Intelligent Adaptation of Ensemble Size in Data Streams Using Online Bagging

Author:
Muhammed Kehinde Olorunnimbe

Supervisor:
Dr. Herna L. Viktor

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfilment of the requirements for the degree of

Master of Science in Systems Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

uOttawa

University of Ottawa

Abstract

In this era of the Internet of Things and Big Data, a proliferation of connected devices continuously produce massive amounts of fast evolving streaming data. There is a need to study the relationships in such streams for analytic applications, such as network intrusion detection, fraud detection and financial forecasting, amongst other. In this setting, it is crucial to create data mining algorithms that are able to seamlessly adapt to temporal changes in data characteristics that occur in data streams. These changes are called *concept drifts*. The resultant models produced by such algorithms should not only be highly accurate and be able to swiftly adapt to changes. Rather, the data mining techniques should also be fast, scalable, and efficient in terms of resource allocation. It then becomes important to consider issues such as storage space needs and memory utilization. This is especially relevant when we aim to build personalized, near-instant models in a Big Data setting.

This research work focuses on mining in a data stream with concept drift, using an *online bagging* method, with consideration to the memory utilization. Our aim is to take an adaptive approach to resource allocation during the mining process. Specifically, we consider metalearning, where the models of multiple classifiers are combined into an ensemble, has been very successful when building accurate models against data streams. However, little work has been done to explore the interplay between accuracy, efficiency and utility. This research focuses on this issue. We introduce an adaptive metalearning algorithm that takes advantage of the memory utilization cost of concept drift, in order to vary the ensemble size during the data mining process. We aim to minimize the memory usage, while maintaining highly accurate models with a high utility.

We evaluated our method against a number of benchmarking datasets and compare our results against the state-of-the art. Return on Investment (ROI) was used to evaluate the gain in performance in terms of accuracy, in contrast to the time and memory invested. We aimed to achieve high ROI without compromising on the accuracy of the result. Our experimental results indicate that we achieved this goal.

Acknowledgements

All praises to God Almighty.

A special gratitude to my supervisor, Dr. Herna L. Viktor, for guiding me through this thesis, from the beginning when I was confused and overwhelmed, to the end when I was quite less confused but still overwhelmed. Thank you for your patience, support and understanding. I have learnt so much from you, and it has been an honour working with you.

Thank you to my mum, my twin sister and the rest of my siblings for the unwavering love and encouragement, through this, and other aspects of my life. Thank you to my dear wife, Jemilat Animashaun, for your limitless love, selfless companionship and enduring support through this journey. I love you.

I would also like to thank my friends and colleagues who have been encouraging and supportive through this endeavour.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ADAMS Advanced Data mining And Machine learning System

ADWIN Adaptive Windowing

AES Adaptive Ensemble Size

AI Artificial Intelligence

Amazon EC2 Amazon Elastic Compute Cloud

ANOVA Analysis of Variance

API Application Program Interface

AUC Area Under the Curve

Bagging Bootstrap Aggregating

CART Classification And Regression Trees

DARPA Defense Advanced Research Projects Agency

DOS Denial-Of-Service

EHA Event History Analysis

FN False Negative

FP False Positive

GUI Graphical User Interface

HT Hoeffding tree

ICMP Internet Control Message Protocol

ID3 Iterative Dichotomiser 3

IMDb Internet Movie Database

IoT the Internet of Things

KDD Knowledge Discovery in Databases

LED Light Emitting Diode

MEKA Multi-label Extension to WEKA

MOA Massive Online Analysis

OzaBag Oza and Russell's Online Bootstrap Aggregating

OzaBagADWIN OzaBag with ADWIN

OzaBoost Oza and Russell's Online Boosting

OzaBoostADWIN OzaBoost with ADWIN

PAC Probably Approximately Correct

R2L Remote to Local

RAM Random Access Memory

RIS Resource Information System

ROI Return on Investment

TCP Transmission Control Protocol

TF-IDF Term Frequency - Inverse Document Frequency

TN True Negative

TP True Positive

U2R User to Root

UBDM Utility-Based Data Mining

UDP User Datagram Protocol

UKD Ubiquitous Knowledge Discovery

USFS United States Forest Service

USGS United States Geological Survey

VFDT Very Fast Decision Trees

WEKA Waikato Environment for Knowledge Analysis

CHAPTER 1

Introduction

Making sense of data is a very important task. Like never before, we are faced with very fast-paced data that are produced at a large scale. "Big Data" is now a common phrase in data analytics, with data being generated every second at an exponential rate. This trend is projected to continue. According to a 2012 report by the data analytics company, IDC (Gantz and Reinsel, 2012), the future data growth rate is estimated at 300%, with the overall data generated to be at 40 zettabyte by 2020. That is 40 trillion gigabyte, or to put in perspective, 5,200 gigabyte for every person on earth.

In this scenario, there is a constant flow of data that needs to be analysed at some point, and it is very difficult to analyse all data at any given point. Aside from the complexity introduced by the constant flow of data, there is also the issue of unforeseen changes in data distribution, with time. This change phenomenon is referred to as *concept drift* (Gama, Žliobaitė, Bifet, Pechenizkiy, and Bouchachia, 2014). Techniques are therefore needed to analyse these types of fast evolving data stream.

In this thesis, we focus our attention on algorithms that aim to discover knowledge from data streams that are susceptible to concept drift. Specifically, we study the use of *Ensemble methods,* since these types of supervised classification technique have shown promising results for data mining tasks in streams with concept drift (Kuncheva, 2004; Bifet, Holmes, Pfahringer, Kirkby, and Gavaldà, 2009b; Wang and Pineau, 2013). An ensemble method involves building a better predictive model by integrating multiple models from one or multiple classification algorithms (or so-called base learners), effectively reducing the

chances of misclassification. According to Bifet, Holmes, Pfahringer, Kirkby, and Gavaldà (2009b), ensemble methods *"can adapt to change quickly by pruning under-performing parts of the ensemble, and they therefore usually also generate more accurate concept descriptions"*, in comparison to single classifier methods.

The number (or size) of the ensemble of models to build the composite model from is usually predetermined. However, this may not be the ideal setting, especially when a stream is suspect to concept drift. This research introduces a methodology that adapts the size of the ensemble based on concept drift in the stream, in order to balance accuracy and cost, where cost is defined in terms of resource allocation.

The rest of this chapter contains the motivation for our study, and the outline of this thesis.

1.1 Motivation

In data analytics, there is a constant requirement to improve on the results obtained. In applications such as intrusion detection, medical prediction and fraud detection, amongst other, misclassification could be counter-productive and dangerous. It, therefore, becomes imperative to continuously thrive for better classification accuracy. In the current era of virtualization, parallel and distributed computing, we may choose to increase the resources allocated to a mining task in a bid to improving the results to be obtained. However, this comes at a cost, in terms of memory utilization, time and processor usage. Although the cost of computing has become relatively cheaper over time, the volume of data has also become massive. This research work takes this cost into consideration, by way of adapting the mining technique to the resource usage. Our approach optimizes for accuracy, with consideration to the computing cost.

Many data mining and machine learning algorithms have been proposed and have been successful in numerous domains (Brazdil, Giraud-Carrier, Soares, and Vilalta, 2009; Rossi, De Carvalho, Soares, and De Souza, 2014). Specifically, ensemble methods, such as *Boosting and Bagging*, have been found to improve on classification accuracy when compared to standard base learners. A number of these previous research focuses on factors such as the right combination of ensemble of models (simply called ensemble), the right size to use for the ensemble, the kind of ensemble technique to use, and other such subjects relating to the characteristics of an ensemble. The cost factors referred to as the *economic cost* in stream mining has, however, not been given as much attention.

Furthermore, determining the optimal size of the ensemble is not a trivial task. In the

context of resource allocation, the direct resource associated with metalearning is the size of the ensemble used. This indirectly interprets to the computing cost, and we further focus on this issue in our research.

We considered the *Online* version of two start-of-the-art ensemble learners. Online learners are the types of classification algorithms that are ideal for data streams because they are able to process chunks of data sequentially, to produce models, without having the complete dataset. The ensemble learning algorithms we considered are *Online Bagging* and *Online Boosting*. Online Bagging draws a random subset of examples from a data stream, and generates independent ensemble of models. The most predicted label is assigned to the new classification instance. Online Boosting, on the other hand, focuses on the hard to learn examples, and uses a weighted vote of the ensemble models to assign a new classification instance (Oza and Russell, 2001).

A number of research have shown that Boosting perform better, in terms of accuracy, when compared to Bagging, especially for non-evolving datasets. However, literature also suggests that Online Bagging algorithm performs better, in terms of accuracy and consistency, in a data stream with concept drift (Oza and Russell; Bifet, Holmes, and Pfahringer, 2010c). For this reason we focus on improving the Online Bagging algorithm with consideration to the cost of the mining process.

1.2 Thesis Objective

Current research in data streams is working towards improving on the classification accuracy, with no specific concern for the monetary cost of the mining process. It follows that there is, however, a monetary cost associated with Big Data. The computing means and cost are to be accounted for, and the larger the resources utilized, the larger the cost. This issue is more prominent in a stream with concept drift, where a fixed estimate cannot be preassigned up front. There is, therefore, a need to introduce cost efficient algorithms, that facilitates improved accuracy, as well as facilitate adaptable resource usage.

To this end, we introduced the *Adaptive Ensemble Size (AES) Online Bagging* algorithm, as an extension of the Online Bagging algorithm. We created an adaptation technique to adapt the size of the ensemble between optimal maximum and minimum values, instead of having a fixed size, as is currently implemented in Online Bagging. This approach was implemented to consider the changes in the data stream with concept drift, and only increase the size of the ensemble when required, in order to guarantee higher accuracy.

Although there have been some research in utility-based data mining in the past

(UBDM, 2005, 2006; Zadrozny, Weiss, and Saar-Tsechansky, 2006), it is still an emerging field in data stream mining. Factors such as cost-sensitive learning and time have received more attention than the cost of building and applying models, in terms of physical resources. In a Big Data setting, we need to take resource utilization into consideration, in order to design energy-efficient learning algorithms. Recently, the term *cost sensitive adaptation* was proposed in (Žliobaite, Budka, and Stahl, 2015), and a framework was introduced using Return on Investment (ROI) as a measurement criteria for measuring costs and benefits in data stream. In the work by Žliobaite, Budka, and Stahl (2015), however, the focus is on measuring the cost of adapting a model, rather than focusing on a holistic cost measure.

In this thesis, we tackle this issue of holistic cost adaptation of data streams that contains concept drift. Our goal is to obtain better ROI, without compromising on the accuracy obtained during the mining process. We compare the efficiency of our implementation with standard Online Bagging, using ROI as the measurement metric. Our results are promising. The strength of our method is that it may be combined with many state-of-the-art ensemble learning algorithms, in order to facilitate cost sensitive learning.

1.3 Thesis Organization

The remainder of this thesis is divided into two parts, consisting of six chapters. The first part consists of the background study and the literature review, in Chapters 2 and 3, respectively. Chapter 2 starts with the introduction of ideas, concepts and methodologies in data mining, and focuses on specific concepts in data stream mining such as concept drift and *adaptivity*. Utility, Efficiency and Cost sensitive adaptation in data stream mining is also discussed in this chapter. In Chapter 3, we introduce *metalearning* and discuss the different algorithms that were utilized in this thesis.

We start the second part of this thesis by introducing our cost and energy efficient approach to improving classification accuracy in a stream with concept drift in Chapter 4. A methodology is detailed in this chapter with the preliminary experimentation and reasoning behind our implementation. The dataset used, our experimentation setup and performance measurement criteria, are detailed in Chapter 5. We present our experimentation results and analysis in Chapter 6. Comparison between our implementation and the state-of-the-art algorithms are provided in this chapter. We provide a conclusion to our research in Chapter 7, providing possible directions for future work.

Part I

PRELIMINARIES

CHAPTER 2

Data Stream Mining: Fundamentals

It is important to understand data streams and the basic concepts and terminologies in data stream mining. Considering that data stream mining as a process of data analytics is a subset of data mining, we complete this literature review following a top down perspective. Data mining, and data stream mining, environments have several techniques and terminologies that are frequently used in their mining processes and the discussion about the taxonomy discussed in section 2.1 is applicable to both. In section 2.2, we proceed with the discussion on data stream mining specifically. We also highlight concept drift and various terminologies associated with it. In the last sections of this chapter, we discuss the issues and applications of data stream mining.

2.1 Taxonomy of Data Mining Tasks and Techniques

There are many definitions of data mining, depending on who is been asked. One general interpretation from these definitions, however, is that data mining involves knowledge discovery from a relatively large amount of data, referred to as KDD (Knowledge Discovery in Databases). This involves using some form of automated computational and statistical processes to examine large datasets in other to generate new information. These processes are referred to as machine learning, an important application of artificial intelligence

(Russell, Norvig, Candy, Malik, and Edwards, 1996).

Taxonomy of data mining can be based on the task to be performed or the technique in use. We will first take a look at the broader classification by the technique employed before looking at the more specific classification by major tasks.

2.1.1 Data Mining Techniques

Data mining problems can generally be categorized into *Supervised (or Predictive)* and *Unsupervised (or Explanatory)* based on the learning technique. Supervised learning methods are methods that attempt to discover the relationship between input attributes (called *training* or *independent* variables) and the target attributes (referred to as *test* or *dependent* variables). The relationship discovered is represented in a structure referred to as a *model*. Usually, models describe and explain phenomena, which are hidden in the dataset and can be used for predicting the values of new unlabelled data based on the labelled training data.

Prediction methods are used to automatically build this behavioural model with the training dataset, and are able to predict values of one or more variables related to new and unseen test dataset. It also develops patterns, which form the discovered knowledge in a way which is understandable and easy to operate upon. Some prediction oriented methods can also help to provide understanding of the dataset. Most of these mining techniques are based on *inductive learning*, where a model is constructed by generalizing from a sufficient number of training examples. In the inductive approach, the implicit assumption is that the trained model is applicable to future unseen data (Maimon and Rokach, 2010). The supervised methods are implemented in variety of domains, such as marketing, finance and manufacturing.

Unsupervised learning refers mostly to techniques that group instances without a pre-specified, dependent attribute. They are oriented to data interpretation, with focus on understanding the relationship between the underlying data and its parts. They are used to uncover hidden patterns within unlabelled data.

Many applications employs both supervised and unsupervised learning techniques. Such an approach is referred to as *Semi-Supervised* learning method. This method uses some unlabelled data with a relatively smaller amount of labelled data for training. This is represented with the dotted lines in figure 2.1. The four primary categorizations of these techniques are also shown in the figure (Kotu and Deshpande, 2015). They are further broken down based on the specific method used, but we would not go into details of these

in this review. We will discuss the algorithms that applies to this research work later in Chapter 3.

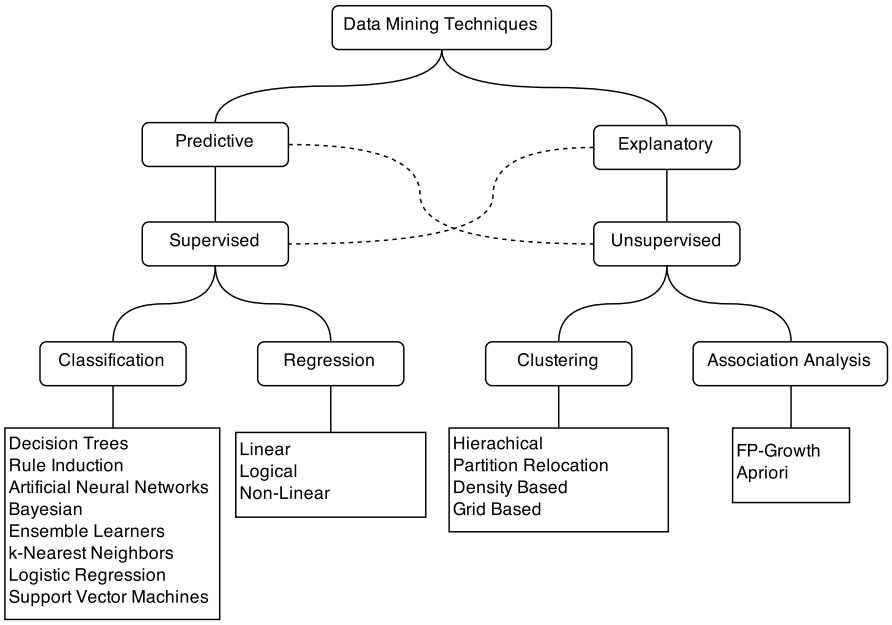


Figure 2.1: Data Mining Techniques

Classification

The classification method is arguably the most common technique used in data mining. Classification algorithms, also referred to as *classifiers*, map the input data into predefined classes provided by the training data. As an example, a classification task might require classifying students by those that has paid their tuition and those with outstanding payment. In that case, a classifier labels students that has paid their tuition as ”good” and those with outstanding payment as ”bad”. The Classification technique is an instance of supervised learning, because of the provided training data with correct labels to base the learning on. In the student example, the classifier is able to identify the good and the bad students based on the training set provided.

Regression

Regression technique estimates the relationship between different variables in a given dataset. Using their respective models, regression algorithms, also referred to as *regressors*, can predict the demand for a certain products given their characteristics. Early work on regression was published by Sir Francis Galton in his 1988 paper (Galton, 1888). He discovered that the height of children of tall parents tend to be slightly shorter than the

parents' while children of short parents are slight taller; the sample *regressed* towards a population mean, hence the term *regression*. Regression involves fitting data with functions or *function fitting*. The value of a dependent variable y is obtained by combining the predictor variables X into a function $y = f(X)$. The value of the known is used to formulate the function that is used to determine the unknown.

Clustering

Besides the two supervised learning methods briefly explained above, for clustering and association analysis, no training data is provided in the learning process. In Clustering, The algorithm identifies points within the dataset that are similar to each other and group them in clusters. By so doing, it means the constituent data within each cluster is similar to one another, and dissimilar to data elements in another cluster. Similarity is commonly measured using the Euclidean distance or other measurement parameters like the Chebychev distance and the percentage disagreement (StatSoft, 2014).

Association Analysis

A key concept in unsupervised learning is *frequent pattern* as this is an important property of datasets. Frequent patterns are simply patterns that appear frequently in data. *Frequent pattern mining* is mining items in a data set to find relationships between them. It is the core concept in *Association Analysis* (Aggarwal and Han, 2014). Unlike predictive methods like classification and regression, this method of data mining is used to find useful patterns in the co-occurrence of multiple items. It measures the strength of co-occurrence amongst these items, discovering hidden patterns in the form of easily recognizable rules in the dataset. It was made better known by Agrawal, Imieliński, and Swami (1993) in the application of the method to determine the set of retail items frequently purchased together in the market basket analysis.

2.1.2 Data Mining Tasks

We can categorize data mining problems based on the mining task to be performed. These tasks are shown in figure 2.2 (Kotu and Deshpande, 2015).

Classification: This involves identifying the category that an incoming data belongs to, based on already known labels of the training set. This task uses supervised classification techniques to sort data into two or more distinct classes or buckets. Models are developed from the training datasets and are applied on new and unseen data. An example of a

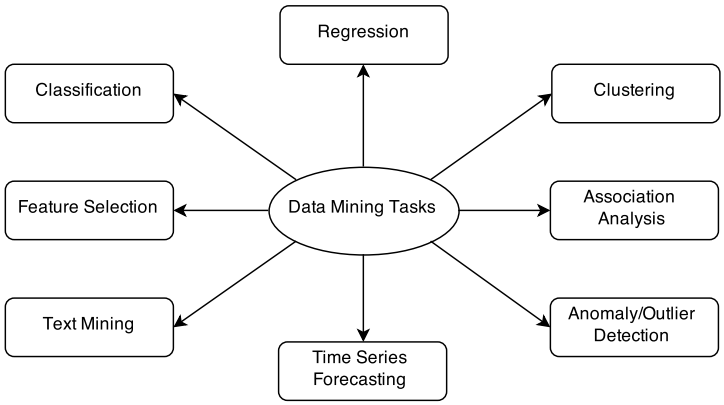


Figure 2.2: Data Mining Tasks

classification task is how spam filters detect spam emails based on known message content and header (Kotu and Deshpande, 2015).

Regression: This is very similar to classification task in the sense that the prediction is based on previously known dataset. The main difference is that while the output variables are categorical or polynomial, in regression, the output variable is numeric (Kotu and Deshpande, 2015).

Clustering: Unlike classification where we have training dataset with known labels, clustering identifies the natural grouping in the data set using unsupervised clustering techniques. An example of a clustering task is the grouping of students in a classroom based on seating arrangement. This differs from a classification task, using the same student group, that might involve classifying by gender. The key difference is that while classification involves determining whether an attribute belongs to a known group, clustering involves dividing data into meaningful groups that are not previously specified. This could also be employed as a preprocessing step for a classification task (Kotu and Deshpande, 2015).

Association Analysis: The objective of association analysis is to find patterns in the co-occurrence of item sets. The task is to determine the relationship that exists between different attributes in a dataset. A well known application of this is the market basket analysis in which co-occurrences are determined between retail items within the same customer transactions. This kind of knowledge will enable the retailer to take advantage of the association, by placing these items together in the store front, or bundling their prices together (Kotu and Deshpande, 2015).

Anomaly/Outlier Detection: The task here is to identify the data attributes that are significantly different from the others in a dataset. This involves using a supervised or semi-supervised learning technique, and the learning accuracy is improved with time. This task is used in cases of bank fraud, intrusion detection, amongst others (Kotu and Deshpande, 2015).

Time Series Analysis: This involves making forecast into future values or events based on past values of the same attribute. The technique frequently used for this is regression analysis. We may recall that in normal regression analysis, we use the value of the unknown variable to formulate the fitting function used to obtain the unknown, usually of a different class. In time series analysis, we use historic data of a specific class to make forecast about the same class. Weather forecasting and pattern recognition are typical applications of time series analysis (Kotu and Deshpande, 2015).

Text Mining: With the relatively recent exponential increase in social media usage, text mining has become very important in data mining and predictive analytics. As is apparent by the name, this is mining data in which the input data is text. This can be in the form of documents, emails, tweets, facebook posts amongst others. The texts are first converted to semi-structured data, where each unique word is considered as an attribute. We can then apply any of the already mentioned data mining techniques, depending on the task at hand. Sentiment analysis is a text mining task, where the polarity of a sentence or phrase is determined based on comparison of the constituent words with training examples (Kotu and Deshpande, 2015).

Feature Selection: This literally means selecting only the attributes or features that matters at the point in time. This is usually a task for data preprocessing before a data mining technique is used. Feature selection is particularly useful in a regression analysis task with many predictor variables. As the number of predictors X increases, it adds computational overhead and reduces the ability to obtain good models. It becomes essential to reduce the number of predictors using feature selection to the required minimum while guaranteeing a good result (Kotu and Deshpande, 2015).

2.2 Data Streams

Advances in technologies in recent years have enabled us to automatically transact information about many activities at a fast rate. Such information transactions generates huge amount of online data growing at an unlimited rate. This kind of continuous flow of data

are referred to as data streams. From computer network traffic, to social network streams, it is practically impossible to do without streams of data in modern live. It becomes imperative to be able to extract meaningful information from this vast, fast-paced data, hence this yields the need for data stream mining.

Data streams differ from static data in a number of aspects, and the difference is fundamental to data stream processing. Some of the key characteristics are itemized below:

- *Unboundedness:* Data streams are potentially infinite in nature and they are typically not stored in their entirety (Gaber, Zaslavsky, and Krishnaswamy, 2004; Gama, 2010).
- *Temporally ordered:* In most application of data streams, the characteristics of the stream elements evolve over time. This property is referred to as temporal locality and adds an inherent temporal component to the data stream mining process (Aggarwal, 2007). This characteristic will be an important factor in measuring the accuracy of the algorithms to be discussed later.
- *High rate of data generation:* Data stream elements are known to be generated at a rapid rate in some applications. Thus the rate of processing such elements has to be timely in order to keep up (Franke, 2008).

The *Temporally ordered* characteristics causes the data distribution to change over time, yielding the phenomenon called *concept drift* (Gama, Žliobaitė, Bifet, Pechenizkiy, and Bouchachia, 2014). This area of data stream mining is frequently called *evolving data stream mining* or *non stationary learning*, and it is actively being researched upon. This thesis is in this field of study. We will discuss more about concept drift in section 2.4.

2.2.1 Online vs Offline Learning

In sections 2.1.1 and 2.1.2, we discussed data mining and its various techniques and tasks. In those discussions, we made a common assumption that the dataset to be processed is available as a whole at the time of processing. This form of learning is referred to as *offline learning*. The model produced from the offline learning process can then be used for prediction when the training is completed. This differs considerably in the case of data stream mining. In data stream mining, the data to be evaluated is never fully available at one time, and can only be evaluated in sequence. Hence the method to be used should be able to process the dataset sequentially and produce models for predictions without having the complete dataset. Such learning process is referred to as *online learning*. In

online learning process, the model is continuously updated as more data arrives (Gama, Žliobaitė, Bifet, Pechenizkiy, and Bouchachia, 2014).

Online learning algorithms updates their model by *incremental learning*. In incremental learning, the learning process takes place batch-by-batch, and the model is updated when new data becomes available. It should be noted that an incremental learner does not necessarily have to process streaming data, as is typically required of online learner (Khreich, Granger, Miri, and Sabourin, 2012). In online learning, data is typically processed once, and have limited memory and time for each processed item.

Online learning methods has the advantage of processing real-time, fast-paced and adaptive datasets, referred to as *'fast data'*. Offline learning methods on the other hand, has the advantage of processing large datasets, called *'big data'*, requiring longer processing time and larger abstraction.

2.2.2 Data Stream Mining

Recall that we discussed the different data mining tasks and techniques in section 2.1. While some of these concepts applies to fast evolving data streams, the general understanding of conventional data mining approaches is that we are processing static, offline dataset. In data mining, the learning algorithm can run through the same set of dataset examples multiple times. The processing time and memory is also relatively unlimited as we have control over these parameters and can choose to alter as we see fit. We also get the most precise representation of the result by the learning algorithm, and we have the luxury of adjusting different parameters multiple times to obtain even better results (Gama, 2010).

All of these points, however, are the direct opposite when we consider the process of mining streams of data. In most instance of data stream mining, the algorithm will only have a maximum of one pass on the data, under limited time and processing power. Because we usually cannot evaluate the data more than once, and only with a portion of the dataset, only an approximate accuracy is guaranteed. The data stream might also be distributed across multiple source and it become imperative for our stream mining system to consolidate the datasets in the mining process (Gama, 2010).

Table 2.1: Difference Between Data Mining and Data Stream Mining

	Data Mining	Data Stream Mining
Number of passes	Multiple	Single
Processing Time	Unlimited	Restricted
Memory Usage	Unlimited	Restricted
Type of Result	Accurate	Approximate

These key differences are summarized in table 2.1.

2.3 Methodologies of Data Stream Mining Systems

Recall that we previously itemized the characteristics of data streams in section 2.2. The *Unboundedness* and *High rate of data generation* nature of a data stream limits the amount of data that can be processed and the time it can be processed. This has led to various data stream summarization and reduction methodologies employed in the mining systems. While the classifications of techniques and tasks discussed in section 2.1 applies to data mining and data stream mining, these methodologies as shown in figure 2.3 applies only to stream mining because of their obvious characteristics. The techniques are used for producing approximate answers from the data stream usually by transforming the data to a suitable form for analysis, considering that we cannot capture the entirely of a data stream for analysis.

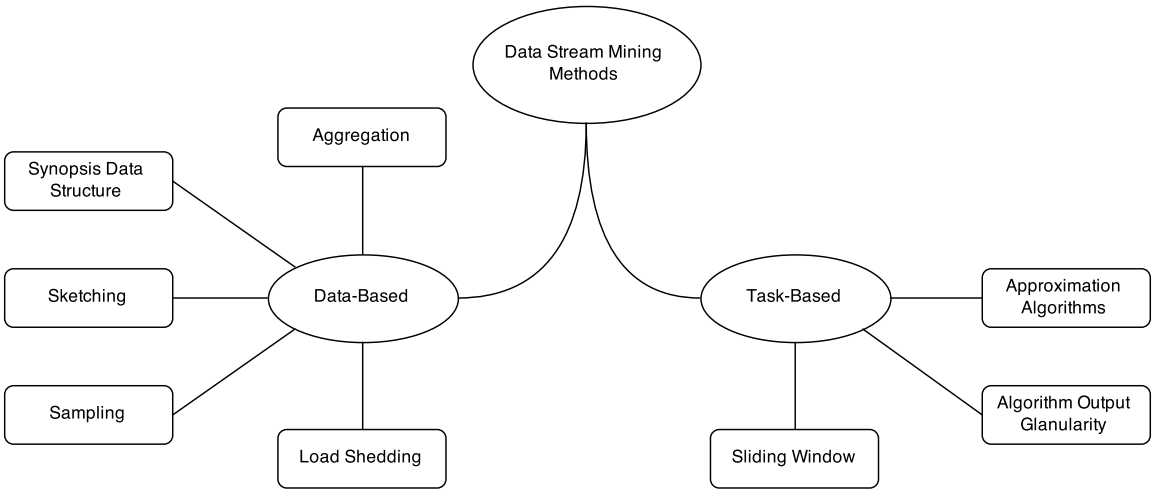


Figure 2.3: Taxonomy of Data Stream Mining Systems' Methods

2.3.1 Data-Based Methods

Sampling: Sampling is the process of representing the data stream with a small sample of the stream by statistically selecting the elements of the incoming stream to be analysed at periodic intervals (Toivonen, 1996). It is the easiest form of stream summarization and other techniques can be built from the sample. Very Fast Machine Learning technique (Domingos and Hulten, 2000) uses the Hoeffding Bound (Hoeffding, 1963) in its measurement of the sample size, according to a loss function derived from the running algorithm. The problem with sampling for data stream analysis is the unknown data

size and fluctuating data rates. A modification to sampling to address this problem is *reservoir sampling*.

