Igor Kononenko. SFOIL: Stochastic Approach to Inductive Logic Programming. 1996.

[102] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[103] J. R. Quinlan. Learning Logical Definitions from Relations. *Machine Learning*, 5(3):239–266, 1990.

[104] J R Quinlan. *C4.5: Programs for Machine Learning*, volume 1. 1993.

[105] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[106] Ramesh Rajagopalan and Pramod K. Varshney. Data-aggregation techniques in sensor networks: A survey, 2006.

[107] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why Should I Trust You?Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 39(2011):117831, 2016.

[108] Ronald L. Rivest. Learning Decision Lists. *Machine Learning*, 2(3):229–246, 1987.

[109] Thuraya Al Riyami. Main Approaches to Educational Research. *International Journal of Innovation and Research in Educational Sciences*, 2(Online):2349–5219, 2015.

[110] E Sandewall and C G Jansson. Handling imperfect data in inductive logic programming. In *Scandinavian Conference on Artificial Intelligence-93: Proceedings of the Fourth Scandinavian Conference on Artificial Intelligence Electrum, Stockholm, Sweden, May 4-7, 1993*, volume 18, page 111. IOS Press, 1993.

[111] Jeffrey C Schlimmer and Richard H Granger. Beyond Incremental Processing: Tracking Concept Drift. In *AAAI*, pages 502–507, 1986.

[112] Martin Scholz and Ralf Klinkenberg. An Ensemble Classifier for Drifting Concepts. *Proceedings of the Second International Workshop on Knowledge Discovery in Data Streams*, pages 53–64, 2005.

[113] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and Beyond: New Aggregation Techniques for Sensor Networks. *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.

[114] Frederic Stahl and Max Bramer. Computationally efficient induction of classification rules with the PMCRI and J-PMCRI frameworks. *Knowledge-Based Systems*, 35:49–63, 2012.

[115] Frederic Stahl, Mohamed Medhat Gaber, and Manuel Martin Salvador. eRules: A modular adaptive classification rule learning algorithm for data streams. *Res. and Dev. in Intelligent Syst. XXIX: Incorporating Applications and Innovations in Intel. Sys. XX - AI 2012, 32nd SGAI Int. Conf. on Innovative Techniques and Applications of Artificial Intel.*, pages 65–78, 2012.

[116] W Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382. ACM, 2001.

[117] Douglas Terry, David Goldberg, David Nichols, and Brian Oki. Continuous queries over append-only databases, 1992.

[118] Hendrik Theron and Ian Cloete. BEXA: A covering algorithm for learning propositional concept descriptions. *Machine Learning*, 24(1):5–40, 1996.

[119] David De Vaus. Analyzing Social Science Data 50 Key Problems in Data Analysis. In *Analyzing Social Science Data 50 Key Problems in Data Analysis*. 2002.

[120] Jeffrey Scott Vitter. Algorithms and Data Structures for External Memory, 2006.

[121] Larry Watanabe and Others. Learning structural decision trees from examples. *Urbana*, 51:61801, 1991.

[122] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[123] Ian .H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. 2005.

[124] Dengyuan Wu, Kai Wang, Tao He, and Jicheng Ren. A dynamic weighted ensemble to cope with concept drifting classification. In *Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008*, pages 1854–1859, 2008.

[125] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining, 2008.

[126] Zhongheng Zhang. Naïve Bayes classification in R. *Annals of Translational Medicine*, 4(12):241–241, 2016.

[127] I Zliobaite, Albert Bifet, M Gaber, and B Gabrys. Next challenges for adaptive learning systems. *SIGKDD Explorations*, 14(1):48–55, 2012.

Appendices

Appendix A

Publications Related to this Research

The works in this thesis resulted and contributed in the following publications:

Direct contributions from this research:

- Le, T., Stahl, F., Gomes, J. B., Gaber, M. M., Di Fatta, G. (2014). Computationally efficient rule-based classification for continuous streaming data. In Research and Development in Intelligent Systems XXXI (pp. 21-34). Springer, Cham. (*One of top three papers in technical stream at SGAI-2014 conference.)*, **Appendix B**.

- Le, T., Stahl, F., Gaber, M. M., Gomes, J. B., Di Fatta, G. (2017). On Expressiveness and Uncertainty Awareness in Rule-based Classification for Data Streams. Neurocomputing.

- Le, Thien, et al. "A Statistical Learning Method to Fast Generalised

Rule Induction Directly from Raw Measurements." Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on. IEEE, 2016.

- Le, Thien, et al. "An Adaptive and Fast Rule-based Classification Algorithm for Streaming Data". (In Press), Expert Systems: the Journal of Knowledge Engineering.

Indirect contributions from this research:

- Almutairi, M., Stahl, F., Jennings, M., Le, T., Bramer, M. (2016). Towards Expressive Modular Rule Induction for Numerical Attributes. In Research and Development in Intelligent Systems XXXIII: Incorporating Applications and Innovations in Intelligent Systems XXIV 33 (pp. 229-235). Springer International Publishing.

