uOttawa

Faculty of Graduate and Postdoctoral Studies

A Comparative Study of Ensemble Active Learning

by

Rabaa Alabdulrahman

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies in partial fulfillment of the requirement for the degree of

Master of Science in System Science

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

Abstract

Data Stream mining is an important emerging topic in the data mining and machine learning domain. In a Data Stream setting, the data arrive continuously and often at a fast pace. Examples include credit cards transaction records, surveillances video streams, network event logs, and telecommunication records. Such types of data bring new challenges to the data mining research community. Specifically, a number of researchers have developed techniques in order to build accurate classification models against such Data Streams. Ensemble Learning, where a number of so-called base classifiers are combined in order to build a model, has shown some promise. However, a number of challenges remain. Often, the class labels of the arriving data are incorrect or missing. Furthermore, Data Stream algorithms may benefit from an online learning paradigm, where a small amount of newly arriving data is used to learn incrementally. To this end, the use of Active Learning, where the user is in the loop, has been proposed as a way to extend Ensemble Learning. Here, the hypothesis is that Active Learning would increase the performance, in terms of accuracy, ensemble size, and the time it takes to build the model.

This thesis tests the validity of this hypothesis. Namely, we explore whether augmenting Ensemble Learning with an Active Learning component benefits the Data Stream Learning process. Our analysis indicates that this hypothesis does not necessarily hold for the datasets under consideration. That is, the accuracies of Active Ensemble Learning are not statistically significantly higher than when using normal Ensemble Learning. Rather, Active Learning may even cause an increase in error rate. Further, Active Ensemble Learning actually results in an increase in the time taken to build the model. However, our results indicate that Active Ensemble Learning builds accurate models against much smaller ensemble sizes, when compared to the traditional Ensemble Learning algorithms.

Further, the models we build are constructed against small and incrementally growing training sets, which may be very beneficial in a real time Data Stream setting.

Acknowledgments

This journey has been both challenging and exciting. Although I came from Saudi Arabia as an international student, I feel this is now my second home. First and foremost, I would like to thank God for being my strength and guide throughout life. Also, I would like to thank every person who inspired me and was part of my journey.

I would like to thank my supervisor Prof. Herna Viktor and express my sincere gratitude for her continuous support and advice for my studies and research. Her patience, encouragement, and knowledge motivated me and kept me going to write this thesis. It has been an honor.

I kindly thank the academic staff of the System Science program at the University of Ottawa.

Also, I would also like to thank the Saudi Cultural Bureau in Canada for their support that helped me achieve my goals.

I would like to thank my family: my parents Saleh and Aisha, for loving and supporting me throughout my life and my six siblings whom I cannot imagine my life without, especially my younger brother, Ahmed, for being a part of my journey in Canada. Thank you family for all the support you gave me despite the distance that came as a result of me pursuing my goals.

Special thanks go to my friends whom I met in Canada Shaikha, Wadha, Jumanah and Fadi who have all become a family to me. Your words kept me going whenever I became frustrated and when coffee lost its effect on me.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

Over recent decades significant changes have influenced our traditional lives. Education, health care, the economy, and every single aspect of our world have been further developed by technology. Whether at hospitals, banks, universities, or broadcast stations, data needs to be stored to serve different kinds of needs. As a consequence, databases became massive data hoarders and the traditional way of handling this data became inefficient. Therefore, data scientists have considered the possibility of obtaining what they need from that database without the need to go through every minor detail in it. A need to understand and extract knowledge from such vast repositories became a priority. To address this need, researchers developed new data mining algorithms to study the patterns of such data to explain the information they need to know. In general, such data mining methods extract knowledge nuggets from large-scale databases, using techniques from a number of areas, including artificial intelligence, statistics, graph theory and visualization.

Data mining methods can be divided into two broad categories: namely supervised and unsupervised learning. In the supervised learning methods, also known as classification, the classes with a set of correctly classified instances are given to the algorithm. According to the given information, the algorithm should be able to classify any new instances. An example of supervised learning is Ensemble Learning, which develops a number of hypotheses from different learners (Sewell, 2008; Zhou, 2012). That is, in an Ensemble, we typically combine a number of classifiers in order to build a model. It often results in improving the performance accuracy when compared to using a single classifier. On the other hand, unsupervised algorithms (also called cluster analysis

algorithms) learn which objects will be grouped together and what classes should be built from a given number of objects (Battula & Prasad, 2013).

With the passage of time and the development of the technology, organizations have started to collect new types of data. A large number of transaction records made in banks or in supermarkets, for example, constitute a large volume of continuously arriving data. Also, the development of networks and sensors has introduced a need to store real time events. These kinds of rapidly growing data, collected and arriving continuously, are called Data Streams (Aggarwal, 2007). This type of data, which are in essence online, brings new challenges to the data science domain. The question is: Will traditional Data Mining techniques be able to build as accurate of models from Data Streams, as they would with static snapshots of data? The online element has opened a new area of research in Data Streams.

Specifically, a Data Stream differs from a static regular database because of its unique formation. In a Data Stream, massive amounts of data usually arrive in changeable continuous time and in high speed (examples include credit cards transaction records, surveillances video streams, networks event logs, and telecommunication records). This fact creates challenges in research within the database space as well as in the computing process. Traditional Data Mining tools require a number of scans of the data to capture the patterns and extract the knowledge, which is hard to do in Data Streams (Ikonomovska, Loskovska, & Gjorgjevik, 2007). Importantly, the class labels are often missing, or arrive at a later stage. This fact makes the application of traditional classification techniques, which require a class label, difficult. To address these issues, the idea of Active Learning was introduced. Active Learning refers to algorithms with the ability to build a number of trained models from the fewest labeled examples as possible (Settles, 2010). Further, in this approach, the user plays an important part in the learning process. That is, Active Learning refers to the scenario where the user is in the loop, guiding the learning process by verifying the class labels. Therefore, the system is able to learn in real time from the examples it has so far, following an online, iterative approach. Also, Active Learning systems are be able to recognize and handle repeated data in order to build accurate models against Data Streams (Tsymbal, 2004).