Sketching: This technique trades off accuracy for storage, unlike the other ones. Sketching builds a statistical summary of the data using a small amount of memory by vertically sampling the incoming stream (Babcock, Babu, Datar, Motwani, and Widom, 2002; Cormode, Datar, Indyk, and Muthukrishnan, 2002). The entirety of the dataset is represented with only important information, using a very small amount of space. Sketching techniques are very convenient for distributed computation over multiple streams. The primary drawback, though, is its accuracy, making it relatively not very effective in stream mining.

Synopsis Data Structures: Creating a synopsis of data refers to the process of applying summarization techniques that are capable of summarizing the incoming stream for further analysis. Synopsis data structures uses small space approximation solution to massive data set problem. Examples are *Wavelet analysis* (Gilbert, Kotidis, Muthukrishnan, and Strauss, 2003), *histograms, quantile and frequency moments* (Babcock, Babu, Datar, Motwani, and Widom, 2002). Wavelets coefficients are projections of the given set of data onto an orthogonal set of basic vector, and they only analyse the precise data by detecting and determining the positions of abrupt signals. In Histogram technique, the data is partitioned into a set of contiguous buckets, with varying width and depth based on the partitioning rule. Histograms can be used to approximate query answers rather than using sampling techniques.

Aggregation: Aggregation is the representation of a number of elements in one element using statistical measures such as means, variance or average. The primary problem with aggregation is that it does not perform well in a stream with fluctuating data distribution. This approach has been successfully used in a distributed stream data environment and with continuous queries over data stream (Babu and Widom, 2001). Merging of online stream aggregation and offline mining was extensively studied for clustering and classification of data streams (Aggarwal, Han, Wang, and Yu, 2003, 2004a,b).

Load Shedding: Load shedding refers to the process of eliminating a batch of subsequent elements from being analysed (Tatbul, Çetintemel, Zdonik, Cherniack, and Stonebraker, 2003; Mayur, Babcock, Datar, and Motwani, 2003). This method is not frequently used because it drops chunks of data that might represent a pattern of interest. Its also shares similar problems with sampling. *Loadstar* algorithm represents the first attempt at using

load shedding for solving high speed data stream classification problem (Chi, Wang, and Yu, 2005).

2.3.2 Task-Based Methods

Sliding Window: The idea of the sliding window technique is that rather than running statistical computations on all or some of the data seen so far, we make decisions based on "recent" data only (Babcock, Babu, Datar, Motwani, and Widom, 2002). More formally, at every time t, a new data element arrives. This element expires at time $t + w$, where w is the window size or length. This technique has an advantage of reduced memory requirements because not all data is attempted to be analysed or stored. The size of the window over time could be of a fixed or variable size (Gama, Žliobaitė, Bifet, Pechenizkiy, and Bouchachia, 2014). For the fixed size sliding window, a fixed number of most recent data is stored, and the oldest is discarded when new data arrives. In a data stream with *concept drift* a variable sized window is preferred because the window size shrinks or grows based on the data distribution. Section 2.4 discusses more about concept drift.

Approximation Algorithm: Approximation algorithms are specifically designed for computationally hard problems (Muthukrishnan, 2003). These make it desirable for data stream mining, given its features of continuity, speed and resource constraint. These features, however, makes it hard for approximation algorithms to provide absolute solution hence they provide approximate solutions with error bounds. Other tools are used with these algorithms to adapt the available resources. Approximation algorithms have been used in association analysis of streaming data (Deng, 2007).

Algorithm Output Granularity: This is the first resource-aware approach to data analysis, making it particularly applicable for data stream mining, especially one with concept drift (Gaber, Zaslavsky, and Krishnaswamy, 2004). The data analysis is performed on a resource constraint device that generates or receives streams of information. This approach has been successfully used in clustering, classification and association analysis (Gaber, Krishnaswamy, and Zaslavsky, 2005). The first part of the algorithm output granularity approach is the mining stage. Then the algorithm is adapted to the resources and streaming rate. The third and last stage involves merging the generated knowledge structures when running out of memory.

2.3.3 Discussion

In this section, we explained the methodologies employed by a data stream mining system. As we explained, these generally falls into two categories, based on the data to be processed, or the specific task. These methodologies are explained with some of their usage scenario. Table 2.2 provide us with a summary of these methods, with their advantages and disadvantages.

Table 2.2: Summary of Data Stream Methods (Han, Kamber, and Pei, 2011)

Technique	Description	Advantages	Disadvantages
Data-based Methods			
Sampling	Choosing a data subset for analysis	Error Bounds Guaranteed	Poor for anomaly detection
Load Shedding	Ignoring a chunk of data	Efficient for queries	Very poor for anomaly detection
Sketching	Random projection on feature set	Extremely Efficient	May ignore Relevant features
Synopsis Structure	Quick Transformation	Analysis Task Independent	Not sufficient for very fast stream
Aggregation	Compiling summary statistics	Analysis Task Independent	May ignore Relevant features
Task-based Methods			
Sliding Window	Algorithms with Error Bounds	Efficient	Resource adaptivity with data rates not always possible
Approximation Algorithm	Analyzing most recent streams	General	Ignores part of stream
Algorithm Output Granularity	Highly Resource aware technique with memory and fluctuating data rates	General	Cost overhead of resource aware component

The next section introduces the concept to represent changes in class labels in streaming data.

2.4 Concept Drift

Concept drift refers to a change in data distribution over time. This causes mismatch in the training and test dataset and is particularly a non stationary learning problem. Concept drift are generally classified into four types, as shown in figure 2.4. An overview of concept drift is provided here. For simplification, we will restrict the number of data chunks over time to $C1$ and $C2$ (Žliobaitė, 2010).

- *Sudden drift:* This represents the simplest pattern of change. It means that at time t, a chunk of data, $C1$, is suddenly replaced by another chunk, $C2$.
- *Gradual drift:* This kind of drift come in two types:
  - The first type of gradual drift refers to a period when both chunks are active and as time goes, the probability of sampling from $C1$ decreases, while the

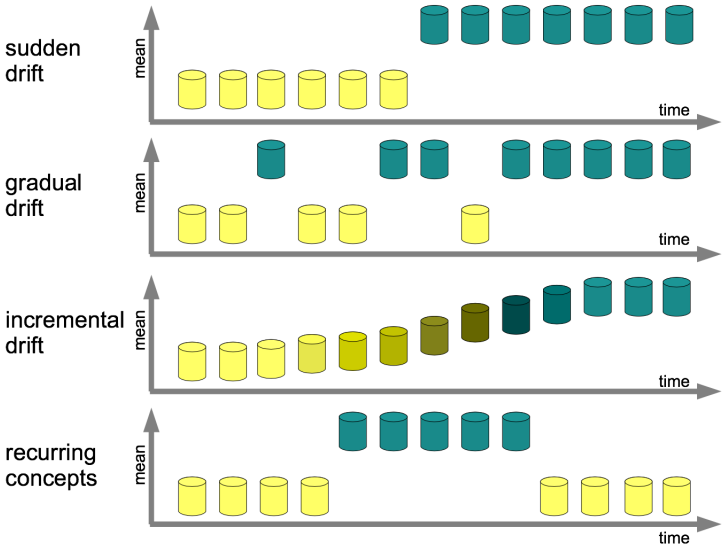


Figure 2.4: Illustration of Types of Concept Drift (Žliobaitė, 2010)

probability of sampling from $C2$ increases. Note that at the beginning of this type of gradual drift, more instances from $C1$ are visible, as instances from $C2$ might be easily mixed up with random noise. This type is generally the one referred to when talking about gradual drift.

  - The second type of gradual drift includes more than two chunks, however, the difference between them is very small, and the drift is noticed only when looking at a longer time period. This type of gradual drift is referred to as *incremental* or *stepwise* drift.

- *Reoccurring context drift:* This refers to a drift in which previously active concept reappears after some time. In this type of drift, it is not certainly periodic, as it is not clear when the source might reappear.

2.4.1 Learner Adaptivity and Model Selection

Adaptivity refers to the ability of a system to adapt its behaviour according to changes in its environment. This is particularly necessary in a stream with concept drift. The main areas of adaptivity in a data stream framework are in the following: The base learner; learner parameterization; training set formation (set selection, set manipulation and feature set manipulation); and Rules for learners or models (Žliobaitė, 2010).

In this study, our focus will be on the adaptivity strategy that is based on training set selection. This is due to the fact that our objective is to be able to select the most informative examples in the data stream to learn from. This is further divided

into *windowing* (selecting training instances in consecutive time) and *instance selection* (sequential in time instances are selected as a training set) (Žliobaitė, 2010). Windowing is preferred for sudden drift, while instance selection is preferred for gradual and reoccurring context drift. The model selection and evaluation strongly depends on the assumption about the adaptivity strategy on which the learner will be applied.

2.4.2 Detecting and Handling Concept Drift

In this section, we will briefly review related work studying concept drift. This will give a general overview in relation to the approach proposed in this thesis. In essence, there are two types of approaches, namely learners that evolve and techniques that are triggered when concept drift occurs.

Evolving Learners

Evolving learners employ change detection mechanism as a tool to reduce computational complexity. There are four main groups of these kinds of techniques and they are as follows (Žliobaitė, 2010):

- *Adaptive Ensembles* – This is a classifier ensemble method that combines or selects classification outputs of several models to get a final decision. The combination or selection rules are called fusion rules and they are used to achieve adaptivity by assigning weights to individual models at each instance.
- *Instance weighting* – The learner can consist of one algorithm or an ensemble, but the adaptivity is achieved by a systematic training set formation. Ideas from boosting is usually used to give more attention to misclassified instances.
- *Feature Space* – This manipulates feature space to achieve adaptivity i.e. new features are added to the training instance, either by transfer learning using information from past model performance, or using time.
- *Base model specific* – In this group of evolving learning algorithm, adaptivity is achieved by managing specific model design or parameter.

Learners with triggers

These groups of methods use triggers to determine how the models or sampling should be changed at a given time. Here, a trigger refers to the detection of a potential drift. The main groups are as follows (Žliobaitė, 2010):

- *Change detectors* – This is a trigger technique, and it is related to sudden drift. The method may be based on raw data, a learner parameter, or the output of the learners. The detection methods usually cut the training window at change point, although the window might be different in some cases.
- *Training windows* – Some learners in this group uses heuristics related to error monitoring, to determine the training window size, by using a look-up table principles. In this setting, there is an action for each possible trigger value. Others uses base learning specific methods or historical accuracy to determining the window size.
- *Adaptive sampling* – The previously listed trigger methods learn during the training windows, or by using instance selection for testing unlabelled incoming instances. On the other hand, in adaptive sampling, a new training set is selected based on the relationship between the testing instance, and a predefined or historic training instance.

2.4.3 Discussion

In this section, we explained the meaning of concept drift, and its different types. In real-world scenarios of data streams, it is almost impossible to have a stream without concept drift because of the constantly changing nature of data. It is, therefore, important to be able to deal with it by adapting our learning process to its occurrence. The two groups of learning methods that deals with concept drift were presented. In this thesis, we employ a combination of adaptive ensemble and the feature space method, as an evolving learner technique to handle concept drift. Instead of using fusion rules, we use the memory information from the past model as a new feature. And also, instead of assigning weights to each model in the ensemble, we adapt the size of the ensemble.

In the next section, the utility factor of data stream mining is explained, before looking at data stream mining challenges and applications in subsequent sections.

2.5 Utility and Efficiency Consideration In Data Streams

There has been extensive research about improving learning accuracy in data stream mining, and indeed, the general field of data mining. There is, however, relatively fewer work on the economic *utility* considerations. In this context, utility has to do with ”factors related to acquiring data, building models, and applying models” (UBDM, 2006). While Utility-Based Data Mining (UBDM) also covers other topics relating to utility in data

mining such as cost-sensitive learning, our focus in this research is on maximizing the economic factors in data stream mining algorithms.

Coincidentally, the term *cost-sensitive adaptation* was recently introduced by Žliobaite, Budka, and Stahl (2015). This should not be mistaken with cost-sensitive learning. While cost-sensitive learning considers different kinds of errors at the model level, cost-sensitive adaptation considers the system level cost of updating the model due to the evolving nature of data streams. The above paper highlighted four adaptation requirements in data stream mining, namely one-time process of examples; memory limitation; limited time; and having the model ready to predict at any time. A conclusion was also made that a data stream mining algorithm should be able to process incoming data at a linear scale in terms of time; use limited memory; and execute adaptation if expected utility is sufficient (Žliobaite, Budka, and Stahl, 2015).

Classification cost-sensitive learning and time are intrinsic factors in UBDM that have received more attention than extrinsic factors such as the cost of building and applying models (Zadrozny, Weiss, and Saar-Tsechansky, 2006; UBDM, 2006, 2005). In the advent of Big Data with finitely available resources, these factors requires more priority than they currently have. They enable us to take profitability into consideration in designing energy-efficient learning algorithms, while still considering conventional assessment measures like accuracy and f-measure. We will discuss more on cost-sensitive learning, accuracy and f-measure in Chapter 5 when we discuss performance evaluation.

Utility considerations such as the physical memory or computing power utilized by the algorithm in the learning process are examples of these extrinsic factors. The overall memory utilization by the algorithm vis-a-vis the time taken during the learning process is considered in this approach. The computing cost becomes more important when we consider that most cloud services bill using Random Access Memory (RAM) utilized per hour of using the service. Google Cloud Platform and Amazon Elastic Compute Cloud (Amazon EC2) are examples of cloud services that takes this into consideration in their billing processes (Google, 2014; Amazon, 2014). RAM-hour was introduced as an evaluation measure in (Bifet, Holmes, Pfahringer, and Frank, 2010d). Žliobaite, Budka, and Stahl (2015) proposed a method for quantifying the gain in performance for each resource invested using Return on Investment (ROI). The ROI on RAM-hour over time between two period of adaptation $\mathcal{T}$ and $\mathcal{T}+1$ is given as:

$$ROI_{\mathcal{T}} = \frac{\gamma_{\mathcal{T}}}{\psi_{\mathcal{T}}} \tag{2.1}$$

Where $\gamma_{\mathcal{T}}$ is the change in prediction accuracy (or error) between adaptation, called the gain of the adaptation; and $\psi_{\mathcal{T}}$ is the overall adaptation cost. The adaptation cost $\psi_{\mathcal{T}}$ is approximated as the computational cost ($\psi_{\mathcal{T}}^{com}$), and it is measured in RAM-hour (Rh). The ROI can be used either as a performance statistics, algorithm effectiveness measure over time; or a comparison measure between an adaptive and a non adaptive model.

To compute the average ROI over all adaptation, a weighted average is given as:

$$\overline{ROI} = \frac{\sum_{i=1}^{\mathcal{T}'}(N_i \times ROI_i)}{\sum_{i=1}^{\mathcal{T}'} N_i} = \frac{1}{\sum_{i=1}^{\mathcal{T}'} N_i} \sum_{i=1}^{\mathcal{T}'} N_i \frac{\gamma_i}{\psi_i} \tag{2.2}$$

Where T' is the total number of adaptations and N_i is the number of samples after the last adaptation (Žliobaite, Budka, and Stahl, 2015).

The above equations enable us to consider the value gained by the adaptive approach employed, and allows us to make an informed decision considering the monetary implications. The Utility factor also affects the relative usefulness of the model to our system. In deciding how much resource to allocate for the adaptivity, we consider the available resources in our system, and a weighted importance to the expected results. Different adaptation strategies have been developed for different kinds of adaptation in data mining algorithms. With consideration to efficiency and resources, there are five possible strategies (Žliobaite, Budka, and Stahl, 2015):

- Fully Incremental: The model is updated using the last window and the current window;
- Summary Incremental: The model is updated using the last window, the current window and a summary of all the previous windows;
- Batch Incremental: A batch of data windows is kept, and updated in sequence;
- Summary Batch Incremental: Similar to the batch increment, but with a summary of all the previous data;
- Non-incremental: A new model is built from past observations, whenever required. This means that a significant size of data is held in memory at any given time, for the model to be built from when required.

With these cost considerations in mind, and also considering the relative utility of the model to be constructed, there is need to derive a reasonable compromise. In this research, we have employed the fully incremental approach. Our objective is to obtain a better algorithm with consideration to the available resources, for a stream with concept

drift. We focus on the ROI, while making efficiency considerations based on memory usage induced by concept drift. The aim is to develop a method that uses memory utilization in its learner adaptivity technique, when mining in a data stream with concept drift.

2.6 Challenges in Mining Data Streams

In the previous sections, we introduced data stream mining with various considerations and methods. In this section, we take a look at the various challenges associated with data stream mining. We intend to provide an encompassing overview of the research challenges associated with data stream mining. This varies from the characteristics of the data and its preprocessing to the evaluation of streaming algorithms (Krempl, Žliobaite, Brzeziński, Hüllermeier, Last, Lemaire, Noack, Shaker, Sievi, Spiliopoulou, and Stefanowski, 2014). These challenges are presented below.

2.6.1 Streaming Data

The inherent characteristics of volume, velocity and volatility of a data streams constitute the primary research challenges in data stream mining. Most of the various discussion in previous sections are towards overcoming these primary challenges, so much that it is seldom not categorized as a data stream mining challenge.

2.6.2 Security/Privacy

As with every other field of technology, privacy and confidentiality are very important in data stream mining, and they constitute the primary challenges associate with this field of study. Privacy preserving data mining technique has been in active research for many years and is still actively researched (Lindell and Pinkas, 2000; Malik, Ghazi, and Ali, 2012; Lu, Zhu, Liu, Liu, and Shao, 2014). Unlike in offline mining, where we can check the privacy concerns before releasing the model, in data stream mining, privacy considerations and measures has to be taken before hand, as the mining is done online. There has been research in this area (Wang and Liu, 2008), but not quite as much as desired, and no data mining or data stream mining security framework exist. The issue of data integrity also comes into consideration, because altering the data attribute in a bid for higher security is bound to affect the resulting model.

Two main concerns are identified in relation to privacy in data stream mining:

- Incompleteness of Information: Because data arrive at intervals in portions, the model is never finalized, hence it becomes difficult to judge the need for privacy before seeing all the data.
- Concept Drift: Because a data stream with concept drift evolves over time, fixed privacy rules may no longer hold after some time in this kind of data stream. This could also go the other way, i.e. a stream evolving to require privacy when it originally does not require one. Adaptive privacy preservation mechanism is therefore an interesting area for further research in such cases.

2.6.3 Streaming Data Management

Unlike conventional data mining, where there is the luxury of tuning the data as required before the mining processing, this is not always possible in data stream mining. This section takes a brief look at challenges associated with preprocessing and availability of streaming data.

- Preprocessing: While there are many procedures available for preprocessing offline data (García, Luengo, and Herrera, 2015), data stream preprocessing has not quite received so much attention. This can be attributed to the fact that manual preprocessing is not feasible in streaming data mining. Fully automated and autonomous methods are required in this case. It would also be necessary for this kind of preprocessing methods to be able to update themselves as required with evolving and new arriving data.
- Information Availability: There is the general assumption by most data mining algorithms that the information related to the data is complete and ready to be processed. This is not always the case in data stream. Challenges like how to address the unpredictability of missing value frequency, how best to feed the stream and how to trade off speed and statistical accuracy are some of the key points to be dealt with (Nelwamondo and Marwala, 2008). Skewness in data distribution (He and Ma, 2013), latency (Krempl, 2011) and cost sensitivity issues (Spiliopoulou and Krempl, 2013) are also areas of challenges being actively researched.

2.6.4 Algorithms Evaluation

Because the characteristics of a data stream is significantly different than static data, most of the current evaluation criteria does not apply for these environments. factors such as

concept drift, latency, cost e.t.c. makes most existing evaluation criteria insufficient. New methods has been proposed for evaluating stream accuracy in both normal stream and stream with concept drift with one type of change (Gama, Sebastião, and Rodrigues, 2013; Bifet, Read, Žliobaitė, Pfahringer, and Holmes, 2013), but not much has been done in a stream involving many types of changes (Brzezinski and Stefanowski, 2014b). More advanced tools for visualizing changes in algorithm prediction with time are also desired.

2.6.5 Legacy Systems

Unlike in controlled research environment where we have the luxury of having a distinct and controllable data source, in most real life applications this is not the case. Data is generated from multiple sources, of varying time of origin and technology. Considering these data sources still provide huge and useful data, there might be need to analyse using real time streaming approaches. Given that it is not always possible to change existing infrastructure because of this, it becomes imperative to efficiently adapt stream mining techniques to these systems.

A recent attempt to this end is the use of data stream mining to monitor the International Space Station (ISS) Columbus Failure Management System (Noack, Luedtke, Schmitt, Noack, Schaumlöffel, Hauke, Stamminger, and Frisk, 2012). Reliability of the system is a key consideration because the life of astronauts and success of the mission depends on it. Another crucial factor considered in that mission is the complexity, considering that the system consists of various legacy modules. As was discussed in section 2.6.3, challenges associated with streaming data availability also was an issue. Issues of latency and transmission interruption comes to play considering the system needs to communicate with the ground station in near real time.

2.6.6 Model Construction

Simplicity of the model is also another area of active research in data stream mining. Because of the many varying factors in a data stream, it is very tempting, to parameterize the system in the name of adding features for various tunable settings. Simplicity should be the feature. Models that are constructed from a wild range range of parameters becomes unreliable and impractical for data stream mining because it is difficult for too many parameters to keep up with the evolving nature of data streams. Research work on tuning these model parameters has been done in offline stream (Guyon, Saffari, Dror, and Cawley, 2010) but not so much in stream mining.

Constructing model that could work for both big and fast data is still an area of challenge. This area of research is now actively talked about and some research work has started showing up in that direction (Zhang, Sow, Turaga, and van der Schaar, 2014), but not quite as much as desired. Because of resource constraints, consideration should also been given to optimizing memory performance, self tuning and auto adaptivity in designing algorithms for streaming data. Methodologies is lacking for optimizing such factors in data stream mining.

2.6.7 Entity Stream Mining

In conventional data stream mining, mining algorithms learn over a single stream of arriving entities or records. Each entity is all encompassing in the information it contains, and there is no assumption of any adjoining information for individual records arriving at a later time. This is quite different in entity stream mining in which stream entities are linked to instances from another stream. This discrete stream can be in the past, current or further point in time. This is related to relational databases, where records are linked together with foreign keys, except that they are not static.

An example of this is will be the arrival of patients in an hospital. Patients come and go at different times, and there are multiple pieces of information associated with each patient at different times. In this scenario, the entity mining is therefore that of creating a model that incorporate the adjoining information irrespective of when they appear in the mining process. The unsupervised clustering task is that of adapting the cluster over time as new information are available with consideration to the speed of availability. Similarly, the supervised classification task involves learning and adapting the classifier as new information become available.

The challenge becomes that of aggregating this discrete information to its corresponding record, as well as prediction. The fact that distinct entities appear and reappear at different time also pose a learning challenge in this kind of data stream. This is because, at the point of reappearance, there might be new information associated with that entity, linking the entity to a conceptually different instance. This phenomena is referred to as *entity drift*, quite different, but similar to concept drift. There has been active research in this area (Spiliopoulou and Krempl, 2013), but it still remain a major challenge because of the complexity.

Event Data Analysis

Learning from the occurrence of discrete events, can also be seen as a form of entity stream mining, considering that an event can be seen as a degenerate instance consisting of a single value. Events can be produced by a single or multiple sources with varying features. Events are not often talked about in data stream mining, even though they are studied through *Event History Analysis (EHA)* in static mining environment. Statistical methods such as survival analysis and hazard rates are used to approximate the likelihood of occurrence of an event in this method (Cox and Oakes, 1984).

These statistical model can also be used in similar ways for streaming data mining, however, the fact that the model may change over time introduces another level of complexity. Dealing with this kind of model change is obviously a challenge for entity analysis in data stream. There has been work in combining EHA with other methods for event detection (Sakaki, Okazaki, and Matsuo, 2013), there is still a long way to go in this area of research considering the need for real-time action in some events like intrusion detection.

2.7 Applications of Mining Data Streams with Concept Drift

In this section, we discuss the real-world applications of data stream mining, in streams with natural occurrence of concept drift. These problems are relevant in both supervised and unsupervised learning process. According to Žliobaitė (2010), these applications are generally categorized into four namely *Monitoring and Control, Personal Assistance and Information, Decision Making*, and *Artificial Intelligence and Robotics.*

2.7.1 Monitoring and Control

Monitoring and control is usually an unsupervised learning problem that detects irregular behaviour. This is typically a problem of analysing big and fast data, with time sensitivity. The monitoring task could be prevention against adversary actions. Examples of these kind of problems are intrusion detection in **network security** (Day, 2013) and fraud detection in **telecommunication** (Hilas, 2009). A combination of supervised and unsupervised learning techniques can also be combined, as is used for fraud detection (Bolton and Hand, 2002) in **finance**. In all of these scenarios, the time to response is very crucial.

The monitoring could also be for management. This kind of application can be seen in **transportation** for traffic management (Crespo and Weber, 2005), **positioning systems** (Liao, Patterson, Fox, and Kautz, 2007) or for **industrial monitoring** (Bakker, Pechenizkiy, Žliobaitė, Ivannikov, and Kärkkäinen, 2009). In these scenarios, change in pattern is required to be picked up with as much accuracy as possible, so as not to disrupt the flow of other activities.

2.7.2 Personal Assistance and Information

These applications are mainly concerned with the flow and organization of information. These could be either user specific information or information related to a business. Concern in these sort of applications is usually not very critical, and the class label is categorized as 'soft'. Personal assistance applications deals with personalizing information flow (Gauch, Speretta, Chandramouli, and Micarelli, 2007), referred to as *information filtering*. Applications related to **web personalization and dynamics** (Scanlan et al., 2008), **textual data** (Lebanon and Zhao, 2008) and **spam filtering** are also relevant areas in stream mining.

Profiling customer aggregated data is also another application of stream mining that enables us segment customers according to interest. This is particularly useful for something like direct marketing where there is a requirement to classify different customers based on product or service preference. Social network analysis is also been employed for this task and it was observed that interest evolves apart for multiple users (Lathia, Hailes, and Capra, 2008).

Handling data distribution over a long period of time is another application of stream mining to information. The task of **Document organization** is to extract meaningful structures from emails, news, or document stream, into topics. A significant project in this area was the organization and analysis of 120 years (1881 - 1999) of scientific topics articles of science magazine, showing emergence, peak and decline of various topics (Blei and Lafferty, 2006).

In **Economics**, mining streams with concept drift is applicable in making macro-economic forecasts (Giacomini and Rossi, 2009). It is also applicable in **software project management** where precise engineering and logic becomes inaccurate if concept drift is not considered in the planning phase (Ekanayake, Tappolet, Gall, and Bernstein, 2009).

2.7.3 Decision Making

Decision making in **Finance** for services such as credit card score or bankruptcy decisions also employs data stream mining, because of the variety of factors that come into play. While it is easy to see these problems as stationary problems, concept drift is closely related to other factors not considered in the original model. **Biomedical** applications are also subject of concept drift due to the mutating and evolutionary resistance of micro-organism to antibiotics. Concept drift applies to adaptivity to changes caused by human demographic (Kukar, 2003), discovery of emerging resistance and nosocomial infections in hospitals (Jermaine, Ranka, and archibald, 2008), as well as biometric authentication (Poh, Wong, Kittler, and Roli, 2009).