- Wrench, C., Stahl, F., Le, T., Di Fatta, G., Karthikeyan, V., Nauck, D. (2016). A Method of Rule Induction for Predicting and Describing Future Alarms in a Telecommunication Network. In Research and Development in Intelligent Systems XXXIII: Incorporating Applications and Innovations in Intelligent Systems XXIV 33 (pp. 309-323). Springer International Publishing. Chicago

Appendix B

One of Top Three Papers at SGAI-2014 Conference

34th Annual International Conference of

BCS The Chartered Institute for IT

Specialist Group on Artificial Intelligence

9th -11th December 2014 Cambridge, UK.

The Specialist Group on Artificial Intelligence (SGAI) recognises the work of:

T.Le and F.Stahl (University of Reading, UK), J.B.Gomes (Institute for Infocomm Research (I2R), Singapore), M.M.Gaber (Robert Gordon University, UK) and G.Di Fatta (University of Reading, UK)

and the paper entitled:

Computationally Efficient Rule-Based Classification for Continuous Streaming Data

[redacted]

Max Bramer

Chairman

Appendix C

Implementation of Hoeffding's Inequality Proof

R code used to simulate and confirm expected behaviours of Hoeffding's Inequality.

```
# start of the code ######################
##########################################

# set seed before rune
set.seed(1)

# create vector 10,000 items where 'A' is 30%
x <- c(rep(x = "A", 3000), rep(x = "B", 7000))

# calculate epsilon (Hoeffding Bound) as described
```

```
11 epsilon <- sqrt(log(1/0.05)/(2*200))
12
13 # flag whether the error rate is violated
14 outOfBound <- FALSE
15
16 # run 10,000 times
17 for(i in 1:10000){
18   falseCount <- 0
19   for(j in 1:100){
20     # select 200 items randomly
21     selectedItems <- sum(sample(x, 200, replace = TRUE)
       ↪ == "A")
22
23     # calculate the detla for probabilty of 'A'
24     delta <- abs(((selectedItems / 200) - 0.3))
25
26     # check if the bound
27     e.pro <- delta > epsilon
28
29     # increase error counter
30     if(e.pro == TRUE) falseCount <- falseCount + 1
31   }
32
33   # check if error rate excessed the defined threshold
```

```
34    if(falseCount > 5) outOfBound <- TRUE
35 }
36
37 #check if the bound
38 print(paste0("Was error rate violated: ", outOfBound) )
39
40 # end of the code ######################
41 ##########################################
```

Appendix D

Implementation of R Code for Numeric Rule Term

R code used to produce examples for numeric rule term based on Gaussian distribution.

```
1  library(ggplot2)
2  set.seed(1)
3  t1 <- round(rnorm(100, 50, 6))
4  t2 <- round(rnorm(100, 40, 6))
5  t3 <- round(rnorm(100, 60, 6))
6
7  d1 <- data.frame(x = t1, Class = 1)
8  d2 <- data.frame(x = t2, Class = 2)
9  d3 <- data.frame(x = t3, Class = 3)
10 data <- rbind(d1, d2, d3)
```

```
11 data$Class <- as.factor(data$Class)
12
13 max_density <- max(density(d$x)$y)
14 max_density_x <- density(d$x)$x[density(d$x)$x >= max_
       ↪ density]
15
16 d <- density(t1)
17
18 densoity_approx <- approx(d$x, d$y, xout = t1)
19 densoity_approx <- data.frame(densoity_approx)
20
21 max_density <- densoity_approx$x[densoity_approx$y >=
       ↪ max(densoity_approx$y)]
22
23
24 densoity_approx <- approx(d$x, d$y, xout = t1[t1 > max(
       ↪ max_density)])
25 densoity_approx <- data.frame(densoity_approx)
26 max_density_right <- densoity_approx$x[densoity_approx$
       ↪ y >= max(densoity_approx$y)]
27 max_density_right <- max(max_density_right)
28
29 densoity_approx <- approx(d$x, d$y, xout = t1[t1 < max(
       ↪ max_density)])
```

```
densoity_approx <- data.frame(densoity_approx)
max_density_left <- densoity_approx$x[densoity_approx$y
    ↪  >= max(densoity_approx$y)]
max_density_left <- min(max_density_left)

g <- ggplot(data, aes(x = x, colour = Class, fill =
    ↪ Class))
g <- g + geom_density(alpha = 0.2) + xlim(0, 100)
g <- g + geom_vline(xintercept=max_density, color = "
    ↪ green", size = 0.8)
g <- g + geom_vline(xintercept=max_density_left, color
    ↪ = "purple", linetype="dashed", size = 1.5)
g <- g + geom_vline(xintercept=max_density_right, color
    ↪  = "blue", linetype="dashed", size = 1.5)
g <- g + ggtitle("Density Plot")
g <- g + xlab("Numeric Value")
g <- g + ylab("Class Density for Numeric Values")
g


g + annotate("rect", xmin = -Inf, xmax = 30, ymin = -
    ↪ Inf, ymax = Inf, fill = "gray", alpha = .5, color
    ↪  = NA)
```