1.1 Motivation

Today's complex market and human lifestyle integrated with technology created a need for more simplification. Users, such as managers and team leaders, need some tools to help them understand the contents and analyze the databases they are working with. According to the Digital Universe study, the world's information will have reached the size of 1.8 zettabytes, and by 2020 this number will double by 50 times (Corporation, 2014; Krempl et al., 2014). Online learning is a new subarea of data mining research area that aims at performing the classification task in real time and being able to fit in a limited memory (Zliobaite, Bifet, Pfahringer, & Holmes, 2014). In other words, these algorithms will update the model whenever new examples arrive in the stream.

In many ways, people tend to learn from Data by taking instructions from authorities, by self-learning, or by asking questions. Therefore, it is normal to extend machine learning tasks to include human expertise. To this end, Active Learning methods have been suggested as a solution for Data Stream classification tasks (X. Zhu, Zhang, Lin, & Shi, 2007; Zliobaite et al., 2014). For this purpose, in this thesis we study the impact of using extending traditional Ensemble Learning algorithms through Active Learning. Our aim is to evaluate the performance of both learning methods (that is, Ensemble Learning and Ensemble Learning combined with Active Learning) and learn if it is really beneficial to use the Active Learning methods. In our suggested framework we thus proceed by extending Ensemble Learning techniques with online Active Learning. The general idea about expanding Ensemble Learning through Active Learning is presented in Figure 1.1. Note that, in the Active Learning setting, we make use of online learning, where training examples are added in an iterative manner (N. Oza, 2001; N. C. Oza & Russell, 2005).

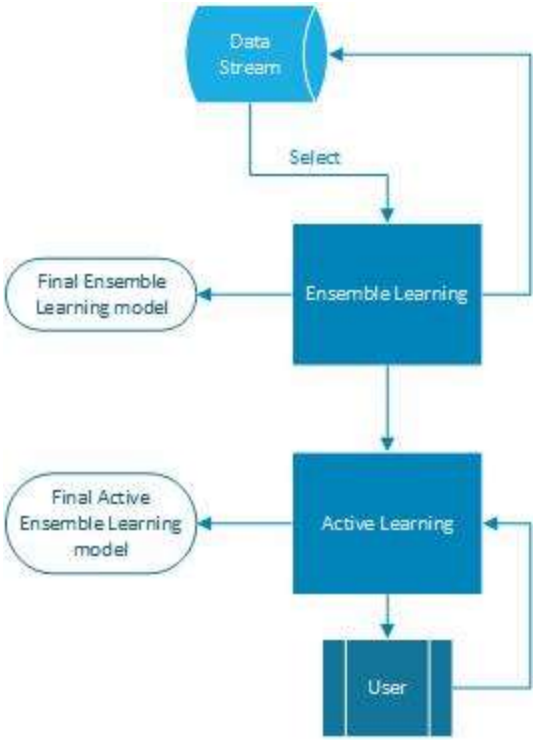

Figure 1: The Active Ensemble Learning Process- General Idea

The suggested framework is mainly divided into two main stages. The first one will carry on the Ensemble Learning process. In the second stage Active Learning will be used. In this study, four learning methods are chosen. Two state-of-the art Ensemble Learning methods, namely Bagging and Boosting, are used to perform the Ensemble Learning stage, whereas Query by Bagging and Query by Boosting are used in the Active Learning stage. Throughout this study, these four algorithms are separately tested using four different base classifiers, namely the C4.5 decision tree algorithm, the Naïve Bayes classifier, the k-Nearest Neighbor algorithm, and the Hoeffding trees. (These algorithms, or so-called base classifiers, were chosen due to the accurate models built during Ensemble Learning (Han, Kamber, & Pei, 2006; Witten & Frank, 2005).) It is interesting to study their performance in the Active Learning methods and to see if they will be able to result in better models compared to the Ensemble ones. More details about these classifiers will be provided in the literature review in Chapter 2.

Section 1.1 introduced the definition of Ensemble Learning and Active Learning. In this thesis, the definition of online learning presented by Oza and Russell is adapted (N. Oza,

2001; N. C. Oza & Russell, 2005). The online learning should be able to learn information from small amounts of training sets and be able to use this information to classify new instances as they arrive in the stream. Also, crucially, it should learn in a real time as the instances continuously arrive in the stream.

From the information above, a number of reasons motivate us to investigate the performance of Ensemble Learning and Active Ensemble Learning. One more reason is the time needed to classify new examples. A Data Stream contains a large amount of data that needs to be handled in a way that captures the important information from the smallest training size as possible. Therefore, Ensemble Learning naturally allows the algorithm to train different classifiers at the same time which helps reduce the execution time. In this study, we evaluate the performance of different learning methods based on the resulted model accuracy and on the execution time.

In this study, our framework provides a comparative study for the performance of both Ensemble Learning and Active Learning. This helps understand the strengths and weaknesses of both learning methods when using different classifiers. Also, it provides a better idea about how beneficial the Active Learning method is.

1.2 Thesis Organization

The rest of this thesis is divided into two parts. The first part covers the literature review of this thesis, which includes two chapters. Chapter 2 introduces Data Stream mining. Also, it presents the four classifiers that are used in this thesis along with the Ensemble Learning methods. In Chapter 3, the Active and online learning concepts and techniques are discussed.

The second part focuses on extending Ensemble Learning through Active Learning. This part introduces the online Active Ensemble Learning framework that is used in this study. The pseudo-code along with running examples and discussion are presented in Chapter 4. In Chapter 5, an overall evaluation of all the resulted data is presented. This evaluation includes the model's accuracy, the ensemble sizes, and the time evaluation. Finally, in Chapter 6 we present our contribution and future work.

1.3 Objectives and Contributions

As mentioned in our motivation for this study, Active Learning has been suggested as a solution for Data Stream mining (X. Zhu et al., 2007; Zliobaite et al., 2014). A number of researchers have indicated that Active Learning may be crucial when aiming to build accurate, just-in-time models. That is, researchers believe that the "user in the loop" paradigm may benefit Data Stream algorithms (X. Zhu et al., 2007). Our aim is to determine how valid this hypothesis, which has not been experimentally tested, actually is. Therefore, the main objective of this thesis is to find out how beneficial Active Learning actually proves to be. The proposed framework in this study thus aims to answer three main questions. First of all, we evaluate the strengths and weaknesses of both Ensemble Learning and Active Learning. Secondly, we provide guidelines in order to indicate when to use the Active Ensemble Learning framework. Thirdly, our goal is to provide researchers with the ability to understand when the Active Ensemble Learning would be beneficial and when to avoid using it. Our other contributions include the study of the suitability of the base classifiers used in our study, as well as the impact of the training set sizes on a number of classifiers in an ensemble.