2.7.4 Artificial Intelligence and Robotics

In Artificial Intelligence (AI), the AI object learns to interact with a dynamic environment using adaptive techniques of data stream mining. *Ubiquitous Knowledge Discovery (UKD)* has to do with a distributed system, operating in an unstable and complex environment. Stanley, the robot that won the Defense Advanced Research Projects Agency (DARPA) grand challenge in 2006 is an example of a robot having to deal with the complexities of UKD using data stream mining with concept drift techniques (Thrun, Montemerlo, Dahlkamp, Stavens, Aron, Diebel, Fong, Gale, Halpenny, Hoffmann, Lau, Oakley, Palatucci, Pratt, Stang, Strohband, Dupont, Jendrossek, Koelen, Markey, Rummel, van Niekerk, Jensen, Alessandrini, Bradski, Davies, Ettinger, Kaehler, Nefian, and Mahoney, 2006).

In the current era of the Internet of Things (IoT), adaptability is also a requirement hence the employment of adaptive data stream mining (Gubbi, Buyya, Marusic, and Palaniswami, 2013). Virtual reality is another fast growing area of technological advancement, requiring adaptive stream mining since it is mostly used for fast changing applications like computer games and flight simulations.

2.8 Summary

In this chapter, we reviewed the concept of data stream mining, starting from its origin in data mining. We analyzed the various taxonomies of data mining and data stream mining based on techniques, task and streaming method. An important terminology called concept drift was also visited in section 2.4, where we explained concept drift and the methods for handling it.

This chapter also showed the various challenges associated with mining data streams. We discussed the issues of security, data management, evaluation method, legacy systems, model construction, and entity stream mining. We ended this chapter by mentioning some applications of mining data stream with concept drift in real-world. In the next chapter, we consider various data mining algorithms, specifically those that apply to our current work. We will also discuss metalearning and the different ensemble mining methods, as these will form the bedrock of our thesis methodology.

CHAPTER 3

Metalearning and Model Selection

In this chapter, we shall be looking at state-of-the-art algorithms in data stream mining, explaining their foundation and processes. We start by explaining what metalearning is and how it matters to our research. Some metalearning algorithms will be discussed in section 3.2. We will explain the model selection procedure in the Bagging and Boosting algorithms as these will form the basis of the online metalearning methods discussed in section 3.4. A technique that works well for detecting changes in data streams is ADWIN and shall be explained in 3.3. This is particularly important in a stream with concept drift because it allows learning algorithms to react to the changes detected.

3.1 Metalearning

Metalearning is the study of methods that exploit knowledge about knowledge, called metaknowledge, to obtain efficient models and solutions by adapting machine learning and data mining processes (Brazdil, Giraud-Carrier, Soares, and Vilalta, 2009). From the Greek work *"meta"*, which literally mean *"after"* or *"beyond"*, we can conclude that metalearning is the knowledge derived *"after"* learning, or in this case, after applying a machine learning algorithm. Characteristics obtained during the metalearning process are referred to as *metafeature*. A common usage of metalearning is in the ensemble process as will be explained in section 3.2. Many application of metalearning in data stream mining has been in the area of algorithm selection, algorithm combination, process management amongst others (Rossi, De Carvalho, Soares, and De Souza, 2014; Bie, Jin, Chen, Xu, and

Huang, 2007).

Metalearning is also used in *Active Learning*. Active learning employs a semi-supervised approach, in which the learning algorithm is able to query the user (called *oracle*) for labels (Settles, 2009). Alabdulrahman (2014) combines active learning with metalearning in a bid to improve classification accuracy. In this thesis however, we employ metalearning to improve our classification accuracy by intelligently adapting the size of the *ensemble* according to changes in its environment. This ability is referred to as adaptivity and we have previously discussed about it in section 2.4.1. In this chapter, we shall explain some ensemble methods and how they work.

3.2 Model Selection in Ensemble Methods

A useful concept to keep in mind in the discussion of data stream mining is *ensemble learning*, which is the process of learning using ensemble methods. The ensemble method is the bedrock of metalearning in data mining. It is the process of building better predictive model by integrating multiple models. Ensemble methods use a combination of models to obtain better predictive performance by reducing the chances of misclassification of a single model, creating an improved composite model.

The implicit assumption of model selection in metalearning in that there is an optimal learning algorithm for each task (Brazdil, Giraud-Carrier, Soares, and Vilalta, 2009). While this holds true in most cases, there is the risk of eventually settling for a suboptimal algorithm by replacing a very good model with a combined average of sub-par models.

This combination of models could be from one learner or multiple learners (de Souza and Matwin, 2013). The learners that makes up this combination of models are referred to as *base learners* or *base-level algorithms*. The ensemble method itself is a supervised learning method because it can be trained and used for predictions (Han, Kamber, and Pei, 2011). Some ensemble methods are Bootstrap Aggregating (Bagging), Boosting and Stacking. Algorithms that make use of the ensemble methods are referred to as *meta learners* or *meta-level algorithms* (section 3.4).

Figure 3.1 (Kotu and Deshpande, 2015) describes the ensemble method process. The value of n is the size of the ensemble i.e. n-number of models are constructs from the base learning algorithm(s) that correspondent to the allowed size. This value is user assigned, but it is common practice to assign a value of 10. This value directly affects our classification accuracy as we shall see in Chapter 4. The ensemble output function $f(x)$ is based on the ensemble method used.

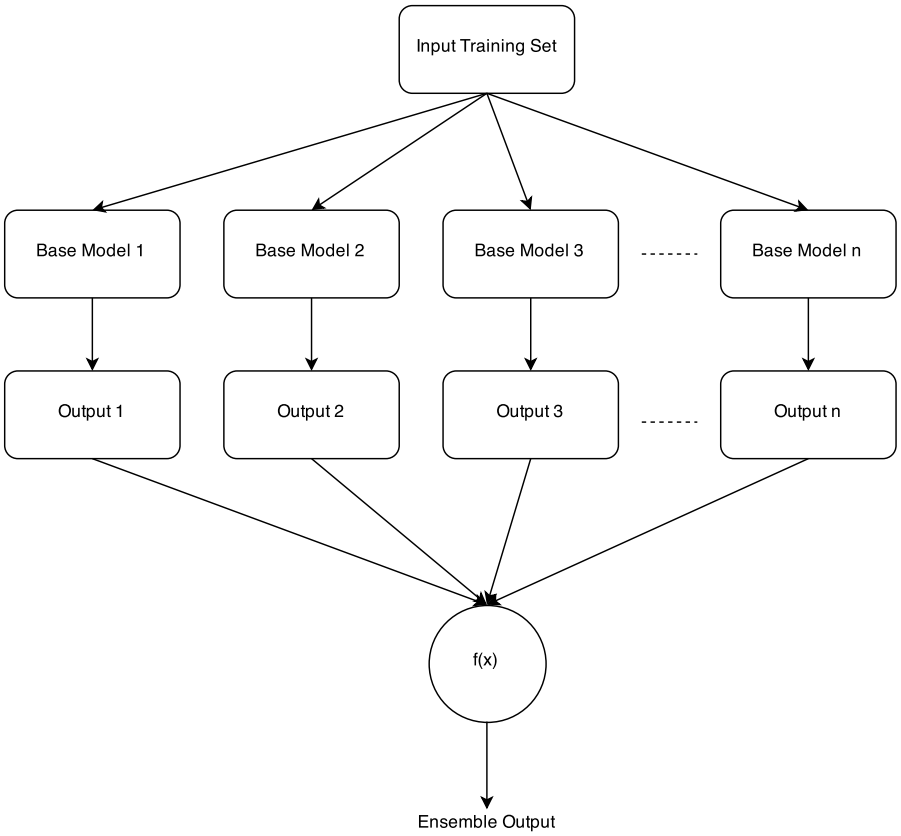


Figure 3.1: Ensemble Methods

In the next section, we will discuss an important concept drift detection technique called Adaptive Windowing (ADWIN). Recall that in section 2.4.1, we explained that one of the methods of adapting to concept drift is by using change detectors. ADWIN is a change detector that is used in combination with other algorithms, for improving learning accuracy in a stream with concept drift. The metalearning algorithms considered in this thesis will be illustrated in subsequent sections.

3.3 ADWIN Drift Detector

ADWIN is a change detector that was implemented as a way of estimating changes in a data stream with concept drift using an adaptive sliding window model (Bifet and Gavaldà, 2007). In this approach, whenever the difference between the mean values of the elements in two sliding windows is more than a threshold value, the older sliding window is dropped. It has a confidence bound δ that indicates how confident the output is.

The window size(s) are not maintained explicitly, as ADWIN uses a variant of exponential histogram technique as shown in figure 3.2. It keeps an approximate number of $1's$

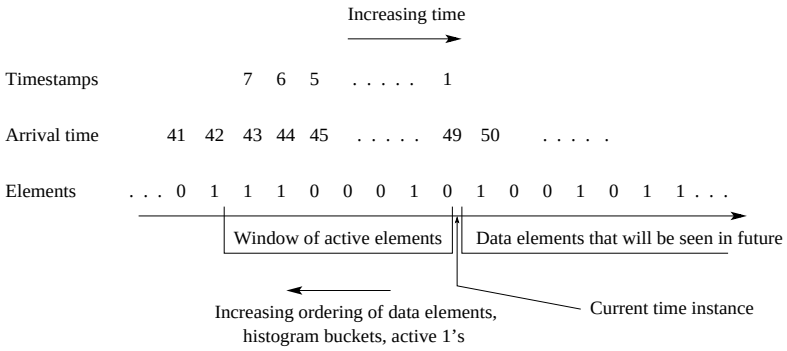


Figure 3.2: Exponential Histogram Illustration (Datar, Gionis, Indyk, and Motwani, 2002)

as the *bucket size* in a sliding window W using $O(M \cdot log(W/M))$ memory and $O(logW)$ processing time.

Algorithm 1 ADWIN

Input: empty list of buckets, W

Initialize W
Initialize WIDTH, VARIANCE and TOTAL
for all $t > 0$ **do**
    SETINPUT(x_t, W)
    output $\hat{\mu}_W$ as TOTAL/WIDTH and ChangeAlarm

SETINPUT(item e, List W)
INSERTELEMENT(e, W)
repeat
    DELETEELEMENT(W)
until $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \epsilon_{cut}$ holds
    for every split of W into $W = W_0 \cdot W_1$

INSERTELEMENT(item e, List W)
create a new bucket b with content e and capacity 1
$W \leftarrow W \cup \{b\}$(i.e., add e to the head of W)
update WIDTH, VARIANCE and TOTAL
COMPRESSBUCKETS(W)

DELETEELEMENT(List W)
update WIDTH, VARIANCE and TOTAL
ChangeAlarm $\leftarrow$ **true**

COMPRESSBUCKETS(List W)
remove a bucket from tail of List W
Traverse the list of buckets in increasing order
    do If there are more than M buckets of the same capacity
        do merge buckets
            COMPRESSBUCKETS(sub-list of W not traversed)

Output: $\hat{\mu}_W$

In the algorithm above, ϵ_{cut} is a threshold value, and it is computed as:

$$\epsilon_{cut} = \sqrt{\frac{2}{m} \cdot \sigma_W^2 \cdot \ln \frac{2}{\delta'}} + \frac{2}{3m} \ln \frac{2}{\delta'} \tag{3.1}$$

Where n_0 and n_1 are the lengths of the splits W_0 and W_1; $n = n_0 + n_1$; $m = \frac{1}{1/n_0 + 1/n_1}$; $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ are the average values of W_0 and W_1; δ is a confidence value; $\delta' = \delta/(\ln n)$; and σ_W is the observed variance of elements in W (Bifet, Holmes, Kirkby, and Pfahringer, 2011).

The ADWIN algorithm takes a confidence value δ, and a sequence of real values $x_1, x_2, ..., x_t, ...$ as input; where x_t is the value at time t according to some distribution D_t. Here, μ_t and σ_t^2 are the mean and variance at t. The value of x_t is assumed to be, or can be normalized to $0 \leq x_t \leq 1$. A sliding window W of length n containing the most recent read value of x_i is kept, with observed average $\hat{\mu_W}$ and unknown average μ_W for $t \in W$. A parameter M with size 2^i is used to control the memory used, and the window size of approximately $M \cdot log(W/M)$ buckets. W_0 & W_1 with corresponding values of μ_W are used to represent the old and current windows respectively. We shall be using ADWIN as a change detector in the metalearning algorithms used in our comparative study.

3.4 Meta-level Algorithms

3.4.1 Bagging

Bagging stands for Bootstrap Aggregating (Breiman, 1996), and it involves having the models in an ensemble vote with equal weight. Though it is frequently used in solving classification problems, it can also be used in solving regression problems.

Algorithm 2 Bagging

Input: T is the training set with examples x;
Y is the finite set of target class values y;
HT is the base learning algorithm (Hoeffding trees);
M is the number of models in the ensemble, each with examples drawn from T;
q is the classification instance;
δ is the generalized Kronecker function: $\delta(a, b) := \begin{cases} 1 \text{ if } a = b \\ 0 \text{ if } a \neq b \end{cases}$

1: **for all** $m = 1$ to M **do**
2: S_m = random sample of size d drawn from T, with replacement
3: h_m = model induced by HT from S_m
4: **for all** new q **do**

Output: hypothesis: $h_{fin}(x) = argmax_{y \in Y} \sum_{m=1}^{M} \delta(y, h_m(x))$

Using an example with Hoeffding tree (HT) as base learner, and a dataset T, Bagging

draws a total of M samples with replacement, and induces models from each of them independently. That is, M number of models h are induced independently. For a new classification problem, the most predicted class amongst the models is assigned to the new instance. This value is obtained by a combined vote of all the classes within the different models (Brazdil, Giraud-Carrier, Soares, and Vilalta, 2009).

Bagging works best with an unstable base learner, because each independently created model will be very different in response to minor changes (Bifet, Holmes, Kirkby, and Pfahringer, 2011). This makes it particularly useful in a data stream with concept drift, because there is a very high chance that the samples the models are built on are very different. Bagging reduces the bias and variance by averaging across all models into a single result.

3.4.2 OzaBag

Recall that in online learning, an algorithm learns one instance at a time, which makes it ideal for learning from a streaming environment, where instance arrive one after the other. Oza and Russell's Online Bootstrap Aggregating (OzaBag) is an online version of Bagging (Oza and Russell, 2001). For a dataset N, while Bagging algorithm assumes a binomial distribution because of the finite size of the dataset in memory, OzaBag assumes a Poisson distribution because of the continuous nature ($N \to \infty$) of the dataset in a data stream. The process samples bootstrap replicates from the data in a data stream, using the Poisson probability distribution.

Algorithm 3 OzaBag

Oza and Russell's Online Bootstrap Aggregating

Input: N is an evolving data stream ($N \to \infty$);
T is the training set drawn from N with examples x;
Y is the finite set of target class values y;
HT is the base learning algorithm (Hoeffding trees);
M is the number of models in the ensemble, each with examples drawn from T;
δ is the generalized Kronecker function: $\delta(a, b) := \begin{cases} 1 \text{ if } a = b \\ 0 \text{ if } a \neq b \end{cases}$

1: **for all** $m = 1$ to M **do**
2: S_m = random sample of size d drawn from T, with replacement
3: h_m = model induced by HT from S_m
4: Set $k = Poisson(1)$
5: **for all** $n = 1, 2, ..., k$ **do**
6: Update h_m with current examples $S_m \in T$

Output: hypothesis: $h_{fin}(x) = argmax_{y \in Y} \sum_{m=1}^{M} \delta(y, h_m(x))$

OzaBagADWIN

OzaBagADWIN was introduced as an improvement to the OzaBag algorithm, with two primary modifications: (1) increasing the rate of resampling, and (2) using output detection codes (Bifet and Gavaldà, 2007). A large Poisson distribution parameter (λ) was used to increase the resampling weight. While the optimal value of λ may not be the same for different datasets, by increasing its value, the diversity of the weight is increased. This effectively modifies the input space of the classifiers inside the ensemble.

Algorithm 4 OzaBagADWIN

Input: N is an evolving data stream ($N \to \infty$);
T is the training set drawn from N with examples x;
Y is the finite set of target class values y;
HT is the base learning algorithm (Hoeffding trees);
M is the number of models in the ensemble, each with examples drawn from T;
λ is a Poisson distribution parameter;
δ is the generalized Kronecker function: $\delta(a,b) := \begin{cases} 1 \text{ if } a = b \\ 0 \text{ if } a \neq b \end{cases}$

1: **for all** training examples (x, y) **do**
2: **for all** $m = 1$ to M **do**
3: S_m = random sample of size d drawn from T, with replacement
4: h_m = model induced by HT from S_m
5: Compute random binary value $\mu_m(y)$
6: Set $w = Poisson(\lambda)$
7: Update h_m with current examples $S_m \in T$, with weight w and class $\mu_m(y)$
8: **if** ADWIN detects change in error of one of the models (h_i) **then**
9: Replace the model with highest error with a new model (h_n)

Output: hypothesis: $h_{fin}(x) = argmax_{y \in Y} \sum_{m=1}^{M} \delta(y, h_m(x))$

For the output detection, the classes assigned to each example are mapped to a new binary class $\{0, 1\}$ for each classifier m and class $y_i \in Y$. The random binary value $\mu_m(y) = \{0, 1\}$ is obtained in a uniform and independent way, with exactly half of the class mapped to each binary value. The class with the highest votes of $\mu_m(y)$ is made the output of the classifier. As can be seen in algorithm 4, $\mu_m(y)$ is used as "colouring", and its matrix is built for each classifier and class.

For every new instance, a random weight of Poisson(λ) is used to train the classifier. Recall that in section 1, we introduced ADWIN as a change detector. When ADWIN detects a change, the worse classifier of the ensemble of classifiers is replaced with a new one. The sum of the colouring/votes $\mu_m(y)$ for all the classifiers is computed for each class (y), and the class with the highest vote is outputted as the predicted class (Bifet, Holmes, Kirkby, and Pfahringer, 2011).

3.4.3 Boosting

Boosting was introduced to address the problem of improving the learning accuracy in the Probably Approximately Correct (PAC) framework (Schapire, 1990). The PAC model identifies unknown data based on randomly selected examples of the dataset. The concepts of *weak learnability* and *strong learnability*, was defined by Kearns and Valiant (1994). A learner is strong if its induced models performs accurately on most instances of the test dataset. It is considered weak if the performance is only slightly better than random guess. The equivalence of these two methods was proved by Schapire (1990).

Algorithm 5 Boosting

Adapting Boosting (AdaBoost.M1) by Freund and Schapire

Input: T is the training set with examples x;
Y is the finite set of target class values y;
HT is the base learning algorithm (Hoeffding trees);
M is the number of models in the ensemble, each with examples drawn from T;
q is the classification instance;
Z_i is a normalization constant, chosen so that D_{i+1} is a distribution;
ϵ_i is the error of the misclassified instance;
β_i is the weight assigned to the model

1: **for all** $k = 1$ to T **do**
2: $D_1(x_k) = \frac{1}{T}$
3: **for all** $i = 1$ to M **do**
4: h_i = model induced by HT from T with distribution D_i
5: $\epsilon_i = \sum_{k:h_i(x_k) \neq y_k} D_i(x_k)$
6: **if** $\epsilon_i > 0.5$ **then**
7: $M = i - 1$
8: Abort loop
9: $\beta_i = \frac{\epsilon_i}{1-\epsilon_i}$
10: **for all** $k = 1$ to T **do**
11: $D_{i+1}(x_k) = \frac{D_i(x_k)}{Z_i} \times \begin{cases} \beta_i \text{ if } h_i x_k = y_k \\ 1 \text{ otherwise} \end{cases}$
12: **for all** new q **do**

Output: hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_m^{h_m(x)=y} \log \frac{1}{\beta_m}$

Boosting works on the observation that finding weak models is easier than finding strong learners (Brazdil, Giraud-Carrier, Soares, and Vilalta, 2009). It continuously aims to force the weak learner to change its induced model by varying the distribution D_i over the training set T, as a function of previously generated models. The initial distribution D_i is uniform and each instance is assigned a constant weight. The first model is also induced. After each iteration, the weight assigned to the misclassified instances is increased allowing the next model to focus on them. This goes on until either a fixed number of

iterations are done, or the overall weight of the misclassified instances is more than 0.5. A weighted vote of these models is eventually used to assign the class of new instances.

Algorithm 5 shows a Boosting implementation for classification problems. A more complex variant of adaboost for regression problems is the AdaBoost-R (Freund and Schapire, 1997).

3.4.4 OzaBoost

Similarly to OzaBag, Oza and Russell's Online Boosting (OzaBoost) algorithm is an online version of Boosting, also by Oza and Russell (2001).

Algorithm 6 OzaBoost

Oza and Russell's Online Boosting.

Input: N is an evolving data stream ($N \rightarrow \infty$);
T is the training set drawn from N with examples x;
Y is the finite set of target class values y;
HT is the base learning algorithm (Hoeffding trees);
M is the number of models in the ensemble, each with examples drawn from T;
λ_x is the Poisson distribution parameter at the latest value of example x;
λ_m^{sc} is the sum of λ_x for all the values of x that were correctly classified by the base model $m \in M$;
λ_m^{sw} is the sum of λ_x for all the values of x that were misclassified by the base model $m \in M$;
ϵ_m is the error of the misclassified instance;
β_m is the weight assigned to the model

1: Initialize base models h_m for all $m \in \{1, 2, ..., M\}, \lambda_m^{sc} = 0, \lambda_m^{sw} = 0$
2: **for all** training examples **do**
3: Set "weight" of example $\lambda_x = 1$
4: **for** $m = 1, 2, ..., M$ **do**
5: Set $k = Poisson(\lambda_x)$
6: **for** $i = 1, 2, ..., k$ **do**
7: Update h_m with the current examples $x \in T$
8: **if** h_m correctly classifies the example **then**
9: $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda_x$
10: $\lambda_x \leftarrow \lambda_x \left(\frac{M}{2\lambda_m^{sc}}\right)$
11: **else**
12: $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda_x$
13: $\lambda_x \leftarrow \lambda_x \left(\frac{M}{2\lambda_m^{sw}}\right)$
14: $\epsilon_m = \frac{\lambda_m^{sw}}{\lambda_m^{sc} + \lambda_m^{sw}}$
15: $\beta_m = \epsilon_m / (1 - \epsilon_m)$

Output: hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_m^{h_m(x)=y} \log \frac{1}{\beta_m}$

As was done in OzaBag, a Poisson distribution (λ_x) was assumed in deciding the probability that a model is used for training. However, the Poisson distribution parameter, λ_x, associated with current example is increased before presenting to the next model

when a misclassification occurs in the base model; otherwise it is decreased. It was noted that the weighting procedure divides the total example weight into equal halves for the correctly classified and misclassified examples (Oza and Russell, 2001). As the ensemble gets more accurate however, the number of misclassification gets less compared to the correct classifications. The misclassified set, therefore, have more weight assigned to them than the correctly classified set (Bifet, Holmes, Kirkby, and Pfahringer, 2011).

To correct this, a weight update procedure was implemented to simulate the batch weight behaviour of a data stream. The value of λ_x is decreased or increased according to the Boosting weight of each model, as it passes through the models. The theory behind using a random Poisson is not as clear as was implemented in Bagging, but preliminary experimentations shows better performance (Bifet, Holmes, Kirkby, and Pfahringer, 2011), hence the justification. Online Bagging, however, yield better results for data streams (Bifet, Holmes, Pfahringer, and Gavaldà, 2009a). This can be explained considering the fact that models in Bagging are generated independently and can be replaced if no longer accurate. The sequential nature of Boosting prevents such replacement, hence the problem with evolving data (Bifet, Frank, Holmes, and Pfahringer, 2010a). OzaBoost with ADWIN (OzaBoostADWIN) is a variant of the OzaBoost algorithm that uses ADWIN as change detector and was implemented to address some of these shortcomings.

3.4.5 Discussion

In this section, we discussed two online ensemble methods introduced above. In some of these research works, Boosting have been shown to yield better accuracy when compared to Bagging, but it also tends to be more likely to over-fit the training data. From studies appearing in the literature (Oza and Russell, 2001; Oza and Russell; Bifet, Holmes, Pfahringer, Kirkby, and Gavaldà, 2009b; Wang and Pineau, 2013), Online Bagging seems to perform better than the Online Boosting methods, in terms of classification accuracy (Bifet, Holmes, and Pfahringer, 2010c).

After a series of comparison with several Bagging and Boosting algorithms, Khoshgoftaar, Hulse, and Napolitano (2011) showed that Bagging techniques generally outperforms Boosting for handling class imbalance in a noisy dataset. A similar comparison was done by Wang and Pineau (2013), using different Online Bagging and Boosting algorithms. It was showed that *"Bagging based algorithms achieve better performance than Boosting based algorithms in terms of consistency and Area Under the Curve (AUC) value"*. The exact reason for this is still unknown. Adding more random weight to all instances seems to

improve accuracy more than adding weight to misclassified instances.

Bagging is regarded as a general technique appropriate for most problems since (i) its performance is either similar or significantly better than a single classifier (but it does not hurt performance as Boosting may do); and (ii) it allows a parallelized development of it's members, while Boosting is a necessarily sequential approach (Alaiz-Rodríguez, 2008).

With regards to the base learner, Bagging reduces the variance of the base models without changing the bias. This helps when: (i) applied with an over-fitted base model; (ii) high dependency on actual training data (data streams); and (iii) improves performance for unstable classifiers which vary significantly with small changes in the dataset (i.e. concept drift).

For the reasons highlighted above, our focus in this thesis is on improving Online Bagging algorithms. Recall that in section 2.5, we discussed utility and efficiency concerns associated with data streams. We thus aim to improve on the ROI of existing implementation.

Further consideration should also be given to the robustness of the base-level algorithm to be used. While Online Bagging does not necessary help much when the base model is robust to the changes in the training data set (i.e. in cases where the algorithm is robust to concept drift), we ensure multiple level of efficiency when we have such a combination. Hoeffding trees (Domingos and Hulten, 2000) are state-of-the-art in classification for data streams. They perform prediction by choosing the majority class of each leaf. Another appealing feature of this algorithm is that it offers sound guarantee of performance (Bifet and Gavaldà, 2009). A more detailed discussion of Hoeffding trees is given in the next chapter, as we shall be using it as base learner in our experimentation.

3.5 Summary

The metalearning concept was introduced in this chapter, by way of showing two state-of-the-art ensemble methods for model selection. We described how the best learning model is averaged or weighted over multiple models. Some algorithms were also discussed, from ADWIN for change detection, to the different meta-level algorithms considered in this research. The online ensemble learning algorithms of Bagging and Boosting were shown in sections 3.2 and 3.4. We concluded this chapter by explaining why we decided to work with Online Bagging in this research. In the next chapter, we shall present the research methodology we used for the algorithms that this thesis work is based on.

Part II

ADAPTIVE ENSEMBLE SIZE

CHAPTER 4

Adaptive Ensemble Size (AES) Online Bagging

According to the "No Free Lunch" Theorem for Supervised Machine Learning (Wolpert, 1996), no algorithm performs better than others when their performances are averaged over all possible scenarios. It is generally known that we should not attempt to find the "best" algorithm, as the performance of one algorithm on a particular task might not be accurate to judge it by every possible task. Recall that in Chapter 2, we illustrated the utility factors of mining in a stream with concept drift. The primary objective of the experimentation in this thesis is to improve on the accuracy of classification results in a data stream with concept drift, with consideration to the ROI of the mining process.

In the previous hapter, we introduced two online mining algorithms, and we concluded on using Online Bagging as base for our improvements. Our conclusion was based on previous research work that have found that Online Bagging performs better than Online Boosting in terms of accuracy, in a stream with concept drift. In this chapter, we introduced an Adaptive Ensemble Size (AES) approach, which allows us to vary the size of the next ensemble, based on the memory utilization of the current ensemble.

While it may be desirable to test multiple base learner for this task, there is no strong indication that the choice of a base-learner will significantly improve performance or consistency of Online Ensemble algorithms (Wang and Pineau, 2013). For this reason, we choose the Hoeffding tree algorithm as base learner because of it's sound guarantee

of performance (Bifet and Gavaldà, 2009). We start this chapter by explaining how the Hoeffding tree algorithm works. This is followed by introducing the methodology for our approach in section 4.2 and we present our algorithms in section 4.3. We will also discuss the thought process that led us to this approach and the preliminary experimentations carried out.

We created this method as a result of observing the behavioural characteristics of a stream with concept drift. Recall that we showed the importance of the memory utilization to data stream mining in section 2.5. Towards finding a way to make improvements to the algorithms, we will compare the memory cost between a data stream with concept drift and one with a static stream in section 4.4. We will also show how the size of the ensemble affects the overall accuracy of the learning algorithm. Our observations are shown in sections 4.4.2 and 4.4.3. Our methodology builds on these observations to create our novel AES approach.

4.1 Hoeffding tree Algorithm

The Hoeffding tree technique is an example of a supervised classification algorithm using the Decision tree technique. A decision tree is in the form of a decision flowchart comprising of *nodes* and *branches*. The node can either be *internal*, denoting a test on an attribute, or the *leaf*, denoting a class label. The branches represents the outcome of the test. The topmost node, called *root* node, is the original hypothesis. Decision trees are used to separate datasets into their respective classes. These classes are usually binary (yes/no, 1/0), but Iterative Dichotomiser 3 (ID3), the variant of the decision tree technique supporting more than two categories was described by Quinlan (1986). Figure 4.1 (Quinlan, 1986) is an example of a simple decision tree to determine the probable weather condition. The internal nodes are represented with a rectangle, and the leaf node represented with an oval. 'P' (or Positive) and 'N' (or Negative) are the class labels.

Classic decision tree algorithms such as ID3, Classification And Regression Trees (CART) and C4.5 are inefficient for online data stream mining, because they generally assume that the complete training dataset is stored in memory. The Hoeffding tree algorithm was introduced to overcome this shortcoming. It is a frequently used benchmarking algorithm that has had success in data stream mining. It was first introduced by Domingos and Hulten (2000) and it was called Very Fast Decision Trees (VFDT). In their paper, the Hoeffding tree algorithm was used as the basis of the algorithm while VFDT introduces some enhancements in its implementation. VFDT is now simply referred to as the Hoeffding

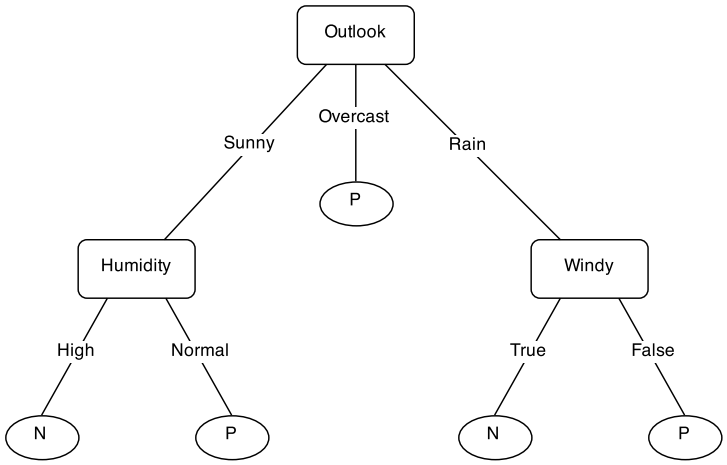


Figure 4.1: A Decision Tree

trees, and there have been several adaptations of the algorithm.

For example, the Hoeffding tree algorithm was also used to track web clicks and construct models to predict which web site is likely to be accessed next by a web user (Domingos and Hulten, 2000). In that experimentation, the learner was to read each example at most once, with very limited processing time. It was observed that to find the best attribute at any given node, it may be sufficient to train with only a small subset of the examples that pass through the node. Hence, given a data stream, the first window will be used to choose the root, and succeeding windows will be passed down to corresponding leaves to choose the appropriate attributes, and so on. The algorithm does this repeatedly, with the assumption that the data distribution is random and does not change over time. The problem of deciding how many examples are required at each node is solved by using a statistical method called the *Hoeffding bound* (ϵ), also known as additive Chernoff bound (Hoeffding, 1963). This bound states that with confidence $1 - \delta$ (where δ is a defined threshold), the random variable r, with range R, will not differ from an estimated mean ($\overline{r}$), after a number (n) of independent observations, by more than ϵ, where:

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{4.1}$$

The Hoeffding bound is independent of the probability distribution generating the observations. It is also more conservative that distribution dependent bounds, taking more observations to reach the same δ as ϵ (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten, 2009).

Consider a sequence of examples S, and $G(X_i)$ is the heuristics measure been used to

choose test attributes. The goal of the algorithm is to ensure that, with high probability, the attributes chosen using very small examples, n, are the same as using infinite examples. Maximizing G, let X_a, X_b be the attribute with the highest and second highest values of $\overline{G}$ respectively after n examples and $\Delta\overline{G} = \overline{G}(X_a) - \overline{G}(X_b) \geq 0$ is the difference between their heuristic values. Given a desired δ, the Hoeffding bound guarantees that X_a is the correct choice with probability $1 - \delta$ if n examples have passed through the node. If $\Delta\overline{G} \geq \epsilon$, i.e. if the observed $\Delta\overline{G}$ is greater than ϵ, then the Hoeffding bound guarantees that the true ΔG is at least $\Delta\overline{G}$, and is greater than 0 with probability $1 - \delta$. Therefore X_a is indeed the best attribute.

Algorithm 7 Hoeffding tree algorithm with VFDT Enhancements

Input: S is a sequence of examples;
X is a set of discrete attributes;
X_0 is a null attributes used for pre-pruning;
$G(.)$ is a split evaluation function;
δ is $1 - p$, where p is the desired probability of choosing the correct attribute at any give node

Let HT be a tree with a single leaf l_1(the root)
Let $X_1 = X \cup \{X_0\}$
Let $\overline{G}_1(X_0)$ be the $\overline{G}$ obtained by predicting the most frequent class in S.
for all class y_k **do**
  for all value x_{ij} of each attribute $X_i \in X$ **do**
    Let $n_{ijk}(l_i) = 0$
for all example (x, y_k) in S **do**
  Sort (x, y) into leaf l using HT
  for all x_{ij} in x such that $X_i \in X_l$ **do**
    Increment $n_{ijk}(l)$
  Label l with the majority class among the examples seen so far at l
  if $n_l \; mod \; n_{min} = 0$ **and** examples seen so far at l are not all of the same class **then**
    Compute $\overline{G}_l(X_i)$ for each attribute $X_i \in X_l - \{X_0\}$ using the counts $n_{ijk}(l)$
    Let X_a be the attribute with highest $\overline{G}_l$
    Let X_b be the attribute with second-highest $\overline{G}_l$
    Compute Hoeffding Bound $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n_l}}$
    if $X_a \neq X_0$ and $(\overline{G}_l(X_0) - \overline{G}_l(X_b) > \epsilon$ or $\epsilon < \tau)$ **then**
      Replace l by an internal node that splits on X_a
      for all branch of the split **do**
        Add a new leaf l_m, and let $X_m = X - \{X_a\}$
        Let $\overline{G}_m(X_0)$ be the $\overline{G}$ obtained by predicting most frequent class at l_m
        for all class y_k and each value x_{ij} of each attribute $X_i \in X_m - \{X_0\}$ **do**
          Let $n_{ijk}(l_m) = 0$

Output: Return decision tree, HT

The pseudocode for the Hoeffding tree algorithm, with the Very Fast Decision Trees enhancements is shown in algorithm 7 (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten, 2009). The count n_{ijk} are the sufficient examples needed for most heuristic

measures; X_0 is a null attribute used for pre-pruning. One of the major improvements introduced as part of the VFDT implementation is a user-specified threshold, τ. This is used to choose two or more attributes having very similar value of G. The VFDT decides if there is a tie and split on current value of X_a if $\Delta\overline{G} < \epsilon < \tau$. The VFDT also allows for the use of either the information gain or the Gini index as attribute evaluation method. The most significant improvement to the algorithm, timewise, is the recomputation of G. Instead of recomputing G for every example, considering the unlikelihood that a splitting decision been made during that window, VFDT allows the user to define a minimum number of new examples, n_{min}, that must accumulate before G is computed. This significantly improves the memory usage, and the speed spend on G computation by a factor of n_{min}, effectively making the algorithm itself much faster (Domingos and Hulten, 2000).

The primary advantage of this classifier is the fact that the nodes are incrementally being built, and we can use it to classify data during the building process. It is particularly preferred for high speed data streams, as the algorithm remains highly efficient; this was rare before its introduction. Independently, the Hoeffding tree algorithm does not perform so well in a stream with concept drift. It, however, performs well as a base learner in a metalearning algorithm (Bifet, Holmes, and Pfahringer, 2010c; Brzezinski and Stefanowski, 2014a).

4.2 Adaptive Ensemble Size Methodology

In this section, we explain the methodology used in the AES Online Bagging algorithm. Recall that in Chapter 2, we indicated that an ensemble learning algorithm requires one or more base-level algorithms to build models. We explained in Chapter 3, that ADWIN was introduced by Bifet and Gavaldà (2007) as a way for dealing with a stream with concept drift. It was later combined with OzaBag and OzaBoost (Bifet, Holmes, Pfahringer, Kirkby, and Gavaldà, 2009b) as a way of improving them in a stream with concept drift. We also explained that based on different research work, it has been shown that Online Bagging generally works better than Online Boosting in a stream with concept drift. The literature has also shown that Hoeffding tree is a good base learner that guarantees good performance, as was stated in the previous section.

Our goal is thus to combine these meta-level and base-level algorithms and improve on their accuracy, when mining in a stream with concept drift. To this end, we observe the physical memory consumption of these combination, as well as changes in accuracy when using different sizes of ensemble. Our method for improving Online Bagging is then

derived based on our observation. Our algorithms are shown in section 4.3.

4.3 Adaptive Ensemble Size Online Bagging Algorithm

The AES methods are presented in algorithms 8 and 9 below.

Algorithm 8 AES with OzaBag

Input: N is an evolving data stream ($N \rightarrow \infty$);
T is the training set drawn from N with examples x;
Y is the finite set of target class values y;
HT is the base learning algorithm (Hoeffding trees);
M is the number of models in the ensemble, each with examples drawn from T;
min and max ensemble size values (set to 10 and 25, respectively);
δ is the generalized Kronecker function: $\delta(a,b) := \begin{cases} 1 \text{ if } a = b \\ 0 \text{ if } a \neq b \end{cases}$

1: **for all** $m = 1$ to M **do**
2: S_m = random sample of size d drawn from T, with replacement
3: h_m = model induced by HT from S_m
4: Compute average change in memory, B
5: Set $k = Poisson(1)$
6: **for all** $n = 1, 2, ..., k$ **do**
7: Update h_m with current examples $S_m \in T$
8: **if** $B > A$ & $M < max$ **then**
9: $M \leftarrow M + 1$
10: **if** $B < A$ & $M > min$ **then**
11: $M \leftarrow M - 1$
12: $A \leftarrow B$

Output: hypothesis: $h_{fin}(x) = argmax_{y \in Y} \sum_{m=1}^{M} \delta(y, h_m(x))$

We presented the OzaBag and OzaBag with ADWIN (OzaBagADWIN) algorithms in chapter 3. Based on the observations from our preliminary experimentation, we made modifications to the ensemble size used in the learning process of these two metalearning algorithms. In the modification, the ensemble size is incremented by one before the next learning process, if the memory utilized by the current ensemble is more than the previous one. Also, if the memory utilized drops, the ensemble size is decremented by 1. A maximum and minimum ensemble size is maintained to guarantee an optimal result.

Similar to the Online Bagging algorithm, the AES draws a training window, T, from a stream of evolving data, N. An ensemble of models, h_m, is induced by Hoeffding trees, from a number of random sample, $S_m \in T$, where $m = 1$ to M. At this instant, the average memory of the models in the ensemble is computed. While the stream is still evolving, the base models are updated with current examples, and the change in average memory utilized

Algorithm 9 AES with OzaBagADWIN

Input: N is an evolving data stream ($N \rightarrow \infty$);
T is the training set drawn from N with examples x;
Y is the finite set of target class values y;
HT is the base learning algorithm (Hoeffding trees);
M is the number of models in the ensemble, each with examples drawn from T;
min and max ensemble size values (set to 10 and 25, respectively);
λ is a Poisson distribution parameter;
δ is the generalized Kronecker function: $\delta(a, b) := \begin{cases} 1 \text{ if } a = b \\ 0 \text{ if } a \neq b \end{cases}$

1: **for all** training examples (x, y) **do**
2: **for all** $m = 1$ to M **do**
3: S_m = random sample of size d drawn from T, with replacement
4: h_m = model induced by HT from S_m
5: Compute random binary value $\mu_m(y)$ and average change in memory, B
6: Set $w = Poisson(\lambda)$
7: Update h_m with current example with weight w and class $\mu_m(y)$
8: **if** ADWIN detects change in error of one of the models (h_i) **then**
9: Replace the model with highest error with a new model (h_n)
10: **if** $B > A$ & $M < max$ **then**
11: $M \leftarrow M + 1$
12: **if** $B < A$ & $M > min$ **then**
13: $M \leftarrow M - 1$
14: $A \leftarrow B$

Output: hypothesis: $h_{fin}(x) = argmax_{y \in Y} \sum_{m=1}^{M} \delta(y, h_m(x))$

by the ensemble is computed, for every new update. The change in average memory for the previous set of T is stored in A, while the change for the current set of T is stored in B.

For every iteration of the algorithm, the values of A and B and compared. The size of the ensemble, M, is incremented by 1 if $B > A$, i.e. more memory is being utilized in the current window, compared to the previous window. The size of M is also decremented by 1 if the value of B is smaller than the value of A, indicating that the memory utilization is lower in the current window. The size of M is kept between the allocated maximum and minimum values, i.e. $min \leq M \leq max$. By varying the size of M as we have done in the AES algorithm, we are able to conserve resources, without compromising on the accuracy of the classification results.

The experimental approach that led us to these modifications are presented below.

4.4 AES Algorithms

Recall that in section 2.5, we introduced the utility and efficiency consideration in data streams. The cost of building and applying the model becomes an important factor in

our improvement technique. We take the memory usage of the ensemble object, and the ensemble size into consideration. To this end, a preliminary experimentation was performed to observe the effect of concept drift on the size of an ensemble object in system memory. We also observed the effect of the ensemble size to the accuracy of the results obtained. Before explaining these further in section 4.4.2 and 4.4.3, the dataset to be used in the preliminary experimentation is shown in section 4.4.1.

4.4.1 KDD'99

The KDD'99 dataset has frequently been used in research on data stream mining. The dataset, which is based on the DARPA'98 network capture, was prepared specifically for the Knowledge Discovery in Databases (KDD) Cup in 1999 (KDD99, 2009). KDD Cup is an annual Data Mining and Knowledge Discovery competition, and the task was to come up with a predictive model (a classifier) for detecting network intrusion, i.e. to distinguish between normal and abnormal network connection. The original raw training dataset contains about 4 gigabyte of compressed tcpdump, representing 7 weeks of network traffic, containing 4,898,431 connection records. The test dataset representing 2 weeks of data, contains 311,029 connection records. Both contains 42 features, the label inclusive.

The training set has label of "normal" or 22 other labels, as shown in table 4.1, based on the attack type. The test set has same labels with additional 17 labels. These attack types fall into four categories:

Table 4.1: Categories of attacks in the KDD '99 dataset

Class	**Attacks in the training set**
Probe	ipsweep, nmap, portsweep, satan
DOS	back, land, neptune, pod, smurf, teardrop
U2R	buffer_overflow, loadmodule, perl, rootkit
R2L	ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster
	Additional attacks in test set
Probe	mscan, saint
DOS	apache2, mailbomb, processtable, udpstorm
U2R	httptunnel, ps, worm, xterm
R2L	named, sendmail, snmpgetattack, snmpguess, sqlattack, xlock, xsnoop

- *Denial-Of-Service (DOS) Attack:* It is a computer network attack that attempts to make a computer or network resource unavailable for use, usually by overloading it with request.
- *User to Root (U2R) Attack:* This is an attack type in which a malicious user is able to gain root/administrative access of a remote system (or server), by sniffing, guessing

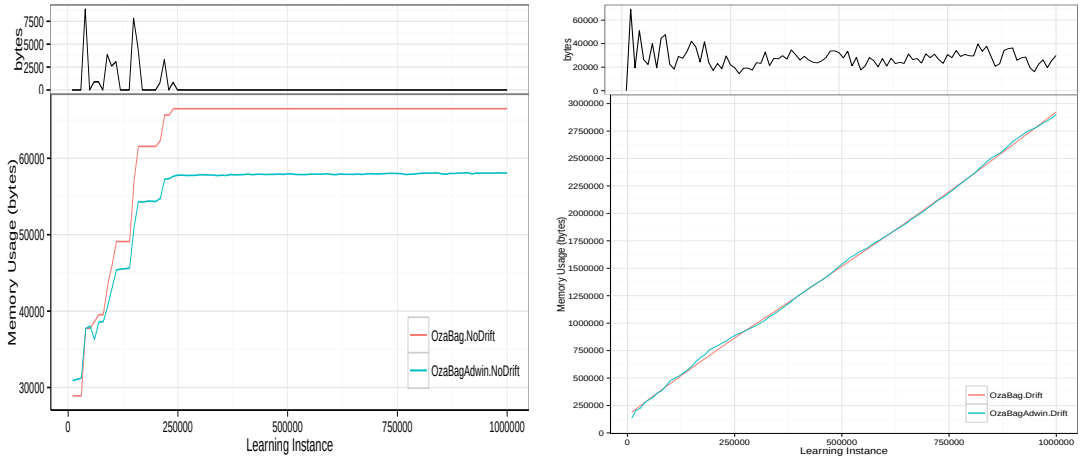
password, or any other way, after starting as an authorized user.

- *Remote to Local (R2L) Attack:* An attack type in which the user attacks the system from any network by exploiting some vulnerability to gain root/administrative access, without actually having an account on that system.
- *Probing Attack:* This is a related to R2L, as the attacker would need to probe the system by scanning the ports, before gaining root access.

The considered protocols in the dataset are Transmission Control Protocol (TCP), User Datagram Protocol (UDP), and Internet Control Message Protocol (ICMP). As it was done in the competition, and also for memory requirements, we will be making use of 10% of the overall dataset, containing 494,021 records with same 41 features, and label.

4.4.2 Size of Ensemble Object in Memory

One basic assumption we are making in our approach is the fact that the size of an ensemble object in memory tends to be proportional to its content. As explained in section 3.2, an ensemble comprises of classification models built from the data stream. We simulated two artificial streams of data in Massive Online Analysis (MOA) to test this assumption. MOA is a framework for data stream mining, and will be introduced in details in section 5.4. One stream is simulated with concept drift and the other without concept drift.



(a) Stream without concept drift

(b) Stream with concept drift

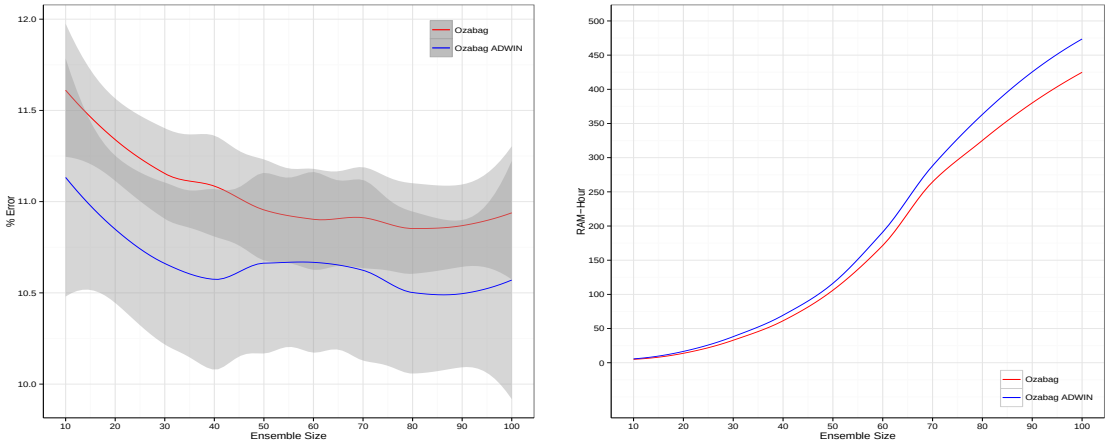
Figure 4.2: Memory Increase in the Data Stream

In figure 4.2, we show the memory increase pattern as more instances are evaluated. The images also show the way the memory usage rise and fall as the learning process proceeds. As we can see from figure 4.2(a), in a steady stream with no randomness or

drift whatsoever, the memory usage quickly builds up to a constant, after which there is no more change or increment, since every new ensemble is exactly the same size as the last. In figure 4.2(b) however, there is a constant increase in memory consumption due to the constant change in the stream. It is also worth noting that both OzaBag and OzaBagADWIN have similar memory usage characteristics, with OzaBagADWIN consuming a little more memory in a stream with no drift. This can be attributed to the fact that they are essentially the same algorithm, with ADWIN only working as a change detector.

4.4.3 Ensemble Size

The question of the ensemble size in ensemble learning is important because it affects the accuracy of the results obtained. It also affects the time and resources used in the learning process. Earlier work from Hansen and Salamon (1990) using sizes 3, 5, 11 and 15, indicated that the gain is levelling off for the higher ensembles, suggesting an optimal size of about 10. Building on this, Opitz and Maclin (1999) performed some further studies and confirmed that with ensemble size approaching 25 and beyond, the error reduction nearly asymptotes to a plateau.



(a) Error with increase in ensemble size

(b) RAM-Hour with increase in ensemble size

Figure 4.3: Error and RAM-Hour With Increase in Ensemble Size

We performed experiments of our own using OzaBag and OzaBagADWIN, and varying ensemble sizes from 10 through 100, at intervals of 10. Figure 4.3(a), shows the error (100%− accuracy) plotted against the ensemble size. As shown in the figure, for this dataset, the error rate becomes statistically insignificant after ensemble size 30. We can also see from figure 4.3(b) that the computing cost of the algorithms drastically increases

between ensemble sizes 20 and 30 and continues in that trajectory. Although a slightly lower error rate is noticed for ensemble sizes between 80 - 100, it costs exponentially more in terms of memory utilization, as can be seen from the figure and a trade-off is required, to balance accuracy with the cost of computing.

Table 4.2: Accuracy, Time and RAM-Hour for Different Ensemble Sizes

	OzaBag Classifier				OzaBagADWIN Classifier			
Size	Accuracy	Error	Time	RAM-Hr	Accuracy	Error	Time	RAM-Hr
10	88.3101	11.6899	258.1505	3.7057	88.8445	11.1555	289.2415	4.0624
20	88.8577	11.1423	545.5511	15.8006	89.2305	10.7695	630.9148	19.4642
30	88.7556	11.2444	828.5681	35.5382	89.2368	10.7632	931.4664	39.7808
40	88.8945	11.1055	1093.4734	62.5377	89.4224	10.5776	1229.6467	70.7882
50	89.1209	10.8791	1420.5763	102.2533	89.4813	10.5187	1588.2306	114.2099
60	89.0197	10.9803	1920.8247	175.1958	89.0602	10.9398	2154.3582	188.7208
70	89.1535	10.8465	2474.9559	252.9820	89.6675	10.3325	2755.5237	288.2580
80	89.0871	10.9129	3098.9755	354.4388	89.2892	10.7108	3414.0819	375.2574
90	89.2343	10.7657	2675.1987	348.0295	89.6773	10.3227	3033.8762	413.6293
100	89.0213	10.9787	3006.8569	435.9619	89.3748	10.6252	3333.2110	477.4898

Taking the lower cost OzaBag algorithm as an example, we can see from table 4.2 that a change from 10 to 20 gives us an accuracy improvement of 0.5474%, but with more than four times increase in computational cost. An ensemble size increase to 40 adds a very slight accuracy increase of 0.0364% but at an increase of about four times of what it cost to compute with an ensemble size of 20. That costs almost seventeen times more than the original size of 10. For these reasons, and going by previous works (Hansen and Salamon, 1990; Opitz and Maclin, 1999), we settled on using a range of 10 to 25 as optimal ensemble size in this experimentation.

4.4.4 Improvement

Taking findings from sections 4.4.2 and 4.4.3 into consideration, we postulate the following:

> *A higher change in memory utilization during stream classification indicates new kinds (by class type) of data at that particular stream window as a result of concept drift. Hence, there is a need to consider adapting the ensemble size to guarantee a higher accuracy.*

In line with this, we propose a method that adapts the size of the ensemble, depending on the memory cost of the current window. By maintaining a variable ensemble size, we improve the accuracy of the learning algorithm without incurring a significantly higher cost.

4.5 Summary

In this chapter, we provided the details of our AES Online Bagging technique, as well as the methodology we followed. This method is intended to maximize result accuracy within acceptable computational cost to the system. To this end, we performed a preliminary set of experiments to determine the size of an ensemble object in memory, as well as the effect of the ensemble size to the accuracy of the results obtained. These enabled us to decide on how best to make our improvements in our proposed algorithms. Building on these, we implemented an AES Online Bagging algorithm. Before presenting the results obtained for our algorithms in Chapter 6, we will be discussing the experimentation setup used, the datasets as well as the evaluation criteria in the next chapter.

CHAPTER 5

Experimentation Setup

In this chapter, we describe the different datasets we used in our experimentation. We will also explain our experimentation setup, and performance measurement criteria.

5.1 Datasets

In the preliminary experimentation we performed in chapter 4, we used the KDD'99 dataset. This enabled us to focus on specific observations, allowing us to extend the algorithms we observed. We eventually added five other datasets, making a total of six datasets, that we used in our comparison. These are well known datasets used in data stream mining and the description of each of them is given below. Before making this comparison, an artificial dataset is generated to enable us determine the effect of concept drift on memory. We used an artificial dataset for this, because it allows us to vary the levels of concept drift in a stream with no noise.

5.1.1 LED Dataset Generator

This Light Emitting Diode (LED) dataset was proposed by Breiman, Friedman, Olshen, and Stone (1984). In the MOA implementation, LED dataset generator is provided, with 7 attributes generated. The task is to predict the number displayed on a 7-segment LED display, where each of the 7 attributes have 10% chance of being noise at its default value. This noise percentage can be alternated or removed as required. The LED generator in

MOA comes in two variants: one with concept drift, and one without. In this thesis, we used the version with concept drift, `LEDGeneratorDrift`, which allows us to simulate concept drift to different number of attributes as required in the stream.

5.1.2 Poker Hand

This dataset contains one million records, with 11 attributes, and the purpose is to predict poker hands. Each record represents 5 playing cards, drawn from a standard deck of 52. Each card is represented twice, with one additional attribute added as description of the ten predictive/class attributes. The classes are represented as numbers 0 to 9 as shown in table 5.1 (UCI, 2007). Concept drift is shown in this dataset by the changing card at hand, i.e. the poker hand.

Table 5.1: Poker Hand Dataset Class Labels

Class	**Decription**
0	Nothing in hand; not a recognized poker hand
1	One pair; one pair of equal ranks within five cards
2	Two pairs; two pairs of equal ranks within five cards
3	Three of a kind; three equal ranks within five cards
4	Straight; five cards, sequentially ranked with no gaps
5	Flush; five cards with the same suit
6	Full house; pair + different rank three of a kind
7	Four of a kind; four equal ranks within five cards
8	Straight flush; straight + flush
9	Royal flush; Ace, King, Queen, Jack, Ten + flush

5.1.3 IMDb

The IMDB-E dataset used is derived from the Internet Movie Database (IMDb) dataset from the Multi-label Extension to WEKA (MEKA) repository. We will discuss more about WEKA in section 5.3. In the original dataset as used by MEKA, the dataset has 28 labels representing movie categories. In the data stream adaptation, it consist of binary labels 1 and 0, representing a user's interest (or not) in a movie at a particular time. To simulate concept drift, the interest changes after some time. The total number of records is 120,919, with 1002 attributes, including the preference label. The records are Term Frequency - Inverse Document Frequency (TF-IDF) representation of movie annotations, to indicate how important they are to the text corpus (Žliobaite, Bifet, Pfahringer, and Holmes, 2014).

5.1.4 Forest Cover Type

The dataset is used to predict the cover type of a forest from different cartographic variables, without any remote data sensing. The dataset contains the cover type of a 30 x 30 meter forest observation from United States Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent data variables were derived from both USFS and United States Geological Survey (USGS) data. It contains 54 attributes, the class labels inclusive. The labels are represented as numbers 1 to 7 as shown in table 5.2 (UCI, 1998). Concept drift in this dataset is as a result of the changes in geographical condition because of time and weather changes.

Table 5.2: Forest Cover Type Dataset Class Labels

Class	**Decription**
1	Spruce/Fir
2	Lodgepole Pine
3	Ponderosa Pine
4	Cottonwood/Willow
5	Aspen
6	Douglas-fir
7	Krummholz

5.1.5 Electricity

The electricity dataset is another dataset used in data stream mining research. The task is to predict the rise and fall of the price of electricity in New South Wales, Australia. It covers a period of 2 years, representing a total of 45,312 instances of 6 input variables, taken every 30 minutes. The class labels are binary values of *UP* or *DOWN* representing when the price rises or fall respectively. Concept drift is shown in the dataset by the changing consumption habits, events, and seasons (Žliobaite, 2013).