Our results may be summarized as follows. In our study we found that extending Ensemble through Active Learning methods did not improve the resulted accuracy against the datasets used in this study, especially in a setting where there is no or little class imbalance. Further, it added the most cost to the running time. We found that the error rate remains the same or increases when the Active Learning methods are applied. However, we found that the Ensemble Learning methods result in more accurate models when the ensemble size of 25 is used. On the other hand, the Active Learning methods created in the most accurate models when the ensemble size of 10 is used. This implies that Active Learning ensembles are typically much smaller and provides us with a more cost efficient learning environment. Another strength of our paradigm that stands in contrast to traditional classification is the fact that while the initial training set size is small and incremental, it still leads to accurate models. Thus, our experimental evaluation suggests that Active Learning would scale well to a cost-sensitive learning setting where the size of the Ensemble is of importance. The above mentioned provides guidelines as to

scenarios where it may be beneficial to employ the Active Ensemble Learning framework.

In summary, our study indicates that Active Learning does not necessarily give the benefits that we expected. While it produces accurate models against smaller ensemble sizes, the accuracy (in the experiments we considered) is not superior to the traditional Ensemble Learning setting. Furthermore, the time taken to build the models are not promising, especially if one wishes to extend this work to massive Data Streams. However, our analysis suggests that this framework might present solutions for very specific classification situations. For instance, in case of having a large number of mislabeled or noisy data, Active Learning may be beneficial to build an accurate model, due to the user-in-the-loop paradigm (X. Zhu et al., 2007). Also, late arriving data and concept drift may both be handled by using the Active Learning methods (Zliobaite et al., 2014). Moreover, Active Learning methods may also be used to solve the multi-label problem (P. Wang, Zhang, & Guo, 2012).

Part I

BACKGROUND STUDY

Chapter 2

Mining Data Streams

Data Stream Mining is different from classical Data Mining since a Data Stream has a unique form that creates challenging research problems. This mining process is defined as the algorithmic ability to extract information from a continuous Data Stream (Mena-Torres & Aguilar-Ruiz, 2014). Therefore, the continuously arriving data at different times poses a need to come up with new techniques instead of the traditional ones. In traditional Data Mining techniques, data is stored then processed offline by algorithms, which usually requires several passes through the data (Mena-Torres & Aguilar-Ruiz, 2014). Also, traditional techniques work with the same data distribution for the stored data. However, in Data Stream, to result in successful classification algorithms, the algorithms must work in real-time, pass over data once, and be able to work with limited memory resources. Also, these algorithms must be able to detect the change in the data distribution and adapt to them. Data Streams are presented in the following Section 2.1.

Overall, the traditional Data Mining process is a cooperative and collaborative process between many domains such as statistics, machine learning, data warehouses and many other (Han et al., 2006). In this thesis two important domains are used; namely statistics and machine learning. In the statistical model a number of mathematical functions are used to associate variables with classes according to their probability distribution. An example of this domain is Ensemble Learning, which uses multiple algorithms to improve the performance of a single classifier (more details about this will be given in Section 2.3). Moreover, machine learning studies how to improve computer performance based on given data. In machine learning there are four major tasks (Han et al., 2006). In this thesis the focus will be on two tasks only. The first one is Supervised Learning, also known as Classification, in which there is a training data set that includes sets of labeled data that supervise the learning process of the classification model. The second is the

Active Learning task. In this task, the user plays an important part in the learning process. More details about this task are given in Chapter 3.

2.1 Introduction to Data Streams

As mentioned in the previous section, a Data Stream is continuously generated data that arrives in a rapid rate and may include an infinite number of instances. Examples include network traffic logs, surveillance video streams, call center data, store transactions, credit card transactions and the stock market. The underlying problem that needs to be solved is in finding methods that can handle an infinite amount of data and modeling them in a way that captures the time and patterns in the stream (H. Wang, Fan, Yu, & Han, 2003). Many algorithms that deal with Data Stream are able to read the incoming data once because of the limitation of memory, time, or sample size. The goal of these algorithms is to predict the class of the newly arrived instances, using some knowledge from the previous instances. To solve the previously mentioned problem that is present in traditional Data Mining techniques, incremental and online Data Mining methods work by endlessly revising and refining the model through considering new data as it arrives (H. Wang et al., 2003).

Recall that when dealing with Data Stream, there are several constraints; such as the large amounts of data that arrived or will arrive, making it impossible to be entirely stored. Hence, a computed summary about the already arrived data will be stored and the rest will be archived. Also, each element should be processed in real time because of the high speed of the arriving data. Finally, the instances distribution can change over time; therefore, old data may become irrelevant (Bifet, 2009). Most of the statistical and machine learning algorithms utilize the assumption of random sampling to create the training data. It is unfortunate that most Data Streams break this assumption. Instances appear in the stream over a period of time and the generating process of them might change over time (Geoffrey Hulten, Domingos, & Spencer, 2005). Data Stream processing is classified into Data Stream management, which consists of querying and summarizing the Data Stream while the second classification is Data Stream mining, which includes performing the Data Mining techniques (Gaber, Zaslavsky, & Krishnaswamy, 2010).

In Data Streams there are two types of changes that might occur. The first one is when the concept changes over time, which is known as Concept Drift. The second one is when new classes evolve in the stream, which is known as Concept Evolution (Piramuthu, 2004). The time period, which the concept changes over, is an important factor. Therefore, researchers classify Concept Drift as sudden, gradual, and incremental. In any classification, we still need to find systems to handle the drift. The first systems to handle the drift were STAGGER, FLORA, and IB3 (Tsymbal, 2004). After that, three system approaches became well known for being handling systems types. These three approaches are: Instance Selection, Instance Weighting, and Ensemble Learning (Tsymbal, 2004). However, in this thesis the Concept Drift issue is not addressed. This will be considered in our future work. Hence, as part of this experiment, the sensitivity of the base classifier to Concept Drift is explored. In particular, as discussed in Section 2.2, the C4.5 decision tree algorithm is included in this experiment. It is interesting to evaluate the performance of this algorithm in a Data Stream setting with the present of concept drift.