5.1.6 Airline

The airline dataset contains a record of flight schedules for commercial flights within the USA from October 1987 to April 2008. The one used in our experimentation, as well as many other data stream mining research is a subset of the original dataset of size 5.76GB containing 116 million records. This version contains 539,383 records, with 8 attributes. The task is to predict if a flight is delayed or not, hence the class label is the ”Delay” attributes, with binary values 1 or 0, indicating if a flight is delayed or not respectively

(Ikonomovska, 2011). Concept drift in the dataset is as a result of the changes in the flight schedules.

Table 5.3 shows a summary of these datasets.

Table 5.3: Summary of Datasets Used

Name	Number of Records	No of Attributes	Characteristics	Concept Drift
LED Generator	As required	7	Numeric	Yes
KDD'99 (10%)	494,021	42	Numeric, Nominal	Yes
Poker Hand	829,201	11	Numeric, Nominal	Yes
IMDb	120,919	1002	Numeric	Yes
Forest Covertype	581,012	55	Numeric, Nominal	Yes
Electricity	45,312	9	Numeric, Nominal	Yes
Airline	539,383	8	Numeric, Nominal	Yes

5.2 Pre-processing the Datasets

A point to remember while pre-processing the datasets is keeping their distribution realistic, because they are datasets from real life scenarios. For the numerical attributes, we compared which works best in terms of time to process, between WEKA's implementations of normalization and standardization. We eventually chose to normalize after a comparative experiment. The normalization procedure reduces the range of the numbers, by converting all numeric values in the given dataset (apart from the class attributes, if set). The resulting values are by default in (0,1) range. The weights of these attributes were kept exactly as they were before the normalization. The standardization attribute filter standardizes all numeric attributes in the given dataset to have zero mean and unit variance (apart from the class attribute, if set). A test was conducted to find out which works best in terms of accuracy, and its was discovered that the standardization method varied the accuracy results slightly, and it is not any faster than normalization. As a matter of fact, its was about 2-3% slower in processing time.

We performed a secondary preprocessing task on the KDD'99 and Forest Cover Type datasets by applying the Obfuscate filter in WEKA. This is an instance filter that renames the relation, attribute names and all nominal (and string) attribute values. It is particularly useful for sensitive data (like medical records), but it was used on these two datasets because the attribute names are much longer than the other datasets. They also contain relatively more attributes than the other datasets, besides the IMDb dataset. Before this filter, the attribute labels on the KDD'99 dataset are:

```
@attribute label
   {back,teardrop,loadmodule,neptune,rootkit,phf,
```

```
satan,buffer\_overflow,ftp\_write,land,spy,ipsweep,multihop,
smurf,pod,perl,warezclient,nmap,imap,warezmaster,portsweep,
normal,guess\_passwd}
```

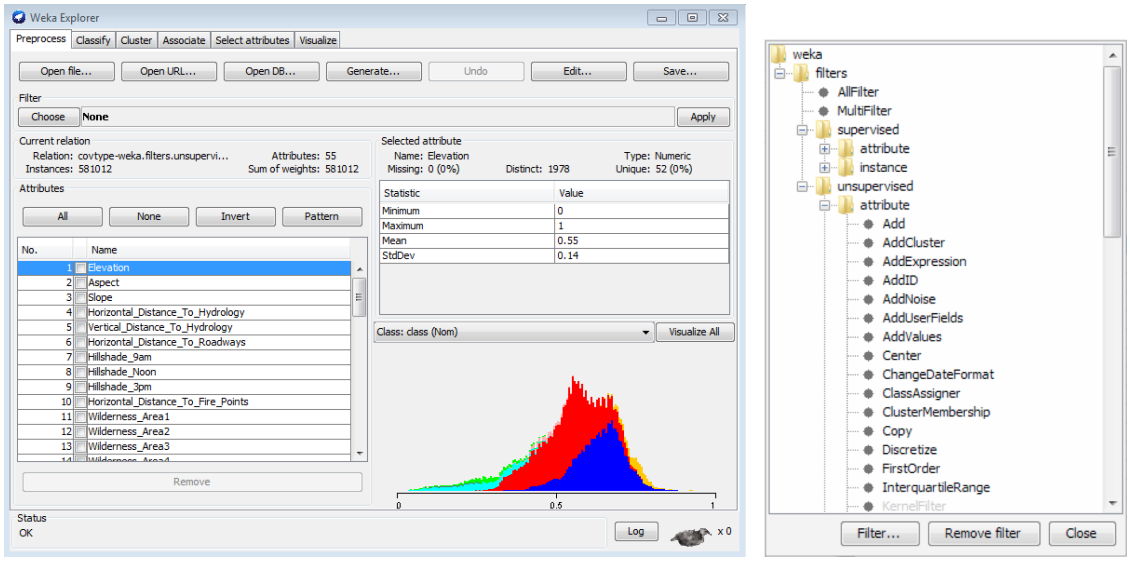
After applying the filter, they were converted to:

```
@attribute A42
    {V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,V11,V12,V13,V14,
    V15,V16,V17,V18,V19,V20,V21,V22,V23}
```

We found out that this procedure has an effect on the memory footprint of the experimentation software. We compared the speed of an experiment before and after the obfuscation, and we observed an increase by 5%. This could be attribute to the fact that having to load the longer attribute names while processing the datasets takes up some more bytes in memory.

5.3 WEKA

Waikato Environment for Knowledge Analysis (WEKA) is a data mining software from the University of Waikato. It was conceived to gather machine learning algorithms in a unified framework to aid researchers in the field. WEKA is now used in that regards, and in some cases for business purposes too (Hall, Frank, Holmes, Pfahringer, Reutemann, and Witten, 2009). Because WEKA is the foundation of other frameworks such as MOA, some tasks are still been handled by WEKA in the University of Waikato data mining suite. One such task is data-preprocessing, as MOA does not have a pre-processing function built in.



(a) WEKA Pre-processing Interface (b) Filter Selection

Figure 5.1: WEKA Pre-processing

Recall that in section 5.2, we explained that we normalized and obfuscated the datasets in our pre-processing procedure. This was done by applying the "Normalize" and "Obfuscate" unsupervised attribute filters on the required datasets. The altered dataset can then be saved for further processing.

5.4 MOA

After the pre-processing is concluded using WEKA, the primary experimentation was performed using the Massive Online Analysis (MOA) software, a framework for data stream mining (Kranen, Kremer, Jansen, Seidl, Bifet, Holmes, Pfahringer, and Read, 2012; MOA, 2013). MOA is designed specifically for running online algorithms on streaming data and it deals with the challenge of simulating streaming environment for testing algorithms on real datasets. That is, it allows us to make algorithm comparison in steaming settings. MOA includes many machine learning algorithms, supporting classification, multi-label classification, regression, clustering and outlier detection tasks. We will be focusing on the classification task as it relates to our experimentation.

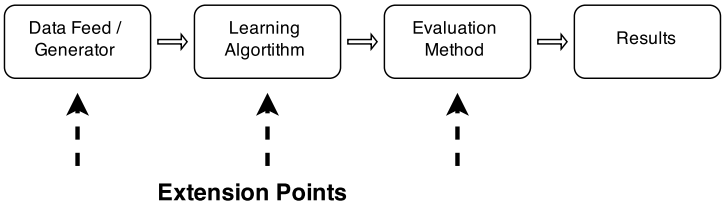


Figure 5.2: MOA Framework Workflow

Figure 5.2 shows a simple representation of the MOA framework workflow. At the beginning of the workflow, data are either fed in or generated by the software. The software is equipped with different data stream generators, and it simulates concept drift with either the generated stream or the data that is fed in. A learning algorithm is subsequently applied on the data, based on the task at hand. All the algorithms we previously represented in chapter 3 are implemented in MOA.

The evaluation method helps to determine which part of the dataset is used for training the algorithm in use, and which is used to test the output model. This is another very important issue to consider in data stream mining. Recall that we described some key characteristics of a data stream in section 2.2. The fact that the data are usually generated at a high rate, and the unboundedness nature therefore makes it impossible to store the dataset in its entirely. We therefore don't have the luxury of a separate test set in most

cases, and will have to work with the stream as it is. There are two commonly used methods in data stream evaluation (Bifet, Holmes, Kirkby, and Pfahringer, 2010b):

Holdout: This method takes a part of the data stream as test set (the holdout set) at periodic intervals, and evaluated on subsequent arriving data, acting as the training set. The holdout test can later be returned as training set after the interval, and a new set of data is selected as the holdout set, to track performance over time. This method obviously does not consider the entire training set in its accuracy measurements, as the result is based only on the holdout set.

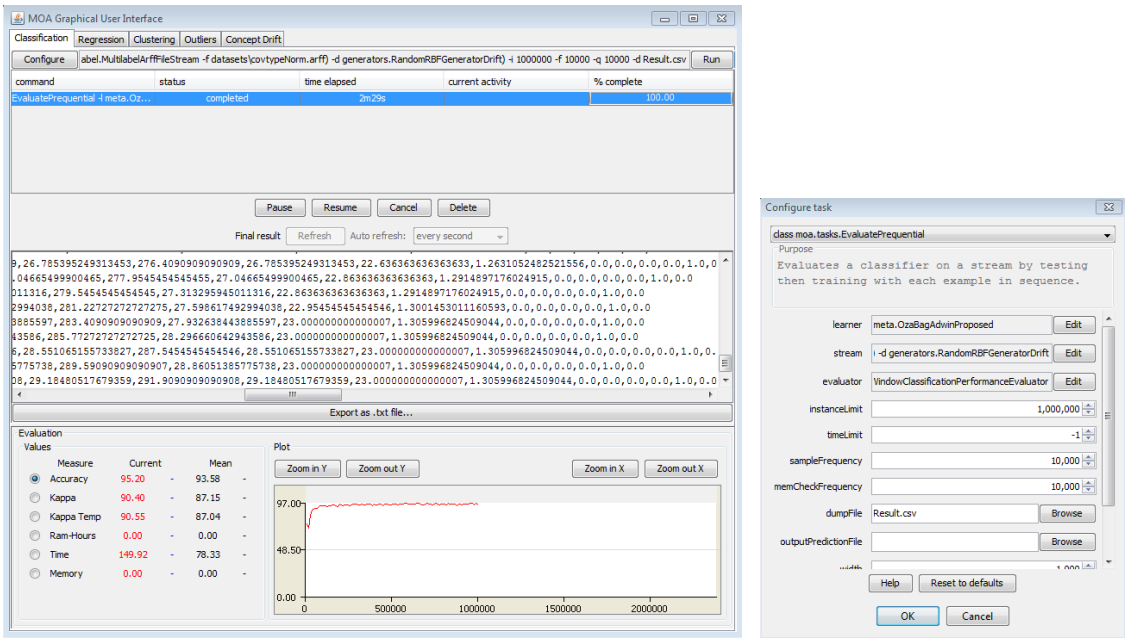
Prequential (Interleaved test-then train): Another method, and the method preferred in this thesis, is the prequential method. In this method, each example in the dataset is first used as a test set, and then used as a training set. The accuracy is then incrementally updated along the way. This method makes maximum use of the streaming data, as it does not require a holdout set, and it ensures that each example in the evaluation window is used in the learning process. It also does not bias the overall results to a particular window, providing a smooth accuracy increment over time (Bifet, Holmes, Kirkby, and Pfahringer, 2010b).

As shown in figure 5.2, the framework can be extended as desired in the data, learning, or evaluation phase to add more capabilities by using the well documented MOA Application Program Interface (API) (MOA). Experimentation can be initiated by using either the command-line or the the software Graphical User Interface (GUI). An example of a command line usage to evaluate 1,000,000 instances of the forest cover type dataset using OzaBagADWIN, will be:

```
java -Xmx12G -cp moa.jar;weka.jar -javaagent:sizeofag.jar
   moa.DoTask "EvaluatePrequential -l (meta.OzaBagAdwin -s
   16) -s (ConceptDriftRealStream -s
   (generators.multilabel.MultilabelArf fFileStream -f
   datasets\covtypeNorm.arff) -d generators.RandomR
   BFGeneratorDrift) -i 1000000 -f 10000 -q 10000 -d
   Result.csv"
```

The `weka.jar` file in the command allows us to use algorithms from WEKA in MOA (Bifet, Kirkby, Kranen, and Reutemann, 2012). To launch the GUI using similar start-up parameters, we use:

```
java -Xmx12G -cp moa.jar;weka.jar -javaagent:sizeofag.jar
   moa.gui.GUI
```



(a) Main MOA Classification GUI (b) MOA Task Configuration GUI

Figure 5.3: MOA Classification and Task Configuration Interface

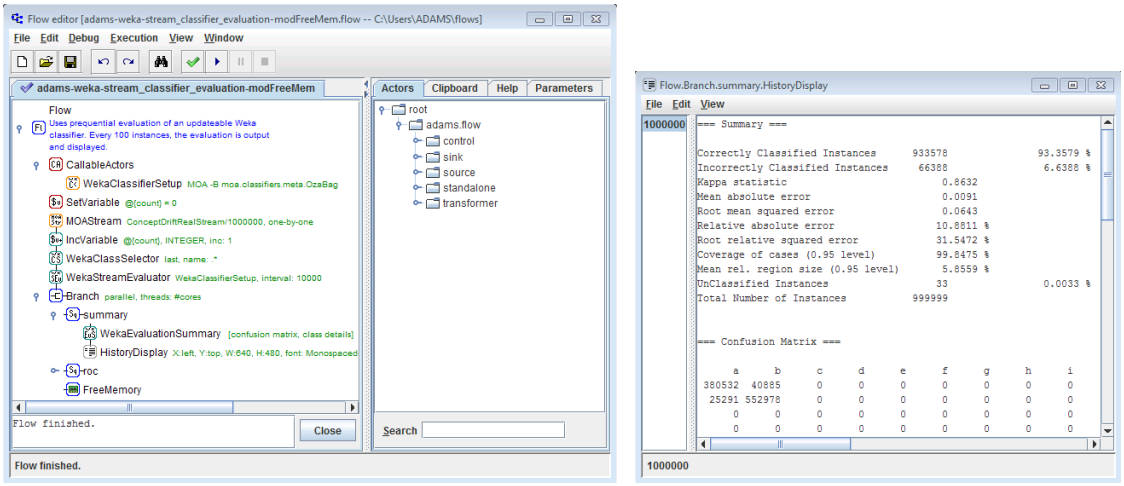
Figure 5.3 shows the GUI of the same action in MOA. In figure 5.3(a) we can see a plot of the accuracy as more instances of the stream are processed. Because we used prequential evaluation, the initial accuracy indicated by the graph is very low. This however quickly improves as more examples are tested and trained until a plateau is reached. The detailed evaluation computations can also be seen and can be exported as a `.txt` if required. These details can as well be saved as a `.csv` file for use after the process is completed. This is the preferred option for us during the experimentation. The task configuration interface handles the major settings and selections, as can be seen in figure 5.3(b). In this interface, the desired evaluation method and algorithm are selected. Other parameters such as the stream method, number and frequency of instance e.t.c. can be applied from this interface.

5.5 ADAMS

Advanced Data mining And Machine learning System (ADAMS) is another novel addition to the machine learning community from the team at University of Waikato. It provides a way to prototype complex workflow, called *flow*, by laying out constituent pieces on a blank page, analogues to drawing on a canvas (Reutemann and Vanschoren, 2012). ADAMS add modules, called *actors*, from other frameworks such as MOA, WEKA, MEKA, R amongst others (ADAMS). While the potential of the flow in ADAMS is vast, it is particularly

useful for us because it provides a way to utilize some measurement parameters, such as *f-measure* and *recall*, not provided in MOA.

While these kind of results can be obtained from WEKA, using it would provide us with an inaccurate representation of what the result should really be. This is because WEKA does not have an in-built data stream evaluation method such as holdout and prequential. ADAMS therefore provides us a way to combine stream evaluation techniques such as the ones in MOA with any combination of algorithms from WEKA or MOA. The data stream can subsequently be processed and the evaluation results can be generated as desired.



(a) ADAMS Flow Editor (b) Result Summary

Figure 5.4: ADAMS Flow Editor and Result Summary

Table 5.4: Weighted Average

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
Weighted Average	0.934	0.075	0.934	0.934	0.933	0.864	0.985	0.986

A simple example of a flow in ADAMS is shown in figure 5.4(a). In the example, the OzaBag algorithm is called as a WEKA classifier using MOA wrapper. The stream can be generated artificially or from a dataset using "MOAStream" actor, and the "WekaStreamEvaluator" allows prequential evaluation on the generated stream. Although it is called WekaStreamEvaluator, it is different from any of the evaluators in the standalone version of WEKA as it currently is. The summary result is then generated as can be seen in figure 5.4(b). The other part of the result not shown in the figure is presented in table 5.4.

5.6 Measuring Learner Performance

Measuring the performance of a data stream mining algorithm is a crucial step in determining the viability of the algorithm to practical applications. The success of an algorithm are measured by considering how well it performs in comparison to other state-of-the-art methods. A common evaluation criteria in data streams is the accuracy of the learner. This takes the class of the data tuples in our measurement. We will discuss about this in section 5.6.1, especially with regards to concept drift in data streams.

5.6.1 Cost-Sensitive Learning and Accuracy

As we have previously indicated, cost-sensitive learning is completely different from cost-sensitive adaptation introduced in section 2.5. Cost-sensitive learning takes the misclassification of data class into consideration in the mining process, with a goal of minimizing the total error cost (Ling and Sheng, 2010). The overarching aim of cost-sensitive learning in data mining is to take every error into account, so as to avoid costlier errors. Using an hospital as an example, while it is inconvenient to wrongly diagnose a patient as having a disease, it is more costly to wrongly flag an ill patient as healthy.

With the assumption that a classification prediction can either be positive or negative, denoting correct classification, or misclassification, cost-sensitive learning uses four terms to identify the classifications:

- False Positive (FP): Actual negative, but falsely misclassified positive;
- False Negative (FN): Actual positive, but falsely misclassified negative;
- True Positive (TP): Actual positive, and correctly classified positive;
- True Negative (TN): Actual negative, and correctly classified negative.

From the four classifications, we can deduce that $TP + FN$ and $TN + FP$ are the actual positive(P) and negative(N) values respectively. FP is equivalent to false alarm, and detecting this is important for sensitive classification tasks.

Table 5.5: Confusion Matrix

	Actual Negative	Actual Positive
Negative Prediction	TN	FN
Positive Prediction	FP	TP

A *Confusion Matrix*, or cost matrix as shown in table 5.5, is used to analyse how well a classifier can recognize datasets of different classes. From the confusion matrix, we can get

the *sensitivity*, and the *specificity*. These are used to get the value of the accuracy. The sensitivity is the rate of TP and is equal to $\frac{TP}{P}$, while the specificity is the rate of TN and is equal to $\frac{TN}{N}$. The accuracy of a classifier, which is the percentage of examples that are labelled correctly by the classifier, is given as (Han and Kamber, 2006):

$$accuracy = sensitivity \cdot \frac{P}{P+N} + specificity \cdot \frac{N}{P+N} = \frac{TP+TN}{P+N} \tag{5.1}$$

The confusion matrix is also useful for deriving the percentage of the retrieved examples that are relevant, and the percentage of the relevant examples that are retrieved. These are known as the *precision*($\frac{TP}{TP+FP}$) and *recall* respectively. The recall is also known as Sensitivity. The harmonic mean of the precision and recall is known as the *f-measure* and is given as:

$$f\text{-}measure = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.2}$$

The purpose of the thesis is to improve on a classification algorithm in a stream with concept drift. We focused on the accuracy of the classifiers, without compromising on other factors such as the processing time and other cost sensitive adaptation consideration of the classification process. Comparison for measurements such as f-measure were taken but we found out that there is no change to this measures before and after our implementation. This can be attributed to the fact that the learning heuristics in the algorithm is exactly the same, as this remains true also while the size of the ensemble is varied on the original algorithms.

5.6.2 Accuracy Measurement in Data Streams with Concept Drift

Accuracy plays a very important role in data stream mining. As shown in the previous section, the accuracy of a classifier on a given dataset is the percentage of the test dataset examples that are correctly classified by the classifier. On a changing data stream, however, the kappa statistics is preferred for accuracy measurement. This is due to the potential changes in the distribution of the classes in the dataset (Bifet and Frank, 2010). The value of the Kappa Statistics is given as:

$$k = \frac{p_0 - p_c}{1 - p_c} \tag{5.3}$$

Where p_0 is the accuracy of the classifier and p_c is the probability that a correct prediction of assigning exactly the same number of examples per class in every iteration

of the window is made by a chance classifier (or no-change classifier). If the no-change classifier is always correct, i.e. $p_c = 1$, then $k = 1$; and if p_c coincides with the prequential accuracy, i.e. $p_c = p_0$, then $k = 1$ (Bifet, Read, Žliobaitė, Pfahringer, and Holmes, 2013). Note that p_0 is also known as the *prequential* accuracy. This measure is related to the prequential evaluation that was previously introduced in section 5.4. While the prequential evaluation methods test and train each example in the dataset, the prequential accuracy is constantly updated as the learning process progresses. p_c is called the No-Change classifier. Bifet and Frank (2010) provides more details about the computation of obtaining these values.

A modified Kappa Statistics, called Kappa Plus Statistics (k^+)(Bifet, Read, Žliobaitė, Pfahringer, and Holmes, 2013) (or temporal kappa) is further recommended to address the temporal component of evolving data stream with concept drift. The temporal component is the change in class label with time. This is directly analogous to concept drift as explained in section 2.4. The Kappa Plus Statistics works by using a sliding window mechanism of size w with the most recent observation, as proposed by Gama, Sebastião, and Rodrigues (2013). The value of the Kappa Plus Statistics is given as:

$$k^+ = \frac{p_0 - {p'}_e}{1 - {p'}_e} \tag{5.4}$$

Where ${p'}_e$ is a the accuracy of a no-change classifier. The no-change classifier uses temporal dependence information by predicting that the next class to be seen will have the same label as the last class that was seen in the sliding window (Bifet, Read, Žliobaitė, Pfahringer, and Holmes, 2013).

5.6.3 Utility Measurement

Recall that in section 2.5, we showed that ROI was introduced as a measurement for the derived value, or utility, by the adaptation of stream mining algorithms. In the literature as presented by Žliobaite, Budka, and Stahl (2015), the focus was not in comparing between ensemble method, but in comparing adaptation strategies, hence the need to measure by changes in adaptation. Working from the ROI as was presented in the literature and to enable us make direct comparison between adapting and non-adapting algorithms, we will measure the ROI between each steps of processed instances. Also, we will only be comparing between the two AES implementations, and both alternative ensemble size setting of OzaBag and OzaBagADWIN.

Recall that in section 2.5, the value of the ROI between two periods of adaptions, $\mathcal{T}$

and $\mathcal{T}+1$, is given as $ROI_{\mathcal{T}} = \frac{\gamma_{\mathcal{T}}}{\psi_{\mathcal{T}}}$; where γ_T is the change in prediction accuracy, and $\psi_{\mathcal{T}}$ is the computational cost, measured in RAM-hour (Rh). To compute the mean ROI, the total number of adaptation performed, $\mathcal{T}'$, becomes $\mathcal{T}$, which is the total number of training set T; and the number of the sample since last adaptation, N_i will be the number of instances in each step of the process (N). Equation 2.2 then becomes:

$$\begin{aligned}\overline{ROI} = \frac{\sum_{i=1}^{\mathcal{T}'}(N_i \times ROI_i)}{\sum_{i=1}^{\mathcal{T}'} N_i} &= \frac{1}{\sum_{i=1}^{\mathcal{T}'} N_i}\sum_{i=1}^{\mathcal{T}'} N_i \frac{\gamma_i}{\psi_i} \\ &= \frac{1}{\mathcal{T} \times N}\sum_{i=1}^{\mathcal{T}} N \frac{\gamma_i}{\psi_i} \\ &= \frac{1}{\mathcal{T}}\sum_{i=1}^{\mathcal{T}} \frac{\gamma_i}{\psi_i}\end{aligned} \tag{5.5}$$

5.6.4 Testing for Statistical Significance

A statistical significance or hypothesis test is used to provide additional support for our observation. It measures the different results obtained to determine if there is an improvement in the model obtained using the different methods or algorithms experimented with. Statistical test used for testing between multiple classifiers, across cases are the Analysis of Variance (ANOVA) and Friedman's test, and they are also called *Omnibus tests* (Japkowicz and Shah, 2011). They both test for multiple hypothesis, and their null hypothesis (H_0) is that all the tested algorithms perform equally. A second hypothesis, H_1, is the rejection of the H_0, and it indicates that there is at least a pair of algorithms in which their performance is significantly different.

Because ANOVA assumes a normal distribution, we will use Friedman's test for our statistical test. Friedman's test is non-parametric. It works without any assumption on the data distribution, by ranking the different classifier results separately and computing the sum of the ranks. Friedman's Statistics (X_F^2) is calculated as (Japkowicz and Shah, 2011):

$$\chi_F^2 = \left[\frac{12}{n \times k \times (k+1)} \times \sum_{j=1}^{k}(R_j)^2\right] - 3 \times n \times (k+1)$$

where n = number of cases; k = number of algorithms; k-1 is known as degree of freedom (5.6)

This value is looked up in the χ^2 distribution table to get the *p value*, signifying $P(\chi_{k-1}^2 \geq \chi_F^2)$. The p value is measured against a predetermined threshold value called the *significance level (α)*. It is common practice to use 1% or 5% for this value, with 5%

been considered a comfortable value, while still having good enough statistical significance (Nuzzo, 2014; Cowles and Davis, 1982). We will be making use of 5%. The null hypothesis is measured against this value of α. In our experimentation, the Friedman's test checks for the following hypothesis:

- H_0 : The performance of the six algorithms are exactly the same;
- H_1 : There exist at least one pair of algorithms in which the performance is different

The null hypothesis, H_0, is accepted if the p value is greater than α, and it is rejected if otherwise (i.e. H_1 is accepted). In our experimentation, this means that the performance of the algorithms are exactly the same when $p > \alpha$; otherwise there is at least two different algorithms with significantly different performance. When the null hypothesis is rejected, the Conover *post-hoc* test (Conover, 1999) is used to rank the algorithms by how best they perform, and identify the significantly different pairs.

5.7 Discussion

In the previous sections, we have presented different measurement metrics for performance measurement in data stream mining. Our first focus is on the Kappa Plus Statistics (k^+), because this is better suited for a stream with concept drift, as explained in section 5.6.2. The results are presented in the next chapter for each dataset used. We use the median of k^+ because a sample mean will constitute a misrepresentation of the result obtained. In section 5.4, we explained that the prequential evaluation first uses each example as test set, and subsequently as training set, incrementally. This means that the initial values of k^+ will be low, but will quickly improve as more evaluations are done. This, therefore, indicates that the initial set of k^+ accuracy values constitute to be outliers, compared to the overall accuracy values obtained. In a data distribution with skewed range or extreme outliers, using the median is a better representation for the measure of central tendency (Krzywinski and Altman, 2014; Massart, Smeyers-verbeke, Capron, and Schlesier, 2005). The accuracy box plots shown in 6.1 shows the sample mean (dark red dot indicator) for comparison. We also use the Friedman's test as a statistical measure, to test the significance of the accuracy results obtained.

As a measure of the utility obtained from the adaptation process, the mean ROI is computed for all the stream mining windows, and it is used to evaluate the value derived as a result of the adaptation. This allows us to make utility comparison between our algorithms and the Online Bagging algorithms used in our experimentation.

The default setting of the ensemble size is compared with our AES implementations. An alternate setting is also used in the comparison, by taking the average of the different ensemble sizes obtained with the AES metalearning algorithms. This is because our implementation varies the ensemble size based on the memory utilization of the last window, as explained in section 4.4. We take the average of the various ensemble sizes used by our implementations as alternate ensemble sizes in our comparison. We therefore have a comparison of our two implementations with the base ensemble size of 10, and an alternate size based on the average size from our implementation, rounded to the nearest whole number. A total of six variants across the different algorithms are compared for each dataset.

5.8 Summary

We presented the datasets and data mining environments used in our experimentation in this chapter. We discussed the different data mining and data stream mining frameworks for the mining process, and the data preprocessing. In section 5.4 we explained the prequential method for evaluating data stream and indicated that we will be using this method in our experimentation. We also showed the methods used in measuring learner performance in section 5.6. We mentioned that the Kappa Plus Statistic (k^+) is the preferred metric for evaluating the learner's accuracy in a stream with concept drift, the Friedman's test is used to measure the statistical significance of the accuracy results, and the ROI is used as the measurement for value derived by the stream mining algorithms.

In the next chapter, we will provide the result to the different experiments performed, comparing our AES implementation to established learning algorithms in the field of data stream mining.

CHAPTER 6

Experimental Results and Discussion

In this chapter, we present the results obtained by the various experimentation performed on the datasets. We compare our implemented AES Online Bagging to OzaBag and OzaBagADWIN algorithms' default implementations, using Hoeffding tree algorithm as base-level leaners. Six different results, representing the different algorithms used in our experimentation are compared for each dataset.