2.2 Classification Methods

The main goal of the classification method is to predict a specific outcome based on a given input. Using a classification algorithm, a relationship between the training set attributes is discovered to predict the outcome. Using the same algorithm, another dataset without the prediction attribute is given. The algorithms subsequently will analyze the given data to come up with a prediction. The algorithm outcome includes models that describe the data classes. Based on this outcome, class labels can be assigned to each record in the dataset (Beniwal & Arora, 2012). To decide how good the algorithm prediction is, one may consider the classification's accuracy. The accuracy is measured by the percentage of the incorrectly classified instances. A short discussion about accuracy and performance measures will be presented in Chapter 6.

Overall, the classification processes is divided into two main steps. The first step is the learning step. In this step the outcome is the classification model. The second step is a classification step, where the outcome is the predicted class label. This step is done by using the classification model to predict labels for the unlabeled dataset (Han et al.,

2006). The classification method can be conducted in different techniques such as Decision Tree, Nearest Neighbor, Bayesian Network, etc. The following section explains some of the classification techniques that are used in this thesis.

2.2.1 C4.5 Decision Trees

C4.5 is an extended algorithm of ID3, and both were developed by Ross Quinlan to generate a decision tree (K. Quinlan, 1999). C4.5 classifier was developed to overcome a limitation in ID3 in which the algorithm is sensitive to a large number of information that may exist in one attribute. The C4.5 algorithm overcomes this problem by using a statistical property that is the Information Gain. The Information Gain measures the best attribute to be used as a separating point in the training set. It works by evaluating the content information value (Han et al., 2006). Using equation 2.1, the attribute with the highest information gain is chosen as a splitting node (Jin & Agrawal, 2003). The Information Gain will reduce the problem of having a low value of entropy for some attributes. Also, Information Gain gives a measurement of the purity amount that is in each split in each subset and then the purity is measured using entropy (equation 2.2) (Bifet & Kirkby, 2009b).

$$\boldsymbol{Info(D) = \sum_{i=1}^{m} p_i \log_2(p_i)} \tag{2.1}$$

$$\boldsymbol{entropy\,(p_1, p_2, \dots, p_n) = \sum_{i=1}^{n} -p_i \log_2 p_i} \tag{2.2}$$

Alongside using C4.5 for classification problems, it can deal with both discrete and continuous attributes. To run, it requires a set of training examples. After analyzing the training sets, the algorithm builds a classifier that is able to correctly classify training and test examples. Each example in the input set should belong to one class. The output is the decision tree and/or set of rules that helps assign each example to a specific class. In general, C4.5 works by using a method known as divide and conquer to create a tree from the training sets. The divide and conquer method will run multiple scans to partition the data until each leaf has an example of the same class or until it reaches the point of

having two examples with the same value but belong to different classes (H. Wang et al., 2003). Therefore, if the algorithm has no conflicting examples, it will correctly classify all the training examples, which is known as overfitting (K. Quinlan, 1999). Overfitting generally causes a decrease in the predictive accuracy of the classifier. The pseudo-code of C4.5 is given in algorithm 1 (Ruggieri, 2002).

Smaller trees are easier to understand; therefore, most learning systems attempt to build as small trees as possible. C4.5 uses Occam's Razor to build the trees from roots to leaves (K. Quinlan, 1999). This states that by having two correct solutions as an input, the simpler one should be chosen. Moreover, there are two splitting criteria used in C4.5; namely Information Gain and Gain Ratio. The Gain Ratio measure is an extension to Information Gain measure. This measure attempts to overcome bias to attributes with more information value in Information Gain. Gain Ratio takes into account the number and size of each branch when choosing an attribute. Also, it corrects the information gain by considering the intrinsic information of the split. The intrinsic information is the entropy of distribution on instances in each branch (Ibrahim, Badr, & Shaheen, 2012). To simplify, Gain Ratio provides us with the information we need to decide what branch an instance belongs to.

This algorithm presents many advantages such as its ability to handle continuous, discrete, and missing attributes. One way to handle the missing attributes is by marking them with "?," then these attributes will not be used in the calculation of entropy and gain (J. Ross Quinlan, 1996). Moreover, it allows updating the final tree by removing some branches and adding some leaf nodes.

One of the classifier limitations is empty branches. These branches will have the value of zero or close to it. In this case, it will not affect the classification process nor contribute to generating rules. Thus, it will increase the tree size and make it more complex. Another limitation is the problem of overfitting. There are two causes of overfitting in the C4.5 algorithm. One is the insignificant branches in the tree. The algorithm will create a number of potential branches to build the final decision tree. In this case, some of these branches will be highly relevant and valuable and the others will add no value or will be left empty. Second, noisy data can mislead the process of growing branches and

classifying data. As mentioned before, this leads to having branches with no value to the process (Mazid, Ali, & Tickle, 2010).

The C4.5 algorithm requires the data to be entirely stored in order to calculate the split point (Bifet & Kirkby, 2009a). However, in Data Stream setup, the data might be infinite and will arrive at different times, which makes it unfeasible to store all the data. This potentially makes it hard for a C4.5 algorithm to perform well, since it needs to scan the entire dataset before building the model (Geoffrey Hulten et al., 2005). Thus, in the Data Stream setting, this algorithm may be used only if the stream is divided into windows, or chunks. The resultant model will be built from such a window. Hence, the model will be heavily influenced by the window size. Therefore, this approach would work well if the Data Stream does not contain Concept Drift (H. Wang et al., 2003). In this work, we include C4.5 algorithm, since we are interested in finding out how stable its performance and to detect any potential changes in the data.

Algorithm 1: Pseudo-code of C4.5 algorithm

Compute class frequency (T);
If $OneClass$ **or** $FewCases$ **return** leaf;
    Create a decision node N;
For Each attribute A ComputeGain (A);
$N.test = AttributeWithBestGain$;
If $N.test$ is continuous find Threshold;
For each $\grave{T}$ in the splitting of T
If T is Empty
    Child of N is a leaf
Else
    Child of N = formTree ($\grave{T}$);
$ComputeError$ of N;
Return N

Algorithm 1: Pseudo-code of C4.5 Algorithm

Another drawback for using C4.5 in the Data Stream setting is the need to perform multiple scans of the data to extract the information (Agrawal & Gupta, 2013). However, this is not practical in the Data Stream setting because of the continuous arriving of data and the change in data characteristics as they arrive in different times (Rutkowski,

Pietruczuk, Duda, & Jaworski, 2013). To address this issue, many methods have been proposed to handle the classification in Data Stream. One method is an incremental algorithm of Very Fast Decision Tree (VFDT), it solves the problem by using the mean of inequality of *Hoeffding Bounds* to decide when to create a new level in the current tree and produce one model to represent the entire Data Stream (Domingos & Hulten, 2000; Law & Zaniolo, 2005; H. Wang et al., 2003). Moreover, C4.5 is classified as a batch learner. In batch learning, the method is trained on batches of the data. New instances will be added to a batch until it is completed then it is passed to the learner to learn it. However, in an instance incremental learning, such as Hoeffding Tree, the method is trained on each instance as they arrive in the stream (Read, Bifet, Pfahringer, & Holmes, 2012). A study shows that using the Hoeffding Tree results in almost the same result as using a non-incremental learner (Geoff Hulten, Spencer, & Domingos, 2001).The Hoeffding Tree algorithm is introduced in Section 2.2.4.

2.2.2 Naïve Bayes Classifier

This classifier is based on the Bayes theorem with a naïve assumption: it is considered to be a probabilistic classifier (Han et al., 2006). The main objective of this classifier is to classify new instances as they arrive based on the existing objects. Therefore, by nature, the Bayesian method learns incrementally, which makes this classifier suitable for mining a Data Stream. Hence, Naïve Bayes is also known as an instance-incremental learner, where a probabilistic operation is performed on each new instance as they arrive in the stream (Read et al., 2012). To build a new model, this classifier uses the Bayes theorem or the *"likelihood"* method. Using Bayesian probability or *likelihood* depends on the problem the model tries to solve. In the case of supervised learning, "*likelihood*" will be used to create the models.

In order to understand how this algorithm works, one must understand how the Bayes theorem works. The Bayes theorem aims to classify instances, using the prior probability. Prior probability is based on the previous experience, which is based on the percentage of the current classes. In this theorem, X is considered to be a data tuple and H to be a hypothesis. Therefore, this theorem will calculate the probability that tuple X belongs to class C given some information about the attribute description. Equation 2.3 shows how

the Bayes theorem is calculated, where $P(H|X)$ is the posterior probability of H conditioned on X.

$$\boldsymbol{P(H|X) = \frac{P(X|H)P(H)}{P(X)}} \quad \textbf{(2.3)}$$

This will calculate the probability that X belongs to class C given a specific attributes description. $P(H)$ is the prior probability. Given no attribute description, what is the probability that X belongs to a class? (Han et al., 2006). The Naïve Bayes pseudo-code is given in algorithm 2 (Lowd & Domingos, 2005).

Algorithm 2: Pseudo-code of Naïve Bayes Algorithm

Input: Training set T, hold-out set H, initial number of components k_o, and convergence thresholds δ_{EM} and δ_{Add}.
Initialize M with one component.
$k \leftarrow k_o$
repeat
  Add k new mixture components to M, initialized using k random examples from T.
  Remove the k initialization examples from T.
  repeat
    $E-step$: Freactionally assign examples in $T$to mixture components; using M
    $M-step$: Compute maximum likelihood parameters for M, using the filled-in data.
    If log $P(H|M)$ is best so far, save Min M_{best}.
    Every 5 cycles, prune low-weight components of M.
  until log $P(H|M)$ fails to improve by ratin δ_{EM}.
  $M \leftarrow M_{best}$
  Prune low weight components of M.
  $k \leftarrow 2k$
until log $P(H|M)$ fails to improve by ratio δ_{Add}.
Execute E-step and M-step twice more on M_{best}, using examples from both H and T.
Return M_{best}

Algorithm 2: Pseudo-code of Naive Bayes Algorithm

Based on the given description of the Bayes theorem, the Naïve Bayes classifier works by treating each feature as an independent factor (Han et al., 2006; Mannila, 1996; Witten & Frank, 2005). It can predict and classify records depending on the present feature

regardless of the absence of some of the other features. Moreover, this classifier works under the assumption that the feature value of unlabeled instance, as in a test dataset (or of a newly arrived instance, as in a Data Stream) is not related to the existing feature. Therefore, it treats each feature independently. In fact, this assumption works for and against this algorithm. On the bright side, it gives the advantage of needing a small amount of training data to estimate the classification result, which gives it the ability to handle Data Streams (Read et al., 2012). On the downside, depending on the dataset, this assumption can sometimes be unrealistic in practice (Bifet & Kirkby, 2009a; Shi, Wang, Webb, & Huang, 2003). It can sometimes mislead the right classification by not taking into account important features (Bouckaert, 2006). Also, the algorithm may not be able to find the correlation between the features (Munro & Chawla, 2004).

Adding to the previous advantage of feature independency, it helps reduce the impact of having a multi-dimensional feature distribution since it estimates the distribution of each feature independently. In addition, it is computationally efficient, because it consumes less from both CPU and memory; therefore, it works well with limited resources of CPU and memory. The performance of the Naïve Bayes learner is very close to other complicated techniques (Witten & Frank, 2005). Besides, the symbolic nature of this classifier makes it easy to understand and implement compared to other basic classifiers. Thus, the simplicity of this classifier makes it a fast learner. Consequently, it has been utilized to be suitable for fast streams such as web proxy logs (Ali, Shamsuddin, & Ismail, 2012).