6.1 Experimentation Results

In this section, we present the result for each dataset individually, with analysis of the result. In Chapter 4, we postulated that the memory utilized is sensitive to the amount of attributes affected by the drift i.e. the higher the concept drift, the higher the memory utilized. We performed an initial experiment to test this hypothesis. The result of this is shown in section 6.1.1. We also present a discussion on the k^+ accuracy of the results obtained, as well as the statistical significance of these results in sections 6.1.2 and 6.1.3. These are followed by the memory usage and the time taken to build the models in sections 6.1.4 and 6.1.5, respectively.

6.1.1 Memory Utilization with Concept Drift

To enable us to test our postulation, we simulated an artificial stream with no noise, and introduced concept drift, using the `LEDGeneratorDrift` that was introduced in section 5.1. Recall that the `LEDGeneratorDrift` is a synthetic dataset generator that allow us to simulate concept drift as applied to a different number of its 7 attributes. This allows us to monitor the effect of concept drift with the memory consumed in the learning process. Having zero noise in the stream helps us to minimize the other factors that might have effect on memory utilization. Figure 6.1 shows the visual representation of this, with varying level of concept drift introduced every 20%. The stream was started with zero concept drift.

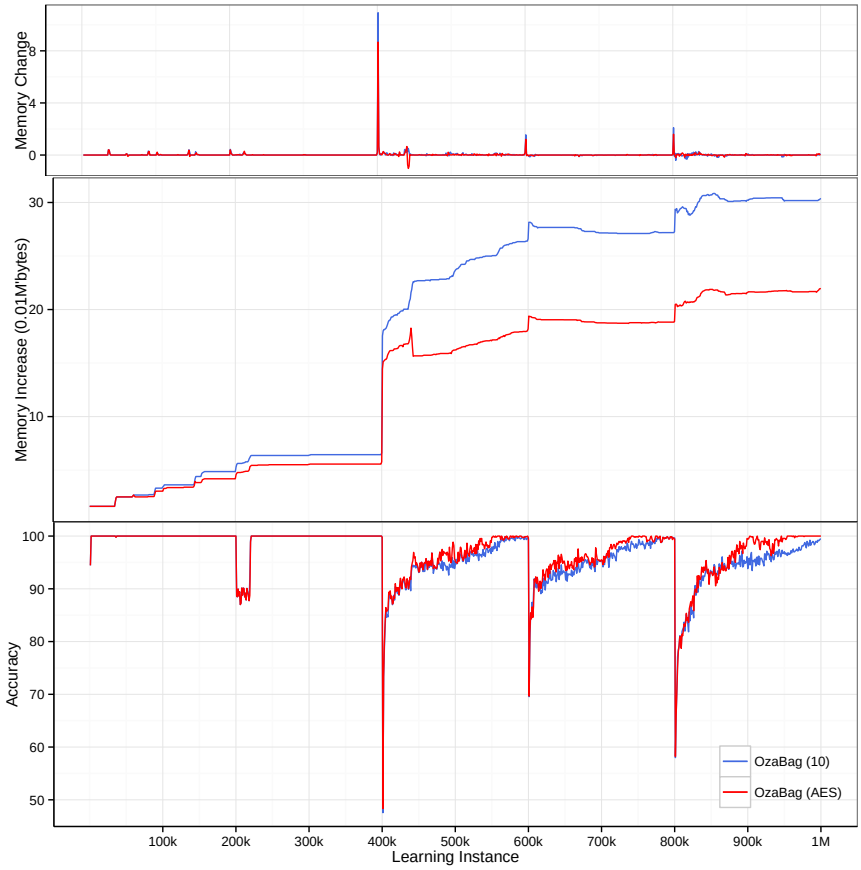


Figure 6.1: Memory Change with Concept Drift

At 20% stream, we introduced a slight drift, affecting 1 of the 7 attributes. We noticed a slight memory rise as a result of this, which remains steady for the duration of the section. At 40% stream, we introduced a full concept drift, affecting all 7 attributes. A much higher memory rise is observed at this point. Our premise is that while other random factors might be able to influence a memory change in any stream, concept drift will cause

a noticeable change in memory, and have a direct effect on the stream mining cost.

At 60%, the concept drift was removed. While there was a little bump in memory consumption for that change, it gradually subsided to the level it was before the change. We introduced a small drift affecting 3 attributes at 80%, and we can see another increase in memory, relative to the concept drift. This shows that an increase in concept drift triggers a proportional increase in memory. In this thesis, we take advantage of these changes to improve the stream mining accuracy of a metalearning algorithm by way of increasing the ensemble size only when necessary.

With this in mind, we present the results of our experimentation on the real life datasets in the remaining part of this section.

6.1.2 Accuracy Comparison

A comparison of the k^+ accuracy results using the algorithms on the different datasets is shown in table 6.1 (e.s. = ensemble size).

Table 6.1: Kappa Plus Statistics Median for the datasets

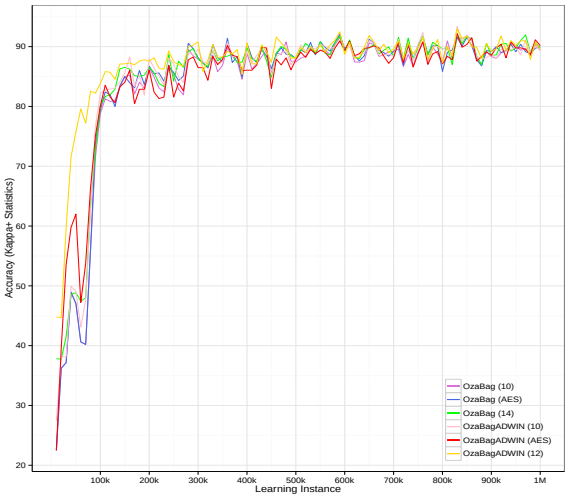
	KDD'99			Poker			IMDb			Forest			Electricity			Airline		
	e.s	k^+	r	e.s	k^+	r	e.s	k^+	r	e.s	k^+	r	e.s	k^+	r	e.s	k^+	r
OzaBag	10	88.0362	6	10	88.7212	4	10	87.8519	5	10	88.3543	6	10	88.4178	2	10	88.2179	4
OzaBag(AES)	14	88.6391	2	15	89.0466	2	12	88.4434	3	14	88.8211	3	14	88.2528	4	12	88.2114	5
OzaBag	14	88.5790	3	15	88.9772	3	12	**88.8997**	1	14	**89.0140**	1	14	88.1120	5	12	88.2061	6
OzaBagADWIN	10	88.5542	4	10	88.7212	4	10	88.3559	4	10	88.3910	5	10	88.4178	2	10	88.3647	3
OzaBagADWIN(AES)	12	88.0736	5	14	88.4064	5	14	87.7557	6	12	88.4043	4	15	88.3007	3	13	88.3848	2
OzaBagADWIN	12	**89.1392**	1	14	**89.2080**	1	14	88.4804	2	12	88.8891	2	15	**88.9806**	1	13	**88.4034**	1

⋆ e.s = ensemble size; r = ranking (lower is better)

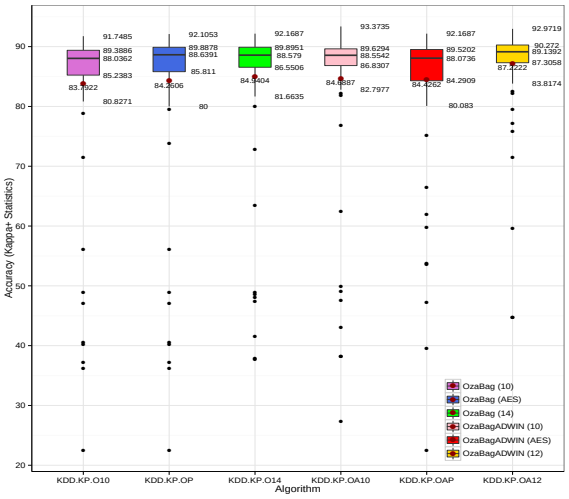
Our first observation is that the results, in terms of accuracy, are quite high. However, the best results are obtained when using OzaBagADWIN in most cases, except for the IMDb and Forest datasets, where we obtained better results using the OzaBag algorithm. Both best results where achieved with the alternate ensemble size. This is quite expected, as we have already established that the larger the ensemble size, the better the classification accuracy.

Recall that we started this experimentation with the aim of intelligently adapting the ensemble size, with change in memory utilization in a stream with concept drift. We can see that our implementation was able to improve the results obtained in most cases, as a result of the adaptation.

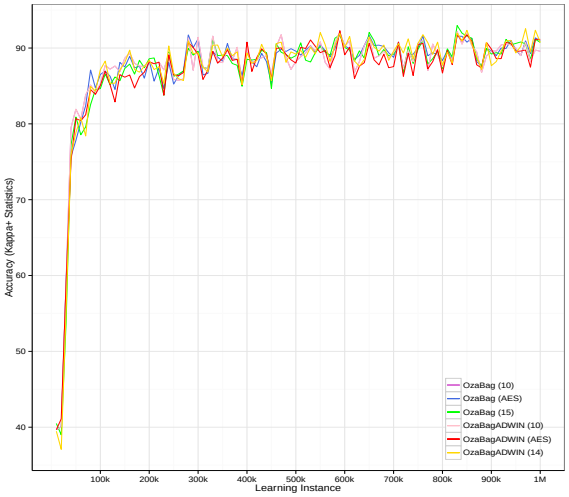
Figures 6.2 and 6.3 shows a visual representation of the k^+ statistics for the various datasets. Given the fact that we don't know the best ensemble size to have in the setting for any random data stream with concept drift, the result obtained when using the AES implementation provides a good compromise. It allows the ensemble size to be set to



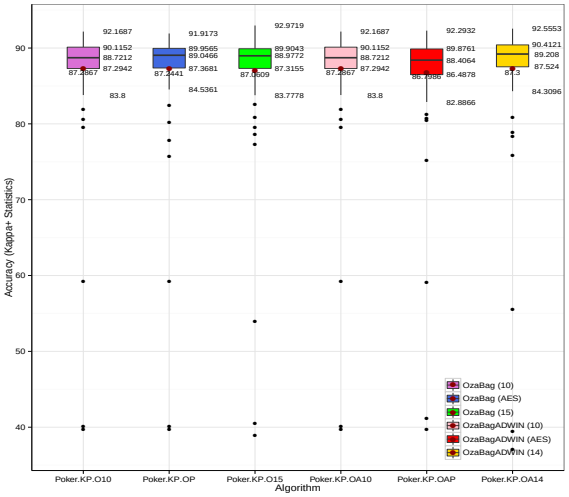
(a) Accuracy Graph Plot for KDD Dataset



(b) Accuracy Box Plot for KDD Dataset

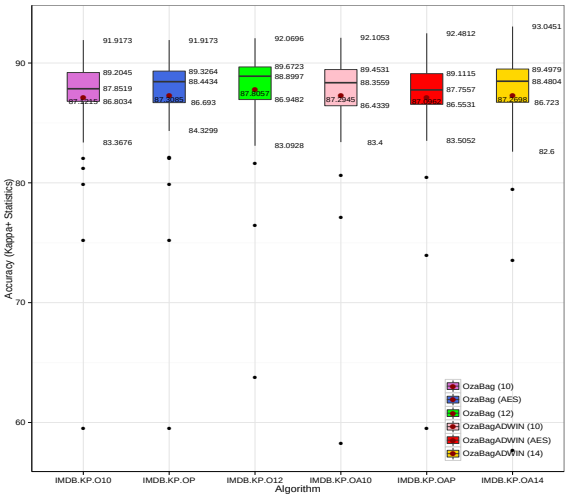


(c) Accuracy Graph Plot for Poker Dataset



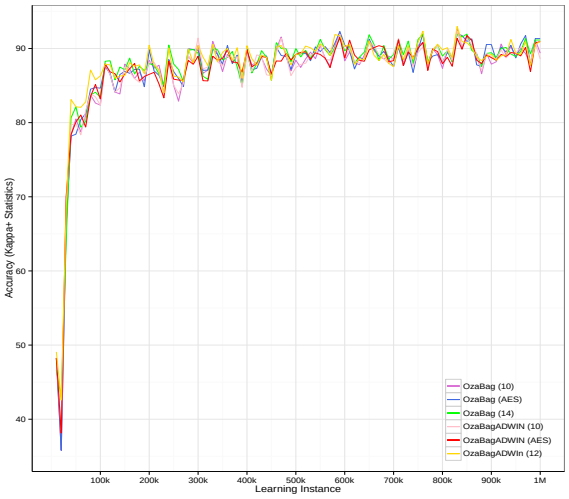
(d) Accuracy Box Plot for Poker Dataset

(e) Accuracy Graph Plot for IMDB Dataset

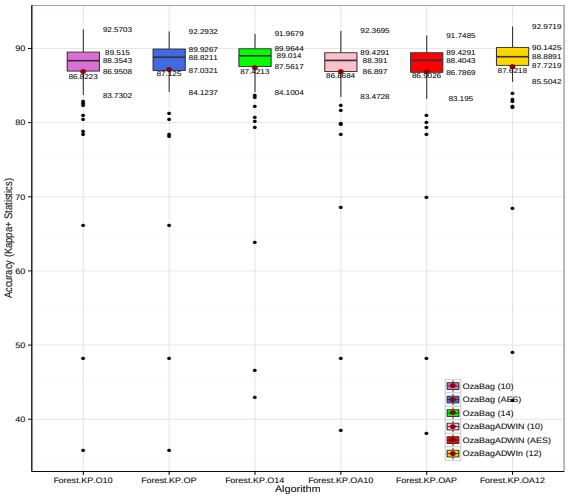


(f) Accuracy Box Plot for IMDB Dataset

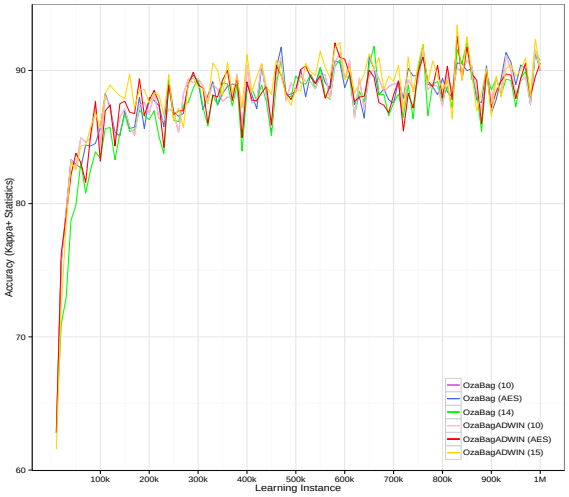
Figure 6.2: Accuracy Graph and Box Plot for KDD, Poker and IMDB Datasets



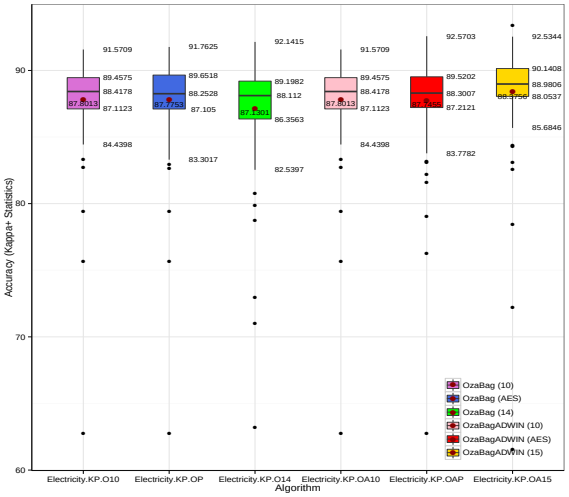
(a) Accuracy Graph Plot for Forest Dataset



(b) Accuracy Box Plot for Forest Dataset

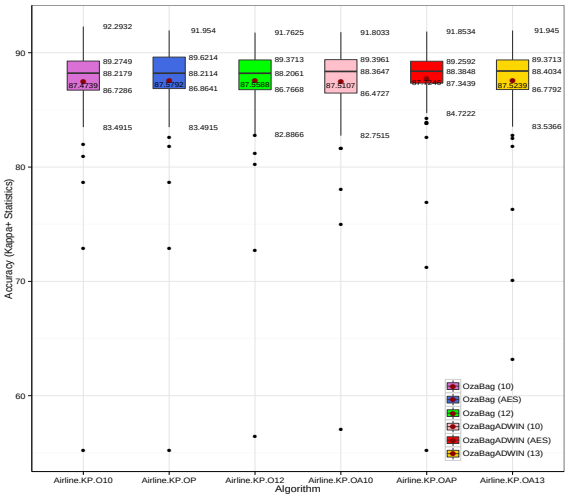


(c) Accuracy Graph Plot for Electricity Dataset



(d) Accuracy Box Plot for Electricity Dataset

(e) Accuracy Graph Plot for Airline Dataset



(f) Accuracy Box Plot for Airline Dataset

Figure 6.3: Accuracy Graph and Box Plot for Forest, Electricity and Airline Datasets

the best minimum setting, with consideration to resources, and it adapts with changes in concept drift to obtain a better result.

6.1.3 Statistical Significance of Accuracy Results

Friedman's test is used to test for statistical significance of our accuracy results. As was explained in session 5.6.4, a threshold value $\alpha = 0.05$ is used as our significance level, and is compared to the p values obtained from the Friedman's test. Table 6.2 shows the various results obtained. We noticed that only the Airline dataset satisfies the null hypothesis, H_0, with a p value greater than α. This means than the performance of all the tested algorithms on the airline dataset does not have any significant difference in terms of classification accuracy. For the remaining five datasets, the null hypothesis is rejected as the value of p is significantly less than α. For each of these datasets, a post-hoc test is used to show how they perform in comparison with each other.

Table 6.2: Result of Friedman's Test

	KDD'99			Poker			IMDb			Forest			Electricity			Airline	
χ_F^2	33.6999			4.4648			4.0808			11.2370			15.0168			1.3113	
p	< .00001			0.00055			0.00135			< 0.00001			< 0.00001			0.25775	
	e.s	p	r	e.s	p	r	e.s	p	r	e.s	p	r	e.s	p	r	e.s	p
OzaBag	10	**2.3900**	1	10	3.6200	3	10	**3.0333**	1	10	3.0250	2	10	3.4650	2	10	-
OzaBag(AES)	14	3.4200	3	15	3.7150	4	12	3.6667	5	14	3.5450	4	14	3.5050	3	12	-
OzaBag	14	3.9200	5	15	3.4500	2	12	4.3333	6	14	4.1500	5	14	**2.4850**	1	12	-
OzaBagADWIN	10	3.7800	4	10	3.6200	3	10	3.3833	3	10	3.0550	3	10	3.4650	2	10	-
OzaBagADWIN(AES)	12	2.5850	2	14	**2.7750**	1	14	3.1250	2	12	**2.9650**	1	15	3.5200	4	13	-
OzaBagADWIN	12	4.9050	6	14	3.8200	5	14	3.4583	4	12	4.2600	6	15	4.5600	5	13	-

⋆ e.s = ensemble size; p=post-hoc test; r = ranking (lower is better)

The performance of each classifier is ranked by how significantly different they are to each other. Because each classifier can be significantly different from one or more other classifiers, the post-hoc test does the ranking and assign the appropriate mean rank to each of the classifiers based on how well it competes.

6.1.4 Memory Comparison

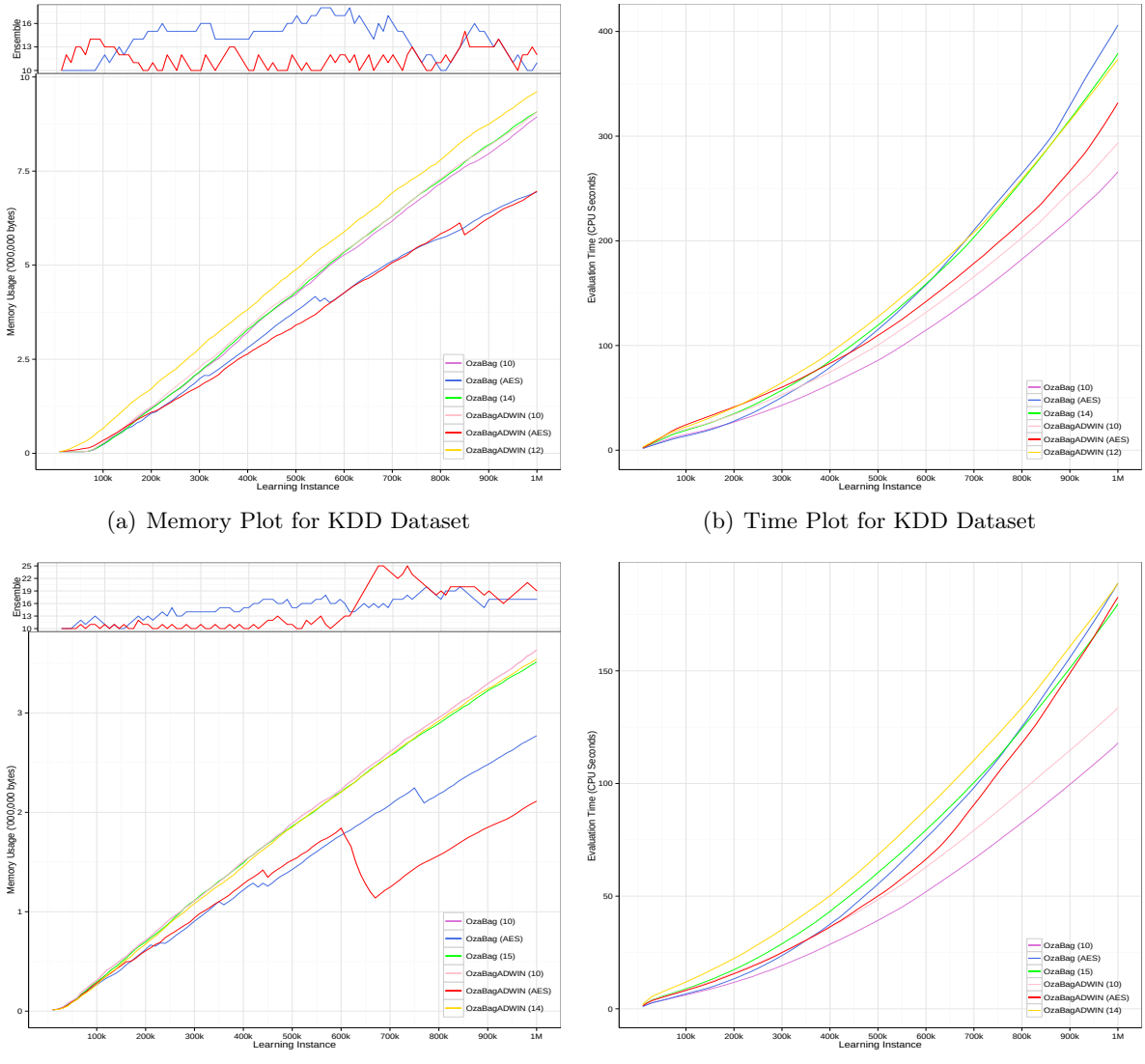
The memory utilized by the various algorithms configurations is shown table 6.3.

Table 6.3: Memory Utilized in Building the Models (M'bytes)

	KDD'99			Poker			IMDb			Forest			Electricity			Airline		
	e.s	Memory	r	e.s	Memory	r	e.s	Memory	r	e.s	Memory	r	e.s	Memory	r	e.s	Memory	r
OzaBag	10	8.9433	3	10	3.6345	4	10	133.0000	4	10	7.9579	4	10	3.7689	5	10	4.0047	5
OzaBag(AES)	14	**6.9464**	1	15	2.7709	5	12	103.0000	2	14	**5.2276**	1	14	2.6238	2	12	2.9499	2
OzaBag	14	9.0757	4	15	3.5128	2	12	132.0000	3	14	7.9105	3	14	3.7561	3	12	4.0673	6
OzaBagADWIN	10	9.0772	5	10	3.6345	4	10	133.0000	4	10	7.9846	5	10	3.7689	5	10	3.8153	4
OzaBagADWIN(AES)	12	6.9682	2	14	**2.1129**	1	14	**91.6000**	1	12	5.6900	2	15	**2.2953**	1	13	**2.7673**	1
OzaBagADWIN	12	9.6077	6	14	3.5436	3	14	132.0000	3	12	8.0030	6	15	3.7573	4	13	3.7390	3

⋆ e.s = ensemble size; r = ranking (lower is better)

As can be seen in the table, our AES implementations performs better than all the other algorithms, in terms of memory utilization. We also provide the visual representation of this results, with respect to the instances. One major observation in both implementation of AES is the occasional reduction in memory usage by these algorithms, as can be seen in figures 6.4(a), 6.4(c), 6.5(a), 6.5(c), 6.5(e), and 6.6(a). It can be observed that this drop in memory usually occurs when there is a significant change in ensemble size, compared relatively to the last set of sizes. This change, as observed, can be either an increase or decrease in ensemble size.



(a) Memory Plot for KDD Dataset

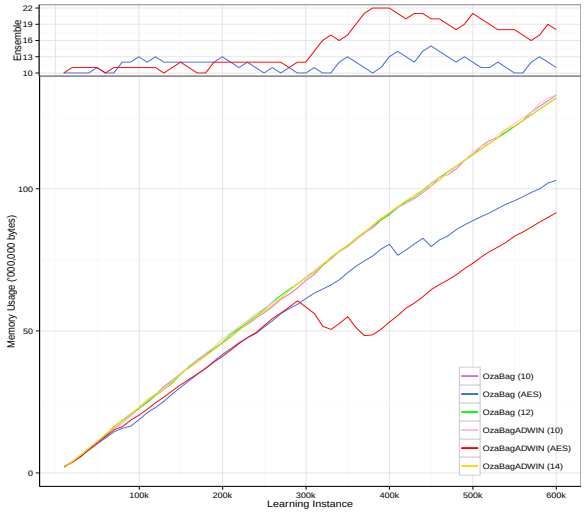
(b) Time Plot for KDD Dataset

(c) Memory Plot for Poker Dataset

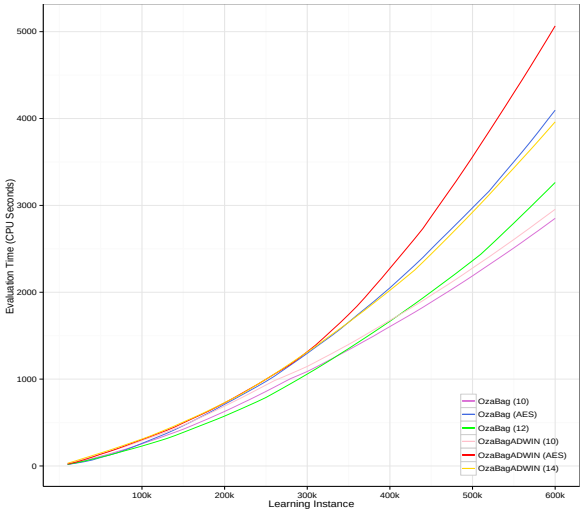
(d) Time Plot for Poker Dataset

Figure 6.4: Memory and Time Plot for KDD and Poker Datasets

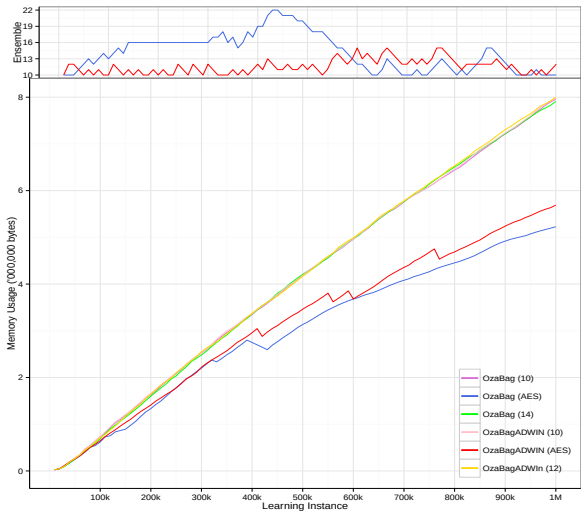
The figures also indicate that these lower memory usages are not depended on this reduction, as the linear trajectory of both of our implementation is still lower than the other algorithms before any changes. The memory graph for both AES implementations



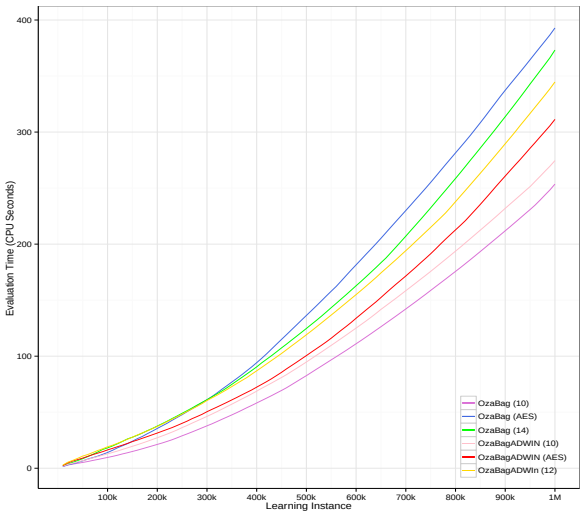
(a) Memory Plot for IMDB Dataset



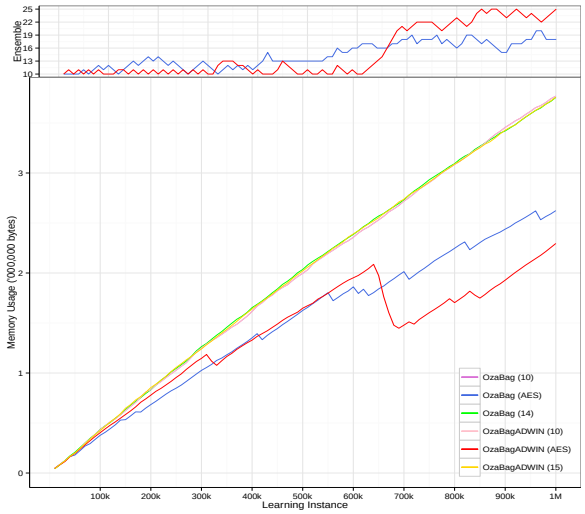
(b) Time Plot for IMDB Dataset



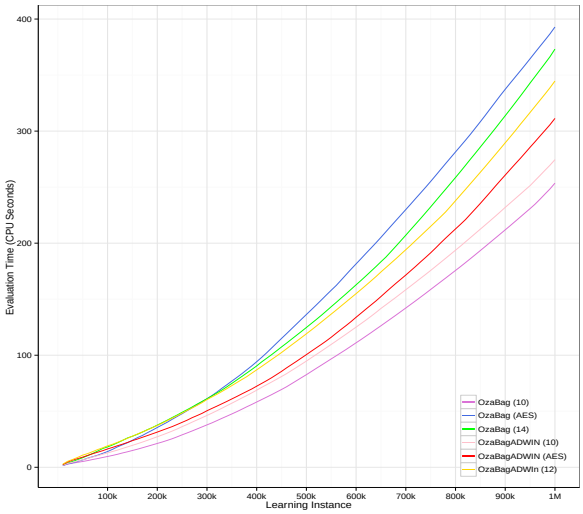
(c) Memory Plot for Forest Dataset



(d) Time Plot for Forest Dataset

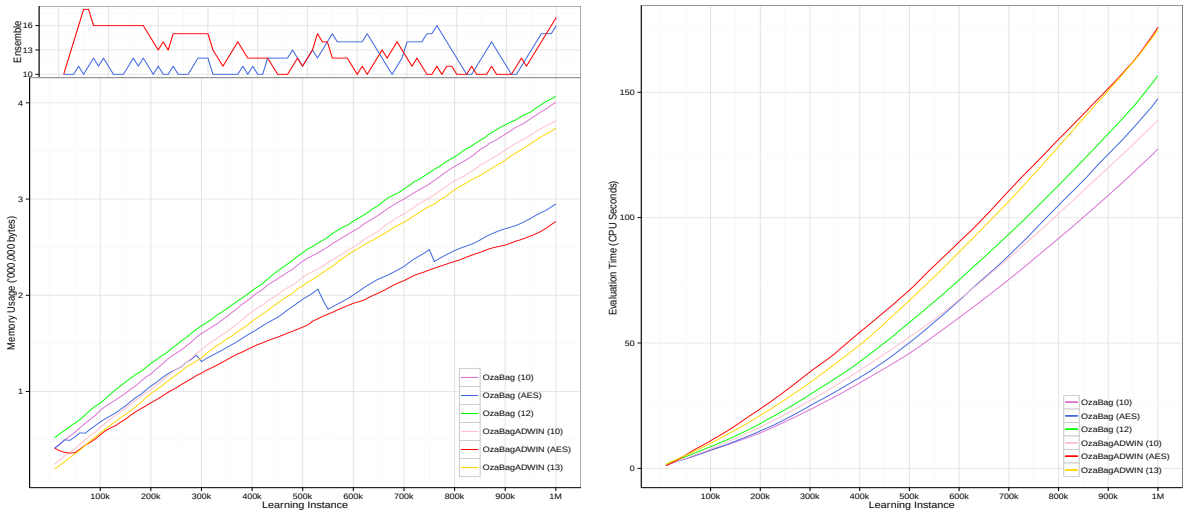


(e) Memory Plot for Electricity Dataset



(f) Time Plot for Electricity Dataset

Figure 6.5: Memory and Time Plot for IMDB, Forest and Electricity Datasets



(a) Memory Plot for Airline Dataset

(b) Time Plot for Airline Dataset

Figure 6.6: Memory and Time Plot for Airline Dataset

can be seen to be lower than the other algorithms they were compared with. This signifies that our implementation was able to minimize memory usage, in comparison to the other algorithms. As we will see in discussion about the overall cost consideration, the memory usage plays a very important role, and any savings we can get is important in the cost saving process. We believe that the insight obtained here can be better harnessed for future improvements.

6.1.5 Time Comparison

We provide the time taken by the different algorithms to construct the models in table 6.4.

Table 6.4: Time Taken to Build the Models

	KDD'99			Poker			IMDb			Forest			Electricity			Airline		
	e.s	Time	r	e.s	Time	r	e.s	Time	r	e.s	Time	r	e.s	Time	r	e.s	Time	r
OzaBag	10	**266.0597**	1	10	**117.9680**	1	10	**2851.0899**	1	10	**253.5640**	1	10	**117.3284**	1	10	**127.4996**	1
OzaBag(AES)	14	406.1642	6	15	188.9172	6	12	4096.3367	5	14	392.9041	6	14	167.5451	4	12	147.4365	3
OzaBag	14	379.1604	5	15	179.6352	3	12	3263.7749	3	14	373.1856	5	14	167.2487	3	12	156.6250	4
OzaBagADWIN	10	293.6407	2	10	133.6773	2	10	2954.9397	2	10	274.5306	2	10	131.7116	2	10	138.8097	2
OzaBagADWIN(AES)	12	332.0013	3	14	182.6928	4	14	5066.2105	6	12	311.3468	3	15	170.4935	5	13	175.9379	6
OzaBagADWIN	12	373.4664	4	14	188.8548	5	14	3961.7234	4	12	344.6998	4	15	201.9589	6	13	174.9707	5

⋆ e.s = ensemble size; r = ranking (lower is better)

The table shows that the OzaBag algorithm, at the default ensemble size of 10 gives the best time across all datasets for the same number of instances. This is expected, as the OzaBag algorithm with ensemble size of 10 takes the minimal processing effort, and obtains the worst accuracy results in all of our comparison. What is, however, not expected is the performance of our implementation when compared with the alternative setting of the average ensemble size. We found out that it compares better in some instances. Comparing both implementations of AES with the alternate ensemble size across all six

datasets, as we can see from table 6.4, the AES algorithms ranked better in five instances: OzaBagADWIN(AES) with the KDD, Poker, Forest and Electricity datasets; and OzaBag with the Airline dataset.

Figures 6.4(b), 6.4(d), 6.5(b), 6.5(d), 6.5(f), and 6.6(b) shows the visual representation of the change in processing time with respect to the instance, for the datasets.

6.1.6 ROI and Results Discussion

Table 6.5 shows the $\overline{ROI}$ for the six datasets. In all of the datasets, our algorithms have the best results, with OzaBag(AES) performing best, followed by OzaBagADWIN(AES). This shows that for all the datasets in our experimentation, our implementation of both AES algorithms has better ROI than the fixed ensemble size setting. This means that the AES algorithms are more efficient, in terms of cost of computing, without compromising on the classification accuracy.

Table 6.5: ROI Comparison between the Algorithms

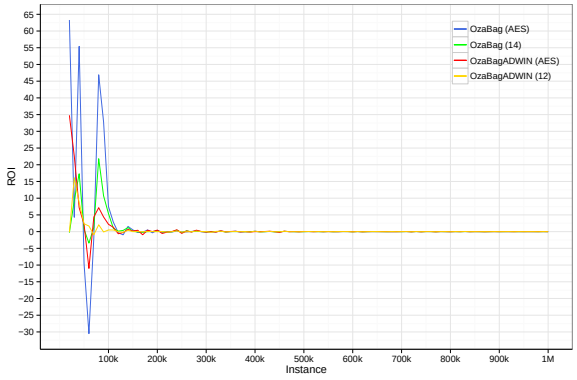
	KDD'99			Poker			IMDb			Forest			Electricity			Airline		
	e.s	ROI	r	e.s	ROI	r	e.s	ROI	r	e.s	ROI	r	e.s	ROI	r	e.s	ROI	r
OzaBag(AES)	14	**1.7207**	1	15	**4.0195**	1	12	**1.3558**	1	14	**0.6184**	1	14	**1.2899**	1	12	**0.3871**	1
OzaBag	14	0.6557	3	15	2.2341	3	12	0.7991	3	14	0.4436	3	14	0.6790	3	12	0.1988	4
OzaBagADWIN(AES)	12	0.7526	2	14	3.4481	2	14	0.8425	2	12	0.3732	4	15	1.1231	2	13	0.2344	3
OzaBagADWIN	12	0.3103	4	14	1.9285	4	14	0.5285	4	12	0.4511	2	15	0.5813	4	13	0.2477	2

⋆ e.s = ensemble size; r = ranking (lower is better)

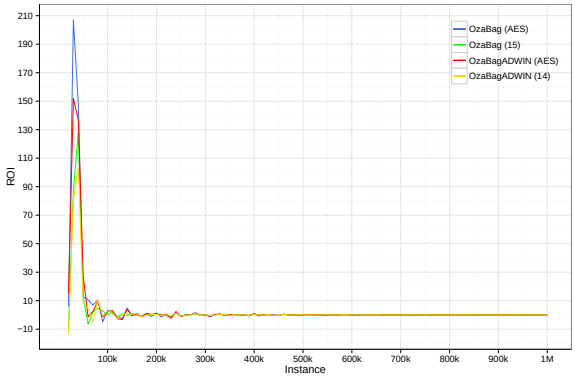
Figure 6.7 shows the graph of the ROI at each step of the mining process. We can notice a surge in ROI at the start for each of the datasets. This can be attributed to the start of the evaluation process, when the accuracy is building up. Recall that we are using the prequential, or test then train, evaluation method. This indicates that there is a rapid build up in accuracy at this point, before it plateaus. The surge quickly reduces, and the ROI becomes close to constant for the rest of the stream. From all of the graphs, it is clear that our algorithms performs the best, in terms of ROI. We can see the line representing OzaBag(AES) achieving the greatest ROI. The graphs also show that OzaBagADWIN(AES) has the second highest utility.

6.2 Synthesis and Lessons Learned

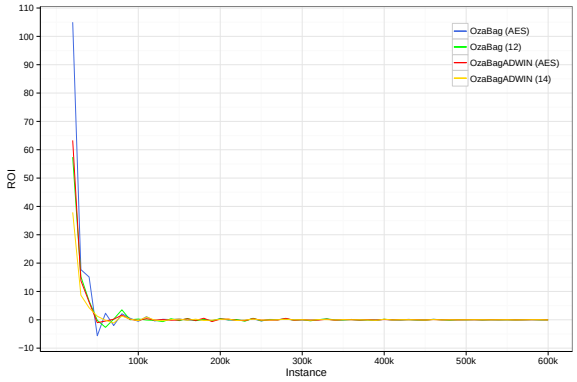
The goal of this experimentation was to improve on the results obtained when mining in a stream with concept drift. With evidence from different literature, we showed that the combination of OzaBag Online Bagging, and Hoeffding tree algorithms are a good combination of meta learning and base learning algorithms. In our preliminary experimentation, we showed the relationship between the ensemble size and the accuracy



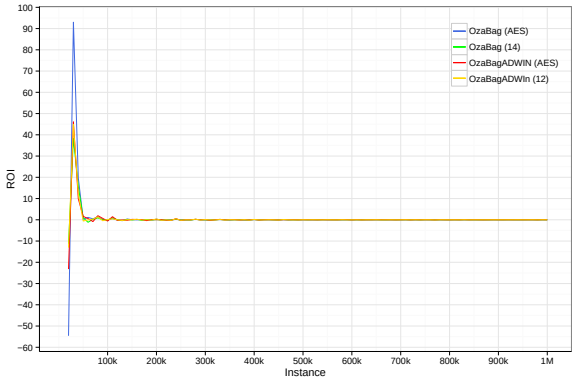
(a) ROI Plot for KDD Dataset



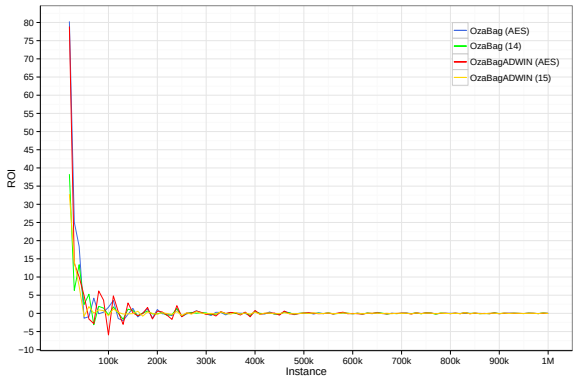
(b) ROI Plot for Poker Dataset



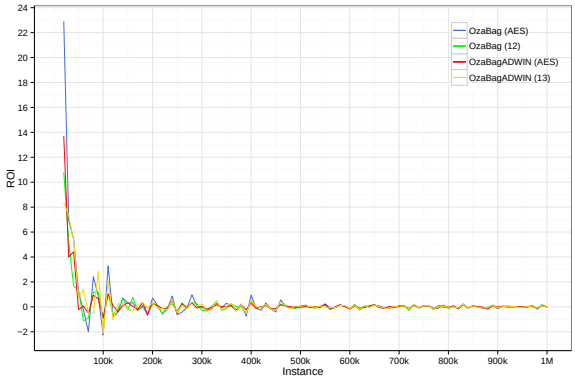
(c) ROI Plot for IMDB Dataset



(d) ROI Plot for Forest Dataset



(e) ROI Plot for Electricity Dataset



(f) ROI Plot for Airline Datasets

Figure 6.7: ROI Plot for the Datasets

result of the classifier. We found that, although a higher ensemble size usually result in a higher accuracy, it is not always better, as the computational cost of the mining process starts to increase exponentially for ensemble sizes between 20 and 30.

For this reasons, we implemented the Adaptive Ensemble Size (AES) approach that varies the size of the ensemble. Our aim for using this approach is to be able to improve on the accuracy, while deriving a high utility during the mining process. We started the experimentation in this chapter by showing that the change in memory utilized by the mining algorithm is proportion to the magnitude of the concept drift. The AES approach takes advantage of this memory change induced by the concept drift to adapt the size of the ensemble during the mining process. The results of this implementation can be seen from section 6.1.2 to 6.1.6. A summary of these results, ranked by how they compare in terms of ROI and k^+ accuracy is shown in table 6.6. The lower the rank, the better.

Table 6.6: Summary of ROI and Accuracy of Algorithms

	KDD'99			Poker			IMDb			Forest			Electricity			Airline		
	e.s	R	k^+	e.s	R	k^+	e.s	R	k^+	e.s	R	k^+	e.s	R	k^+	e.s	R	k^+
OzaBag(AES)	14	1	2	15	1	2	14	1	3	14	1	3	14	1	3	12	1	
OzaBag	14	3	3	15	3	3	14	3	1	14	3	1	14	3	4	12	4	
OzaBagADWIN(AES)	12	2	4	14	2	4	14	2	4	12	4	4	15	2	2	13	3	
OzaBagADWIN	12	4	1	14	4	1	14	4	2	12	2	2	15	4	1	13	2	

⋆ e.s = ensemble size; R = ROI

In the summary table shown above, we rank how best each algorithm performed with each of the datasets, based on the utility gained from building models (ROI), and the k^+ accuracy of the results obtained. Recall that the value of the ROI is derived from the time to process, memory and k^+. We used the Kappa Plus Statistics (k^+) for accuracy. As explained in section 5.6.2, k^+ is a good metric for measuring accuracy in a stream with concept drift, because of the temporal nature of such data. The table represents a complete summary of all the results obtained in our final experimentation. The accuracy ranking of the Airline dataset is omitted from this table, because the results obtained are equivalent, according to Friedman's test performed in section 6.1.3.

We are making a comparison between our implementations, and both versions of OzaBag and OzaBagADWIN. The ensemble sizes of both algorithms are set to the mean ensemble sizes obtained from our implementations. We omitted the default size of ensemble size 10 from these comparison because our goal is to be able to achieve higher accuracy. In section 4.4.2, we have already established the fact that for ensemble sizes within 10-25, the higher the ensemble size, the higher the accuracy of the result obtained. This, however, is resource intensive in terms of memory utilization and time, hence the reason for computing the ROI.

We implemented the AES method as a middle ground for having a high cost of building the models, and the accuracy obtained. This means that we can make a decision not to have a larger ensemble size if it only gets a relatively small increase in accuracy at a higher overhead. For this reason, the AES approach adapts the ensemble size only when required. One major observation is that ADWIN does not have a positive effect on the AES algorithms, as the OzaBag implementation performs better than OzaBagADWIN for the KDD'99, Poker, IMDb and Forest datasets in terms of k^+ accuracy.

However, our implementations were the best for all of the datasets in terms of ROI. As can be seen from the summary table, OzaBag(AES) has the best ROI for all of the tests, followed by OzaBagADWIN(AES). Recall that our implementations were able to build the models with the least memory utilized, with comparable accuracy results obtained. Considering that the k^+ accuracy is between 2nd and 3rd, we consider this results to be good, in line with what we set out to achieve.

Our implementation shows an important direction, and a proof of concept. We have already established the savings that we were able to get from the lower memory utilized by the AES algorithms. To make this translate to even better ROI, however, improvements has to be made to these algorithms to save on time. From tables 6.3 and 6.1.5, we can see that our implementation consumes less memory for all of the datasets, but also takes the longest time to process. The reason for taking a longer time to process is because of the necessary computation required for adapting the size of the ensemble. Ways to reduce the time taken will be explored in future research. Methods to improvement on the accuracy will also be explored.

We noticed that ADWIN has a negative effect on our implementation, in most cases. This is counter to what we expected, considering the improvement ADWIN adds to OzaBag as a drift detector. This negative effect appears to be due to the fact that this approach utilizes different ways of adapting to concept drift. Instead of the cost consideration that we have used in this thesis, an alternate implementation is to employ ADWIN's drift detecting method as a trigger for the ensemble size adaptation. This will keep the response to the drift synchronized. Also, considering the fact that ADWIN responds much better to the changes induced by concept drift, this approach may further improve on the results obtained. This can also be combined with consideration to the cost. Exploring these options should enable us get a better result in terms of accuracy.

6.3 Conclusion

In this chapter, we have provided the various results obtained in our experimentation. We started our experimentation with the aim of obtaining a better accuracy when learning from a stream with concept drift, without compromising on utility. We were able to get better accuracy than the default, in most instances, but at a longer processing time, even though the memory utilized during our implementation is significantly better in all cases. However, the time factor does not have a significant effect on the utility, as we were able to get the best ROI across all datasets used in this research.

In the next chapter, we provide a general conclusion to this thesis, as well as our contribution and future work.

CHAPTER 7

Conclusions

In this thesis, we focused on improving the utility derived from mining in a stream with concept drift. We aimed to achieve high ROI without compromising on the accuracy of the resultant models. We conducted an experimental approach in order to extend the Online Bagging algorithm. We performed an experimentation to observe the impact of concept drift on the memory usage. We also observed the impact of different ensemble sizes on the prediction error, and on the computational cost (in terms of memory utilization per hour). This chapter provides an overall conclusion to the thesis. We discuss our contributions, as well as future work.

7.1 Thesis Contributions

There were a few early research works in Utility-Based Data Mining (UBDM) in the early 2000s. However, studying utility and efficiency considerations in data stream mining is quite new. We foresee an expansion of interest in this area, because of the infinitely expanding stream of data and the cost associated with available resources. While a number of work exists in data stream mining research related to improving algorithm results in a stream with concept drift, our main contribution is the idea of varying the ensemble size based on the memory utilized as a result of concept drift. This is done with consideration to efficiency and utility; specifically to reduce the computational cost of mining in the stream, while increasing the accuracy of the result obtained. We introduced the Adaptive Ensemble Size (AES) algorithms as an implementations of this idea.

Our implementation of the AES algorithm sets a minimum and maximum values, as the range of the ensemble size during the process of building the models. By doing this adaptation, we are able to obtain highly accurate models. While having a fixed ensemble size based of the average size obtained from the AES implementation obtains better accuracy in most cases, we showed that the results obtained by our approach is statistically significant, hence, it should be considered. The higher ensemble size will also costs more (computationally), since there is no intelligence to the ensemble size, with regards to the drift in concept in the stream.

Considering the fact that the best ensemble size is not usually known before hand, by utilizing this technique, we are able to find a balance between cost and accuracy. We believe that this can be combined with many state-of-the-art ensemble learning algorithms, in order to make improvements to their results. When compared with default fixed ensemble size settings, our implementation achieved the best ROI in all of our test cases.

7.2 Future Work

While our implementation had good results in terms of ROI, we believe that we can get better result by improving on the time it take to process. One possible way to achieve this will be by training base learners in parallel, instead of looping through the ensemble. A weighted average can then be used to update the model before the next iteration is done. It is not clear what effect this might have on the overall accuracy obtained, but it is something we will consider for future research.

We discussed cost-sensitive adaptation in chapter 2, and we also looked at how the utility of the model is a factor for the cost assigned to the mining task. As an idea for future work, specialized domain knowledge can be incorporated into the mining cycle, such that special cases are given extra attention and required resource. This could be in the form of active learning, with a professional health official being the oracle. In this way, a higher accuracy may be obtained, indicating better confidence in the interpretation of the result.

We did not discuss other factors that can also affect cost-sensitive adaptation, such as the device in use (is it a mobile phone or a desktop computer?), and computing environmental issues (are we working on a cloud computer, a virtual computer or a desktop computer?). Because of these very different working environments, we cannot simply develop a one-size fit-all model, and expect the adaptation to work in a similar fashion. However, it will be very useful if the algorithm can adapt to to the environment before

building the model. In this way, we are guaranteed of a best-case result for the environment the model is built from.

In the implemented algorithms, we made an assumption that the change in memory utilization is as a result of concept drift. There are, however, other factors that are not currently been considered, such as the noise in the stream. It will be important to take this into consideration. The reactions of the utilization cost to noise, and how to harness this to improve accuracy is another area of research worth considering. We intend to explore this by experimenting with ways of minimizing the effect of noise in the stream. We will also consider the noise ratio before processing any stream window, so that we can get the best result possible in that processing cycle, at the least cost.

Another area to be explored for future research is the memory response to the ensemble size adaptation. While we were expecting some savings in terms of memory utilization as a result of our implementation, we did not foresee the occasional memory drop. Recall that in section 6.1.4, we observed further drop in memory when there is a significant change in ensemble size. Although the general memory usage trajectory of our implementation was lower than the other algorithms, this occasional drop positively affect the ROI. The cause of this is unknown at this time, and we plan to explore it further to give better insight into making future improvements in memory consumption.

Bibliography

ADAMS. Adams - the advanced data mining and machine learning system. https://adams.cms.waikato.ac.nz/about.html. (Visited on 01/19/2015).

Charu Aggarwal. *Data Streams: Models and Algorithms*, volume 31 of *Advances in Database Systems*. Springer-Verlag, Berlin, 2007.

Charu C. Aggarwal and Jiawei Han, editors. *Frequent Pattern Mining*. Springer, 2014. ISBN 978-3-319-07820-5. doi: 10.1007/978-3-319-07821-2. URL http://www.springer.com/978-3-319-07820-5.

Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 81–92. VLDB Endowment, 2003. ISBN 0-12-722442-4. URL http://dl.acm.org/citation.cfm?id=1315451.1315460.

Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 852–863. VLDB Endowment, 2004a. ISBN 0-12-088469-0. URL http://dl.acm.org/citation.cfm?id=1316689.1316763.

Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. On demand classification of data streams. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 503–508, New York, NY, USA, 2004b. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014110. URL http://doi.acm.org/10.1145/1014052.1014110.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993. ISSN 0163-5808. doi: 10.1145/170036.170072. URL http://doi.acm.org/10.1145/170036.170072.

Rabaa Alabdulrahman. A comparative study of ensemble active learning. Master's thesis, University of Ottawa, Ottawa, Canada, 2014.

Rocío Alaiz-Rodríguez. Local decision bagging of binary neural classifiers. In Sabine Bergler, editor, *Advances in Artificial Intelligence*, volume 5032 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-68821-1. doi: 10.1007/978-3-540-68825-9_1. URL http://dx.doi.org/10.1007/978-3-540-68825-9_1.

Amazon. Aws — amazon ec2 — pricing. http://aws.amazon.com/ec2/pricing/, 08 2014.

Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM. ISBN 1-58113-507-6. doi: 10.1145/543613.543615. URL http://doi.acm.org/10.1145/543613.543615.

Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, September 2001. ISSN 0163-5808. doi: 10.1145/603867.603884. URL http://doi.acm.org/10.1145/603867.603884.

J. Bakker, M. Pechenizkiy, I. Žliobaitė, A. Ivannikov, and T. Kärkkäinen. Handling outliers and concept drift in online mass flow prediction in cfb boilers. In *Proceedings of the Third International Workshop on Knowledge Discovery from Sensor Data*, SensorKDD '09, pages 13–22, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-668-7. doi: 10.1145/1601966.1601972. URL http://doi.acm.org/10.1145/1601966.1601972.

Rongfang Bie, Xin Jin, Chuanliang Chen, Chuan Xu, and Ronghuai Huang. Meta learning intrusion detection in real time network. In JoaquimMarques Sá, LuísA. Alexandre, Włodzisław Duch, and Danilo Mandic, editors, *Artificial Neural Networks – ICANN 2007*, volume 4668 of *Lecture Notes in Computer Science*, pages 809–816. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74689-8. doi: 10.1007/978-3-540-74690-4_82. URL http://dx.doi.org/10.1007/978-3-540-74690-4_82.

Albert Bifet and Eibe Frank. Sentiment knowledge discovery in twitter streaming data. In *Proceedings of the 13th International Conference on Discovery Science*, DS'10, pages 1–15, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16183-9, 978-3-642-16183-4. URL http://dl.acm.org/citation.cfm?id=1927300.1927301.

Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. In *SDM*, pages 443–448, 2007.

Albert Bifet and Ricard Gavaldà. Adaptive learning from evolving data streams. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII*, IDA '09, pages 249–260, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03914-0. doi: 10.1007/978-3-642-03915-7_22. URL http://dx.doi.org/10.1007/978-3-642-03915-7_22.

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Ricard Gavaldà. Improving adaptive bagging methods for evolving data streams. In Zhi-Hua Zhou and Takashi Washio, editors, *Advances in Machine Learning*, volume 5828 of *Lecture Notes in Computer Science*, pages 23–37. Springer Berlin Heidelberg, 2009a. ISBN 978-3-642-05223-1. doi: 10.1007/978-3-642-05224-8_4. URL http://dx.doi.org/10.1007/978-3-642-05224-8_4.

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, pages 139–148, New York, NY, USA, 2009b. ACM. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557041. URL http://doi.acm.org/10.1145/1557019.1557041.

Albert Bifet, Eibe Frank, Geoffrey Holmes, and Bernhard Pfahringer. Accurate ensembles for data streams: Combining restricted hoeffding trees using stacking. In *Proceedings of the 2nd Asian Conference on Machine Learning, ACML 2010, Tokyo, Japan, November 8-10, 2010*, pages 225–240, 2010a. URL http://www.jmlr.org/proceedings/papers/v13/bifet10a.html.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, August 2010b. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1859903.

Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. In *ECML/PKDD*, pages 135–150, 2010c.

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Eibe Frank. Fast perceptron decision tree learning from evolving data streams. In *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, PAKDD'10, pages 299–310, Berlin, Heidelberg, 2010d. Springer-Verlag. ISBN 3-642-13671-0, 978-3-642-13671-9. doi: 10.1007/978-3-642-13672-6_30. URL http://dx.doi.org/10.1007/978-3-642-13672-6_30.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. *Data Stream Mining - A Practical Approach*. COSI, 2011.