2.2.3 k-Nearest Neighbor Classifier

The k-Nearest Neighbors algorithm (kNN) is a method for classification and regression. It predicts the class value by using the k closest training examples. It is also a type of instance-based learning or lazy learning. Lazy learning works by putting off the processing of the examples until a request for information is made (X. Zhu & Yang, 2008). In a Data Stream, this request comes in the form of a test instance that needs to be classified. When the request is made, the search begins according to some measure of distance. Lazy learning requires less time in training but more in predicting. Moreover, it

constructs hypotheses directly from the training instances themselves and this is the reason it is called instance-based learning (Cunningham & Delany, 2007).

kNN is a very simple classifier that works by finding the k training samples to determine the k-nearest neighbor using a distance measure. Given N training vectors, kNN algorithm identifies the k nearest neighbor of newly arrived instance regardless of their classes. For example, if k is set equal to 3 and the classes of the current neighbor are 'b' and 'c,' the algorithm will choose the nearest three instances. Assuming there are two of class 'b' and one of class 'c', in this case, the new instance will be assigned to class 'b.' There are important elements to this classifier: a set of labeled instances, the value of k, and a distance matric to calculate the distance between the instances (Wu et al., 2008). The pseudo-code of kNN is given in algorithm 3. There is no specific method to decide what value of k is the best. Different research suggests that the value can be decided by experiments, starting with k = 1 and increasing it in each experiment. The value with the lower error rate can be chosen as the best value (Han et al., 2006). Choosing the right value of k is very crucial. If k is very small, the result can be sensitive to noise, whereas a large k means too many instances from other classes. For this reason, in this thesis the value of k is determined using cross-validation which will be explained in Section 5.2.2.

For numeric attributes, usually the neighbor's closeness may be decided based on the Euclidean distance (Amores, Sebe, & Radeva, 2006). However, the distance measure may vary depending on the type of classification and data. The Euclidean distance between two points is measured by the following equation:

$$\boldsymbol{dist}(\boldsymbol{X_1}, \boldsymbol{X_2}) = \sqrt{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{X_{1i}} - \boldsymbol{X_{2i}})^2} \quad \textbf{(2. 4)}$$

where $X_1 = (X_{11}, X_{12}, \dots\dots, X_{1n})$ and $X_2 = (X_{21}, X_{22}, \dots\dots, X_{2n})$. Usually, a normalization of the data is done before the distance calculation. This helps prevent attributes with a large range to overweigh the ones with smaller range. However, in the case of categorical attributes, the algorithm simply compares the value of the attributes. If the two attributes are identical, the difference between them will be zero. If they are different, the distance

will be set to one. Choosing the right distance method depends on the performed task. For example, in case of document classification, the cosine measure outperforms the Euclidean distance (Wu et al., 2008). Also, the Euclidean measure is sensitive to a large number of attributes.

It follows that kNN is easy to implement and to understand. Surprisingly, in many cases, it can result in highly accurate classifications. Even though the classifying process tends to be slow with a large dataset and Data Stream, the training process is very fast (Bhatia, 2010). As mentioned before, this classifier is a lazy learner. In other words, the kNN algorithm does not build the classifier in advance. This feature makes it suitable for Data Streams (Cunningham & Delany, 2007; Khan, Ding, & Perrizo, 2002). As examples arrive in the stream, the kNN classifier will find the k-nearest neighbors based on the distance metric. The new instance will be assigned to the popular class among these neighbors (Khan et al., 2002). Also, it has a highly adaptive behavior to new unlabeled data since it works by using local information. In Data Streams, examples arrive continuously at different times. In this case, observing a new example requires only an update to the distance database.

Algorithm 3: k- Nearest Neighbors Algorithm

For $i = 1$ to number of data object **do**
    Find the distances of the i^{th} object to all other objects.
    Sort these distances in decreasing order.
    (Keep track of which object is associated with each distance).
  return the objects associated with the first K distances of the sorted list
end for

Algorithm 3: Pseudo-code of kNN Algorithm

There are some downsides to this classifier. It is very sensitive to outlier instances, meaning the performance can decrease if the data is not preprocessed to remove outliers (Wu et al., 2008). Although building kNN models is easy, classifying new instances is relatively expensive and can be a slow process. This classifier requires computing the distance between new instances to all of the labeled set, which can be time and memory consuming for large datasets and Data Streams. Moreover, as mentioned before, the kNN

algorithm is a lazy learner. This happens because there is no preprocessing step until a request to classify new unlabeled instances is made. Whenever new instances arrive and need to be classified, the algorithm scans the entire training data. This can be costly and time consuming in cases of extremely large datasets and Data Streams (Perrizo, Ding, Khan, & Denton, 2007). Note that by sampling the stream, using clustering or windows, this downside may be eliminated (Su, 2011; Zhang, Jagadish, Dai, & Ramamohanarao, 2010).

2.2.4 Hoeffding Tree

In traditional decision trees, a multiple scan of the training data should be done in order to classify unlabeled instances. To overcome this problem and to give a decision tree algorithm the ability to handle Data Streams, Domingos and Hulten introduce the Hoeffding Tree (HT) (Domingos & Hulten, 2000). They refer to it as a VFDT, which stands for a Very Fast Decision Tree learner. This algorithm has the ability to handle the Data Stream and remain accurate by learning incrementally from the stream. They solved the previously mentioned problem by using Hoeffding Bounds, which is responsible for deciding the number of instances required in each node to make a statistical decision about splitting this node (Hoeglinger & Pears, 2007). Also, it works by performing one scan on the existing examples and building one model to represent the entire stream (Domingos & Hulten, 2000; Law & Zaniolo, 2005; H. Wang et al., 2003). The Hoeffding Bound is calculated using equation 2.5:

$$\textbf{Hoeffding bound } € = \sqrt{\frac{R^2 \ln(1/8)}{2nl}} \quad \textbf{(2. 5)}$$

As mentioned before, the Hoeffding Tree algorithm incrementally generates a decision tree from Data Streams. It examines each example as it arrives in the stream without the need to store them for further updates in the tree. The only memory needed to store is the tree itself. However, valuable information is stored in the leaves to help with the future growth at any time between the processing and training examples (Bifet & Kirkby, 2009b). Furthermore, the quality of the generated tree is the same as a tree generated with

batch learning algorithms, despite the fact that it was built in an incremental way (Bifet & Kirkby, 2009b). This is important to know, because batch learning decision trees, such as C4.5, are one of the best performers in machine learning models. However, in instance incremental learning, the method is trained on each instance as they arrive in the stream and without the need to access previous data (Polikar, 2006; Read et al., 2012).

Algorithm 4: Hoeffding Tree Pseudo-code

Let HT be a tree with single leaf (the root)
For all training examples **do**
    Sort example into leaf l using HT
    Update sufficient statics in l
    Incremental n_l, the number of examples seen at l
    If n_l mod n_{min} = 0 **and** examples seen at l not all of the same class then
        Compare $\overline{G_l}\ (X_l)$ for each attribute
        Let X_a be attribute with highest $\overline{G_l}$
        Let X_b be attribute with second-highest $\overline{G_l}$
        Compute Hoeffding bound € = $\sqrt{\frac{R^2 \ln(1/8)}{2nl}}$
        If $X_a \neq X_\emptyset$ **and** ($\overline{G_l}\ (X_a)$ - $\overline{G_l}\ (X_b)$ > € **or** € < τ) **then**
            Replace l with internal node that splits on X_a
            for all branches of the split **do**
                Add a new leaf with initialized sufficient statics
            end for
        end if
    end if
end for

Algorithm 4: Pseudo-code of Hoeffding Tree Algorithm

As mentioned before, the Hoeffding Tree algorithm utilizes the fact that choosing an optimal splitting attribute can be done using a small sample (Bifet, Holmes, Kirkby, & Pfahringer, 2010). This classifier monitors the performance of each branch on the tree. When the accuracy decreases, it replaces the branch with a more accurate one. The Hoeffding Tree pseudo-code given in algorithm 4 (Bifet & Kirkby, 2009b). To illustrate, this algorithm starts by working on each single root node then every example is assigned to the appropriate leaf. Each leaf will contain the important information that will help

grow the tree later if needed. This information includes the statistics about the number of class labels that would apply to each attribute. The Information Gain function mentioned before is Section 2.2.1 decides where to split the tree.

The accuracy of Hoeffding Tree is consistent in different types of problems, which makes it popular in classification problems related to Data Stream. The systems before the Hoeffding Tree worked by applying batch learning and performed multiple passes over the data (Bifet & Kirkby, 2009a). This type of learning requires an extra storage to store the entire data; therefore, they are not suitable for Data Stream where instances arrive rapidly in high volume. Domigos and Hulten then found that other systems that were designed for incremental learning also did not perform well in practical use. The Hoefdding Tree solves this problem since it builds the tree by inspecting each instance in the stream one time only (Bifet & Kirkby, 2009b).

In general, the algorithm is easy to understand and build. Moreover, it is easy to update and grow the tree. The update can be done at any time while processing training examples. However, one of the technical drawbacks of this algorithm is the memory limitation. Although the stored data includes the tree itself and the sufficient information in the leaves, the size of the stored data may increase as the data increases. This means that when dealing with large Data Stream, large storage is required. In some situations, the storage limitation could be a technical problem (Trambadiya, 2012). Also, the computational time depends on the time the algorithm needed to calculate and store information. Hence, in large Data Stream this could be an issue. Another drawback is the aging problem. The algorithm a uses time window based on keeping the training instances. As the data moves out of the window zone, the information is considered aged and will be lost. This information is important because in some environments, patterns can reappear over a period of time (Hoeglinger & Pears, 2007).

2.2.5 Discussion

In this chapter, the concept of Data Stream mining is introduced. Data Stream mining has specific characteristics that any leaning algorithm should adapt to. One of the main challenges in Data Stream mining is to cope with the limited resources of memory. Also,

the data distribution is changeable as the training examples arrive in different times. The learning algorithm for this type of data should also require performing one scan through the entire data set and be able to process each example as it arrives.

Next, in Section 2.2, the strength and limitation of the four methods that are used in this thesis in terms of classifying Data Streams are discussed. The C4.5 method is presented in Section 2.2.1, the overfitting issue can happen in this method because of empty branches or noisy data. Although this algorithm is easy to understand and is one of the most used algorithms in data mining, its accuracy in a Data Stream setting is poor since it must performs multiple scans on the entire training data before calculating the split point. Therefore, one solution for this problem is to use windows. In general, C4.5 algorithm will perform well if the window size is chosen correctly and the stream has no Concept Drift in it. Also, an incremental decision tree was introduced to overcome these problems. The Hoeffding Tree solves these problems by using Hoeffding bounds as introduced in Section 2.2.4. It requires only one scan of the existing examples to build one model that represents the entire existing stream. Hence, it processes each example as it arrives in the stream. Next, this example is deleted since it is not needed for further update.

Another method is the Naïve Bayes method. Like C4.5, this classifier is easy to understand. Moreover, this method is known for its ability to incrementally learn from the training data. It treats each feature independently. This means that the algorithm uses the present feature of each new example in order to classify it, regardless of the lack of the other features. Hence, as mentioned in Section 2.2.2, this character can work for or against this classifier. It gives it the ability to handle Data Streams, but at the same time it can mislead the classification process by ignoring some important features.

The third classifier method presented in this chapter is k-Nearest Neighbor. Similar to Naïve Bayes, this algorithm can handle Data Stream since it is an instance-based learner. This means that it does not build the classifier until a classification request is made. It is only processed every instance as it arrives and calculates the distance to be stored in the distance matrix. Therefore, this classifier tends to be slow when dealing with large datasets or Data Streams. One of the challenges in using this algorithm is choosing the right value for k; choosing a small k value can increase the classifier sensitivity to noise

whereas a large k would include too many instances from other classes. The next section will discuss the Ensemble Learning concept and its techniques.

The following Table 1 shows a summary of the classification method used in this thesis in regards to the Data Stream environment.