Albert Bifet, Richard Kirkby, Philipp Kranen, and Peter Reutemann. Massive online analysis: Manual. March 2012. URL http://moa.cms.waikato.ac.nz/documentation/.

Albert Bifet, Jesse Read, Indrė Žliobaitė, Bernhard Pfahringer, and Geoff Holmes. Pitfalls in benchmarking data stream classification and how to avoid them. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Zelezný, editors, *ECML/PKDD (1)*, volume 8188 of *Lecture Notes in Computer Science*, pages 465–479. Springer, 2013. ISBN 978-3-642-40987-5. URL http://dblp.uni-trier.de/db/conf/pkdd/pkdd2013-1.html#BifetRZPH13.

David M. Blei and John D. Lafferty. Dynamic topic models. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 113–120, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143859. URL http://doi.acm.org/10.1145/1143844.1143859.

Richard J. Bolton and David J. Hand. Statistical fraud detection: A review. *Statist. Sci.*, 17(3):235–255, 08 2002. doi: 10.1214/ss/1042727940. URL http://dx.doi.org/10.1214/ss/1042727940.

Pavel Brazdil, Christophe Giraud-Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1 edition, 2009. ISBN 3540732624, 9783540732624.

Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, August 1996. ISSN 0885-6125. doi: 10.1023/A:1018054314350. URL http://dx.doi.org/10.1023/A:1018054314350.

Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.

D. Brzezinski and J. Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(1):81–94, Jan 2014a. ISSN 2162-237X. doi: 10.1109/TNNLS.2013.2251352.

D. Brzezinski and J. Stefanowski. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(1):81–94, Jan 2014b. ISSN 2162-237X. doi: 10.1109/TNNLS.2013.2251352.

Yun Chi, Haixun Wang, and Philip S. Yu. Loadstar: Load shedding in data stream mining. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 1302–1305. VLDB Endowment, 2005. ISBN 1-59593-154-6. URL http://dl.acm.org/citation.cfm?id=1083592.1083757.

William Jay Conover. *Practical nonparametric statistics*. Wiley series in probability and statistics. Wiley, New York, NY [u.a.], 3. ed edition, 1999. ISBN 0471160687. URL http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+24551600X&sourceid=fbw_bibsonomy.

Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 335–345. VLDB Endowment, 2002. URL http://dl.acm.org/citation.cfm?id=1287369.1287399.

Michael Cowles and Caroline Davis. On the origins of the .05 level of statistical significance. *American Psychologist*, 37(5):553–558, May 1982. doi: 0003-066X/82/3705-0553$00.75.

D. R. (David Roxbee) Cox and D Oakes. *Analysis of survival data*. London ; New York : Chapman and Hall, 1984. ISBN 041224490X. Includes indexes.

Fernando Crespo and Richard Weber. A methodology for dynamic data mining based on fuzzy clustering. *Fuzzy Sets and Systems*, 150(2):267 – 284, 2005. ISSN 0165-0114. doi: http://dx.doi.org/10.1016/j.fss.2004.03.028. URL http://www.sciencedirect.com/science/article/pii/S016501140400140X.

Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows: (extended abstract). In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 635–644, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics. ISBN 0-89871-513-X. URL http://dl.acm.org/citation.cfm?id=545381.545466.

Christopher Day. Chapter 26 - intrusion prevention and detection systems. In John R. Vacca, editor, *Computer and Information Security Handbook (Second Edition)*, pages 485 – 498. Morgan Kaufmann, Boston, second edition edition, 2013. ISBN 978-0-12-394397-2. doi: http://dx.doi.org/10.1016/B978-0-12-394397-2.00026-X. URL http://www.sciencedirect.com/science/article/pii/B978012394397200026X.

Erico N. de Souza and Stan Matwin. Improvements to boosting with data streams. In OsmarR. Zaïane and Sandra Zilles, editors, *Advances in Artificial Intelligence*, volume 7884 of *Lecture Notes in Computer Science*, pages 248–255. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38456-1. doi: 10.1007/978-3-642-38457-8_23. URL http://dx.doi.org/10.1007/978-3-642-38457-8_23.

Fan Deng. *Approximation Algorithms for Frequency Related Query Processing on Streaming Data*. PhD thesis, Edmonton, Alta., Canada, 2007. AAINR32950.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM. ISBN 1-58113-233-6. doi: 10.1145/347090.347107. URL http://doi.acm.org/10.1145/347090.347107.

J. Ekanayake, J. Tappolet, H.C. Gall, and A. Bernstein. Tracking concept drift of software projects using defect prediction quality. In *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, pages 51–60, May 2009. doi: 10.1109/MSR.2009.5069480.

Conny Franke. *Adaptivity in Data Stream Mining*. PhD thesis, 2008.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997. ISSN 0022-0000. doi: 10.1006/jcss.1997.1504. URL http://dx.doi.org/10.1006/jcss.1997.1504.

MohamedMedhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Towards an adaptive approach for mining data streams in resource constrained environments. In Yahiko Kambayashi, Mukesh Mohania, and Wolfram Wöß, editors, *Data Warehousing and Knowledge Discovery*, volume 3181 of *Lecture Notes in Computer Science*, pages 189–198. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22937-7. doi: 10.1007/978-3-540-30076-2_19. URL http://dx.doi.org/10.1007/978-3-540-30076-2_19.

MohamedMedhat Gaber, Shonali Krishnaswamy, and Arkady Zaslavsky. On-board mining of data streams in sensor networks. In *Advanced Methods for Knowledge Discovery from Complex Data*, Advanced Information and Knowledge Processing, pages 307–335. Springer London, 2005. ISBN 978-1-85233-989-0. doi: 10.1007/1-84628-284-5_12. URL http://dx.doi.org/10.1007/1-84628-284-5_12.

Francis Galton. Co-relations and their measurement, chiefly from anthropometric data. In *Proceedings of the Royal Society of London*, pages 135–145, 1888.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, March 2014. ISSN 0360-0300. doi: 10.1145/2523813. URL http://doi.acm.org/10.1145/2523813.

Joao Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, 1st edition, 2010. ISBN 1439826110, 9781439826119.

João Gama, Raquel Sebastião, and PedroPereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013. ISSN 0885-6125. doi: 10.1007/s10994-012-5320-9. URL http://dx.doi.org/10.1007/s10994-012-5320-9.

John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. Technical report, IDC iView, December 2012. URL http://idcdocserv.com/1414.

Salvador García, Julián Luengo, and Francisco Herrera. *Data Preprocessing in Data Mining*. Springer International Publishing, 2015. ISBN 978-3-319-10246-7. doi: 10.1007/978-3-319-10247-4. URL http://link.springer.com/book/10.1007/978-3-319-10247-4.

Susan Gauch, Mirco Speretta, Aravind Chandramouli, and Alessandro Micarelli. The adaptive web. chapter User Profiles for Personalized Information Access, pages 54–89. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-72078-2. URL http://dl.acm.org/citation.cfm?id=1768197.1768200.

Raffaella Giacomini and Barbara Rossi. Detecting and Predicting Forecast Breakdowns. *Review of Economic Studies*, 76(2):669–705, 2009. URL http://ideas.repec.org/a/oup/restud/v76y2009i2p669-705.html.

Anna C. Gilbert, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. One-pass wavelet decompositions of data streams. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):541–554, March 2003. ISSN 1041-4347. doi: 10.1109/TKDE.2003.1198389. URL http://dx.doi.org/10.1109/TKDE.2003.1198389.

Google. Google cloud pricing - cloud platform products — google cloud platform. https://cloud.google.com/pricing/, 08 2014.

Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645 – 1660, 2013. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2013.01.010. URL http://www.sciencedirect.com/science/article/pii/S0167739X13000241.

Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley. Model selection: Beyond the bayesian/frequentist divide. *J. Mach. Learn. Res.*, 11:61–87, March 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1756009.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL http://doi.acm.org/10.1145/1656274.1656278.

Jiawei Han and Micheline Kamber. *Data Mining - Concepts and Techniques*. Morgan Kaufmann, 2nd ed. edition, 2006. ISBN 1558609016.

Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011. ISBN 0123814790, 9780123814791.

L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12 (10):993–1001, October 1990. ISSN 0162-8828. doi: 10.1109/34.58871. URL http://dx.doi.org/10.1109/34.58871.

Haibo He and Yunqian Ma. *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press, 1st edition, 2013. ISBN 1118074629, 9781118074626.

Constantinos S. Hilas. Designing an expert system for fraud detection in private telecommunications networks. *Expert Systems with Applications*, 36(9):11559 – 11569, 2009. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2009.03.031. URL http://www.sciencedirect.com/science/article/pii/S095741740900284X.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, March 1963. URL http://www.jstor.org/stable/2282952?

Elena Ikonomovska. Airline dataset. http://kt.ijs.si/elena_ikonomovska/data.html, 2011. (Visited on 01/20/2015).

Nathalie Japkowicz and Mohak Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011. doi: 10.1017/CBO9780511921803.

Christopher Jermaine, Sanjay Ranka, and lennox archibald. Data mining for multiple antibiotic resistance. 2008.

KDD99. Kdd cup 1999: Computer network intrusion detection, 2009. URL http://www.sigkdd.org/kdd-cup-1999-computer-network-intrusion-detection.

Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, January 1994. ISSN 0004-5411. doi: 10.1145/174644.174647. URL http://doi.acm.org/10.1145/174644.174647.

Taghi M. Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Comparing boosting and bagging techniques with noisy and imbalanced data. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 41(3):552–568, 2011. doi: 10.1109/TSMCA.2010.2084081. URL http://dx.doi.org/10.1109/TSMCA.2010.2084081.

Wael Khreich, Eric Granger, Ali Miri, and Robert Sabourin. A survey of techniques for incremental learning of hmm parameters. *Inf. Sci.*, 197:105–130, August 2012. ISSN 0020-0255. doi: 10.1016/j.ins.2012.02.017. URL http://dx.doi.org/10.1016/j.ins.2012.02.017.

Vijay Kotu and Bala Deshpande. *Predictive Analytics and Data Mining*. Morgan Kaufmann, Boston, 2015. ISBN 978-0-12-801460-8. doi: http://dx.doi.org/10.1016/B978-0-12-801460-8.00002-1. URL http://www.sciencedirect.com/science/article/pii/B9780128014608000021.

Philipp Kranen, Hardy Kremer, Timm Jansen, Thomas Seidl, Albert Bifet, Geoff Holmes, Bernhard Pfahringer, and Jesse Read. Stream data mining using the moa framework. In Sang-goo Lee, Zhiyong Peng, Xiaofang Zhou, Yang-Sae Moon, Rainer Unland, and Jaesoo Yoo, editors, *Database Systems for Advanced Applications*, volume 7239 of *Lecture Notes in Computer Science*, pages 309–313. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-29034-3. doi: 10.1007/978-3-642-29035-0_27. URL http://dx.doi.org/10.1007/978-3-642-29035-0_27.

Georg Krempl. The algorithm apt to classify in concurrence of latency and drift. In João Gama, Elizabeth Bradley, and Jaakko Hollmén, editors, *Advances in Intelligent Data Analysis X*, volume 7014 of *Lecture Notes in Computer Science*, pages 222–233. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24799-6. doi: 10.1007/978-3-642-24800-9_22. URL http://dx.doi.org/10.1007/978-3-642-24800-9_22.

Georg Krempl, Indre Žliobaite, Dariusz Brzeziński, Eyke Hüllermeier, Mark Last, Vincent Lemaire, Tino Noack, Ammar Shaker, Sonja Sievi, Myra Spiliopoulou, and Jerzy Stefanowski. Open challenges for data stream mining research. *SIGKDD Explor. Newsl.*, 16(1):1–10, September 2014. ISSN 1931-0145. doi: 10.1145/2674026.2674028. URL http://doi.acm.org/10.1145/2674026.2674028.

Martin Krzywinski and Naomi Altman. Points of significance: Visualizing samples with box plots. *Nat Meth*, 11(2):119–120, 02 2014. URL http://dx.doi.org/10.1038/nmeth.2813.

Matjaž Kukar. Drifting concepts as hidden factors in clinical studies. In Michel Dojat, ElpidaT. Keravnou, and Pedro Barahona, editors, *Artificial Intelligence in Medicine*, volume 2780 of *Lecture Notes in Computer Science*, pages 355–364. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20129-8. doi: 10.1007/978-3-540-39907-0_49. URL http://dx.doi.org/10.1007/978-3-540-39907-0_49.

Ludmila I. Kuncheva. Classifier ensembles for changing environments. In Fabio Roli, Josef Kittler, and Terry Windeatt, editors, *Multiple Classifier Systems*, volume 3077 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004. ISBN 3-540-22144-1. URL http://dblp.uni-trier.de/db/conf/mcs/mcs2004.html#Kuncheva04.

Neal Lathia, Stephen Hailes, and Licia Capra. knn cf: A temporal social network. In *Proceedings of the 2008 ACM Conference on Recommender Systems*, RecSys '08, pages 227–234, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-093-7. doi: 10.1145/1454008.1454044. URL http://doi.acm.org/10.1145/1454008.1454044.

Guy Lebanon and Yang Zhao. Local likelihood modeling of temporal text streams. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 552–559, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390226. URL http://doi.acm.org/10.1145/1390156.1390226.

Lin Liao, Donald J. Patterson, Dieter Fox, and Henry Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5–6):311 – 331, 2007. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/j.artint.2007.01.006. URL http://www.sciencedirect.com/science/article/pii/S0004370207000380.

Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 36–54, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67907-3. URL http://dl.acm.org/citation.cfm?id=646765.704129.

Charles X. Ling and Victor S. Sheng. Cost-sensitive learning. In *Encyclopedia of Machine Learning*, pages 231–235. 2010. doi: 10.1007/978-0-387-30164-8_181. URL http://dx.doi.org/10.1007/978-0-387-30164-8_181.

Rongxing Lu, Hui Zhu, Ximeng Liu, J.K. Liu, and Jun Shao. Toward efficient and privacy-preserving computing in big data era. *Network, IEEE*, 28(4):46–50, July 2014. ISSN 0890-8044. doi: 10.1109/MNET.2014.6863131.

Oded Maimon and Lior Rokach, editors. *Data Mining and Knowledge Discovery Handbook, 2nd ed.* Springer, 2010. ISBN 978-0-387-09822-7. URL http://dblp.uni-trier.de/db/reference/dmkdh/dmkdh2010.html.

M.B. Malik, M.A. Ghazi, and R. Ali. Privacy preserving data mining techniques: Current scenario and future prospects. In *Computer and Communication Technology (ICCCT), 2012 Third International Conference on*, pages 26–32, Nov 2012. doi: 10.1109/ICCCT.2012.15.

D. L. Massart, A. J. Smeyers-verbeke, Capron, and Karin Schlesier. Visual presentation of data by means of box plots, 2005.

Brian Babcock Mayur, Brian Babcock, Mayur Datar, and Rajeev Motwani. Load shedding techniques for data stream systems. In *Proc. of the 2003 Workshop on Management and Processing of Data Streams (MPDS*, 2003.

MOA. Moa api. http://www.cs.waikato.ac.nz/~abifet/MOA/API/index.html. (Visited on 01/22/2015).

MOA. Overview - moa (massive on-line analysis), 2013. URL http://moa.cms.waikato.ac.nz/overview.

S. Muthukrishnan. Data streams: Algorithms and applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 413–413, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics. ISBN 0-89871-538-5. URL http://dl.acm.org/citation.cfm?id=644108.644174.

Fulufhelo Vincent Nelwamondo and Tshilidzi Marwala. Key issues on computational intelligence techniques for missing data imputation - a review. In *Proceedings of World Multi Conference on Systemics, Cybernetics and Informatics*, volume 4, pages 35–40, 2008.

E. Noack, A. Luedtke, I. Schmitt, T. Noack, E. Schaumlöffel, E. Hauke, J. Stamminger, and E. Frisk. The columbus module as a technology demonstrator for innovative failure management. In *Proceedings of the Deutscher Luft– und Raumfahrtkongress des DGLR*, 12 2012.

Regina Nuzzo. Scientific method: Statistical errors, 2014. [Online; posted 12 February 2014].

David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

Nikunj C. Oza and Stuart Russell. Experimental comparisons of online and batch versions of bagging and boosting. In *ACM SIGKDD*, pages 359–364.

Nikunj C. Oza and Stuart Russell. Online bagging and boosting. In *Artificial Intelligence and Statistics 2001*, pages 105–112, January 2001.

Norman Poh, Rita Wong, Josef Kittler, and Fabio Roli. Challenges and research directions for adaptive biometric recognition systems. In Massimo Tistarelli and MarkS. Nixon, editors, *Advances in Biometrics*, volume 5558 of *Lecture Notes in Computer Science*, pages 753–764. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-01792-6. doi: 10.1007/978-3-642-01793-3_77. URL http://dx.doi.org/10.1007/978-3-642-01793-3_77.

J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. ISSN 0885-6125. doi: 10.1007/BF00116251. URL http://dx.doi.org/10.1007/BF00116251.

Peter Reutemann and Joaquin Vanschoren. Scientific workflow management with adams. In PeterA. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 7524 of *Lecture Notes in Computer Science*, pages 833–837. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33485-6. doi: 10.1007/978-3-642-33486-3_58. URL http://dx.doi.org/10.1007/978-3-642-33486-3_58.

André Luis Debiaso Rossi, André Carlos Ponce De Leon Ferreira De Carvalho, Carlos Soares, and Bruno Feres De Souza. Metastream: A meta-learning based method for periodic algorithm selection in time-changing data. *Neurocomput.*, 127:52–64, March 2014. ISSN 0925-2312. doi: 10.1016/j.neucom.2013.05.048. URL http://dx.doi.org/10.1016/j.neucom.2013.05.048.

Stuart J. Russell, Peter Norvig, John F. Candy, Jitendra M. Malik, and Douglas D. Edwards. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 3rd edition, 1996. ISBN 0-13-103805-2.

T. Sakaki, M. Okazaki, and Y. Matsuo. Tweet analysis for real-time event detection and earthquake reporting system development. *Knowledge and Data Engineering, IEEE Transactions on*, 25(4):919–931, April 2013. ISSN 1041-4347. doi: 10.1109/TKDE.2012.29.

Joel Scanlan, Jacky Hartnett, and Raymond Williams. Dynamicweb: Adapting to concept drift and object drift in cobweb. In *Proceedings of the 21st Australasian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, AI '08, pages 454–460, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-89377-6. doi: 10.1007/978-3-540-89378-3_46. URL http://dx.doi.org/10.1007/978-3-540-89378-3_46.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, July 1990. ISSN 0885-6125. doi: 10.1023/A:1022648800760. URL http://dx.doi.org/10.1023/A:1022648800760.

Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

Myra Spiliopoulou and Georg Krempl. Mining multiple threads of streaming data. *Tutorial at the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2013)*, April 2013. URL https://kmd.cs.ovgu.de/tutorial_pakdd2013.html.

StatSoft. How to group objects into similar categories, cluster analysis. http://www.statsoft.com/textbook/cluster-analysis, 2014.

Nesime Tatbul, Uğur Çetintemel, Stan Zdonik, Mitch Cherniack, and Michael Stonebraker. Load shedding in a data stream manager. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 309–320. VLDB Endowment, 2003. ISBN 0-12-722442-4. URL http://dl.acm.org/citation.cfm?id=1315451.1315479.

Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, 23(9):661–692, September 2006. ISSN 0741-2223. doi: 10.1002/rob.v23:9. URL http://dx.doi.org/10.1002/rob.v23:9.

Hannu Toivonen. Sampling large databases for association rules. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 134–145, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN 1-55860-382-4. URL http://dl.acm.org/citation.cfm?id=645922.673325.

UBDM. Ubdm 2005: Proceedings of the first international workshop on utility-based data mining. New York, NY, USA, 2005. ACM. ISBN 1-59593-208-9.

UBDM. Ubdm 2006: Second international workshop on utility-based data mining. New York, NY, USA, 2006. ACM. ISBN 1-59593-440-5.

ML Repository UCI. Uci machine learning repository: Covertype data set. https://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.info, 1998. (Visited on 01/20/2015).

ML Repository UCI. Uci machine learning repository: Poker hand data set. https://archive.ics.uci.edu/ml/datasets/Poker+Hand, 01 2007. (Visited on 01/20/2015).

Indrė Žliobaitė. Learning under concept drift: an overview. *Computing Research Repository*, abs/1010.4784, 2010. URL http://arxiv.org/abs/1010.4784.

Indre Žliobaite. How good is the electricity benchmark for evaluating concept drift adaptation. *CoRR*, abs/1301.3524, 2013. URL http://arxiv.org/abs/1301.3524.

Indre Žliobaite, Albert Bifet, Bernhard Pfahringer, and Geoffrey Holmes. Active learning with drifting streaming data. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(1):27–39, Jan 2014. ISSN 2162-237X. doi: 10.1109/TNNLS.2012.2236570.

Indre Žliobaite, Marcin Budka, and Frederic Stahl. Towards cost-sensitive adaptation: When is it worth updating your predictive model? *Neurocomputing*, 150, Part A(0):240 – 249, 2015. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2014.05.084. URL http://www.sciencedirect.com/science/article/pii/S0925231214012971.

Boyu Wang and Joelle Pineau. Online ensemble learning for imbalanced data streams. *CoRR*, abs/1310.8004, 2013. URL http://arxiv.org/abs/1310.8004.

Ting Wang and Ling Liu. Butterfly: Protecting output privacy in stream mining. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1170–1179, April 2008. doi: 10.1109/ICDE.2008.4497526.

David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Comput.*, 8(7):1341–1390, oct 1996. ISSN 0899-7667. doi: 10.1162/neco.1996.8.7.1341. URL http://dx.doi.org/10.1162/neco.1996.8.7.1341.

Bianca Zadrozny, Gary Weiss, and Maytal Saar-Tsechansky. Ubdm 2006: Utility-based data mining 2006 workshop report. *SIGKDD Explor. Newsl.*, 8(2):98–101, December 2006. ISSN 1931-0145. doi: 10.1145/1233321.1233338. URL http://doi.acm.org/10.1145/1233321.1233338.

Yu Zhang, Daby Sow, Deepak Turaga, and Mihaela van der Schaar. A fast online learning algorithm for distributed mining of bigdata. *SIGMETRICS Perform. Eval. Rev.*, 41(4):90–93, April 2014. ISSN 0163-5999. doi: 10.1145/2627534.2627562. URL http://doi.acm.org/10.1145/2627534.2627562.

APPENDIX

A

Implementation in the MOA Source Code

A.1 Modification to Measurement Class

This first implementation in the source code is to the measurement class in `Measurement.java`. In the original implementation, the averaged measurements where provided in an `ArrayList` of class `Measurement` called `averageMeasurements`. Because our implementation requires calling the current value of the *"[avg]model serialized size (bytes)"*, searching through this List for every iteration of the method will be quite inefficient. We created a `LinkedHashMap` variable called `averagedMap` to put this values. This allow us to efficiently call any measurement parameters without any significant performance hit. We however return this values to the `averageMeasurements` variable by the end of the method because other parts of the software (not requiring a search) uses the variable as an `ArrayList`.

Source Code 1: averageMeasurements method

```
1 public static Measurement[]
    averageMeasurements(Measurement[][] toAverage) {
2     List<String> measurementNames = new ArrayList<String>();
3     for (Measurement[] measurements : toAverage) {
4         for (Measurement measurement : measurements) {
5             if
                  (measurementNames.indexOf(measurement.getName())
                  < 0) {
```

```
6                   measurementNames.add(measurement.getName());
7               }
8           }
9       }
10      GaussianEstimator[] estimators = new
           GaussianEstimator[measurementNames.size()];
11      for (int i = 0; i < estimators.length; i++) {
12          estimators[i] = new GaussianEstimator();
13      }
14      for (Measurement[] measurements : toAverage) {
15          for (Measurement measurement : measurements) {
16              estimators[measurementNames.indexOf(measurement.getName())
                   ].addObservation(measurement.getValue(), 1.0);
17          }
18      }
19      Map<String, Double> averagedMap = new
           LinkedHashMap<String, Double>();
20      for (int i = 0; i < measurementNames.size(); i++) {
21          String mName = measurementNames.get(i);
22          GaussianEstimator mEstimator = estimators[i];
23          if (mEstimator.getTotalWeightObserved() > 1.0) {
24              averagedMap.put("[avg]"+mName,
                   mEstimator.getMean());
25              averagedMap.put("[err]"+mName,
                   mEstimator.getStdDev()
26                        /
                              Math.sqrt(mEstimator.getTotalWeightObserved()));
27          }
28      }
29      measurementMap = averagedMap;
30
31      List<Measurement> averagedMeasurements = new
           ArrayList<Measurement>();
32
33      for (Entry<String, Double> entry :
           averagedMap.entrySet())
34          averagedMeasurements.add(new
               Measurement(entry.getKey(), entry.getValue()));
35
36      return averagedMeasurements.toArray(new
           Measurement[averagedMeasurements.size()]);
37  }
```

A.2 Implementaton of Algorithm

The modifications below are made to methods in the `OzaBag` class for our implementation. Similar changes where also made for the `OzaBagAdwin` variant. The `ensemble` variable in the original implementation in the source code is of type `Array`. We changed it to an

`ArrayList` in our implementation to enable us manipulate the length as required. This affects some methods in the class, and they were adjusted as required, as shown in codes 2, 3, 4, 5 and 6.

Source Code 2: resetLearningImpl method

```
public void resetLearningImpl() {
    this.ensemble = new ArrayList<Classifier>();
    Classifier baseLearner = (Classifier)
       getPreparedClassOption(this.baseLearnerOption);
    baseLearner.resetLearning();

    for (int i = 0; i < ensembleSize; i++) {
            this.ensemble.add(baseLearner.copy());
    }
    for (int i = 0; i < 3; i++)
        serializedMem.add(null);
}
```

Source Code 3: trainOnInstanceImpl method

```
public void trainOnInstanceImpl(Instance inst) {
    if (memIncrease0 != null && memIncrease1 != null){
      if (memIncrease1 > memIncrease0 && ensembleSize < 25)
         ensembleSize += 1;
      if (memIncrease1 < memIncrease0 && ensembleSize > 10)
         ensembleSize -= 1;
    }
    if (ensembleSize > ensemble.size()){
      Classifier baseLearner = (Classifier)
         getPreparedClassOption(this.baseLearnerOption);
      baseLearner.resetLearning();
      for (int j = ensemble.size(); j < ensembleSize; j++)
      this.ensemble.add(baseLearner.copy());
    }
    for (int i = 0; i < ensembleSize; i++) {
        int k = MiscUtils.poisson(1.0,
           this.classifierRandom);
        if (k > 0) {
            Instance weightedInst = (Instance) inst.copy();
            weightedInst.setWeight(inst.weight() * k);
            this.ensemble.get(i).trainOnInstance(weightedInst);
        }
    }
    if (Measurement.measurementMap != null)
      serializedMem0 =
         Measurement.measurementMap.get("[avg]model
         serialized size (bytes)");

    serializedMem.add(serializedMem0);
    if (serializedMem.size() > 3)
```

```
27        serializedMem.remove();
28
29      if (serializedMem.get(0) != null && serializedMem.get(1)
           != null){
30        if (serializedMem.get(0).doubleValue() !=
             serializedMem.get(1).doubleValue())
31          memIncrease0 = Math.abs(serializedMem.get(0) -
               serializedMem.get(1));
32      }
33      if (serializedMem.get(1) != null && serializedMem.get(2)
           != null && memIncrease0 != null){
34        if (serializedMem.get(1).doubleValue() !=
             serializedMem.get(2).doubleValue()){
35          Double t = Math.abs(serializedMem.get(1) -
               serializedMem.get(2));
36          if (memIncrease0.doubleValue() != t.doubleValue())
37          memIncrease1 = t;
38        }
39      }
40  }
```

Source Code 4: getVotesForInstance method

```
1  public double[] getVotesForInstance(Instance inst) {
2      DoubleVector combinedVote = new DoubleVector();
3      for (int i = 0; i < ensembleSize; i++) {
4          DoubleVector vote = new
               DoubleVector(this.ensemble.get(i).getVotesForInstance(inst));
5          if (vote.sumOfValues() > 0.0) {
6              vote.normalize();
7              combinedVote.addValues(vote);
8          }
9      }
10     return combinedVote.getArrayRef();
11 }
```

Source Code 5: getModelMeasurementsImpl method

```
1  protected Measurement[] getModelMeasurementsImpl() {
2      return new Measurement[]{new Measurement("ensemble size",
3              this.ensemble.isEmpty() ? 0 : ensembleSize)};
4  }
```

Source Code 6: getSubClassifiers method

```
1  public Classifier[] getSubClassifiers() {
2      return this.ensemble.toArray(new
           Classifier[ensemble.size()]).clone();
3  }
```