Table 1: Classification Methods Advantages and Disadvantages

Method	Advantages	Disadvantages
C4.5	- Produces accurate ensemble - Is suitable for parallel and distributed learning	- Needs entire data to calculate the splitting point - Needs to perform multiple scans
Naïve Bayes	- Learns incrementally - Features independently - Requires small amount of training data	- Has the Independent Feature assumption
k-Nearest Neighbor	- Instance based learner - Highly adapted behavior to new unlabeled data - New examples require only updating the distance database	- Is time and memory consuming - To classify new instances, the entire training data needs to be scanned
Hoeffding Tree	- Works incrementally - Performs one scan of the existing data	- Has memory limitations - Is high in terms of computational time - Uses time window to keep the training instances

2.3 Ensemble Learning

Ensemble Learning is the process of combining multiple models, such as classifiers or experts, to solve a computational problem. The main objective of using Ensemble Learning is to improve the model performance, such as classification and prediction accuracy (Read et al., 2012; Witten & Frank, 2005). Different errors result from different classifiers, combining these classifiers results in an ensemble with more accurate decisions (Ko, Sabourin, & de Souza Britto Jr, 2007). However, Ensemble Learning is also used in incremental learning. Recall that in Incremental Learning the classifier should have the ability to learn information about new instances without forgetting the previously gained knowledge, and without the need to access previous data (Polikar,

2006). Incremental Ensemble Learning is used in Data Streams. Hence, when using non-incremental Ensemble Learning with large amounts of data, the data will be partitioned into smaller subsets. The classifier proceeds to train each subset separately. Finally, the resulted models will be combined using some combination rule. Hence, if the data is too small, a random sample with replacement will be done and trained by different classifiers.

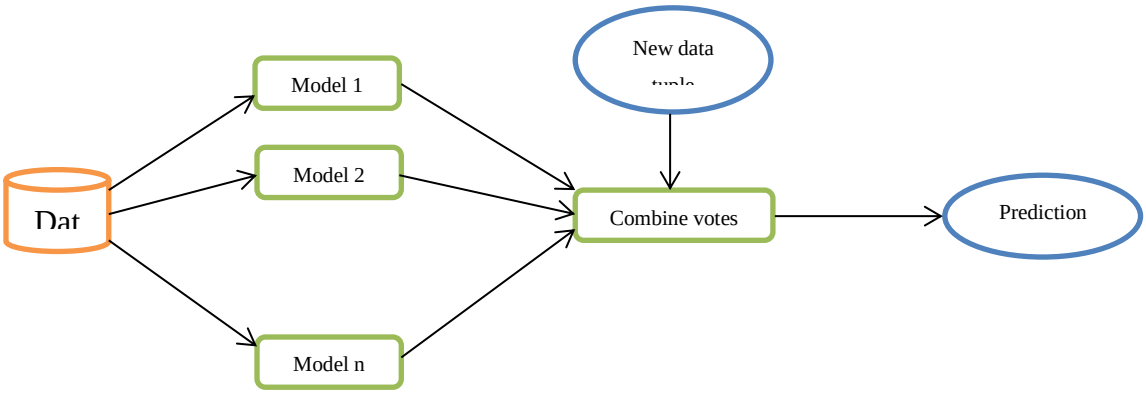

Figure 2: General Idea Behind the Ensemble Learning Process.

Using the Ensemble of Classifiers, there are two challenges. One is selecting classifiers from a pool of different ones. The second is choosing the right rule to combine the classifiers output (Ko et al., 2007). In general, during the classification stage, the committee will be developed by a number of hypotheses and a decision will be made by using voting, weighted voting or selecting the most relevant hypothesis (Melville & Mooney, 2003; Tsymbal, 2004). Different models will result from the Ensemble Learning processes. However, these models are only effective if they are not complementing each other (H. Wang et al., 2003). In general, the entire Ensemble Learning methods work to improve the generalization error, which is determined by the diverseness of the ensemble (Melville & Mooney, 2003). In case of classification problems, each individual classifier votes and the prediction result will be based on the vote collection. As mentioned before, Ensemble Learning is used to improve the classification accuracy. This happens because if a single classifier predicts the wrong class value, the Ensemble Learning will take into consideration the entire vote from all the trained classifiers. In this case, if one is

incorrectly classified, the other correctly classified results will overcome it. Figure 2.1 shows the general idea of how Ensemble Learning works.

In general, creating a single hypothesis from one learning algorithm has three problems. These problems can be divided into statistical, computational and representational problems. The first one exists when the searching space of hypotheses is too large compared to the amount of training data. This type of problem is referred to as "*high variance.*" The second problem is when the learning algorithm has a difficulty finding the best hypothesis within the hypothesis space, which is referred to as "*computational variance*". The final one is when the hypothesis space does not contain the right hypothesis to solve the problem; it is referred to as "*high bias.*" However, the Ensemble Learning helps reduce the impact of these problems (Arbib, 2003).

2.3.1 Boosting

The Boosting algorithm originated from *Probably Approximately Correct* (PAC) learning, which is a mathematical analyzing structure of machine learning (Freund & Schapire, 1997). The idea of Boosting started by converting a random guessing in PAC learning to a more accurate one. It works by building models in a continuous number of iteration. Also, it uses a number of weak classifiers; therefore, each model helps improve the accuracy of the previous model by reducing the weight error for each instance. Hence, the incorrectly handled instances in the previous model will be corrected in the following models and so on (Witten & Frank, 2005). By doing so, this method helps the weak learning classifier to become a stronger one. A weak learner is a classifier that hardly corresponds to the correct classification. However, Boosting improves the power of a weak classifier by combining a set of classifiers after updating a given weight to each of the training instances.

The AdaBoost algorithm is one of the most wildly used Boosting algorithms. It was developed by Freund and Schapire in 1996 to extend Boosting to multi-class and regression problems. Basically, it uses a random number of base learners on the same training set. A weight value is assigned to each instance and the probability distribution is maintained over the training set. After that, this value is increased if the instance is incorrectly classified and their weight will have a total of half of the total weight of the

training set. The other half is used for the correctly classified instances. The higher the weight, the more ineffective the instance on the classifier would be (J Ross Quinlan, 1996).

The algorithm starts by taking training set $(x_1, y_2), \ldots\ldots, (x_n, y_n)$ as an input. Each x_i belongs to instance space X and each y_i belongs to domain space Y. T is the number of iteration that the algorithm will run in which the base model passes through the training set. M is the number of the base model used in the ensemble. Algorithm 5 shows the AdaBoost algorithm (Han et al., 2006).

Algorithm 5: AdaBoost, a Boosing Algorithm

Input:
- D, a set of d class-labeled training tuples.
- k, the number of rounds (one classifier is generated per round).
- A classification learning scheme.

Output: A composite model

Method:

Initialize the weight of each tuple in D to $1/d$;

for i=1 to k **do** // for each round:
  sample D with replacement according to the tuple weights to obtain D_i;
  use training set D_i to derive model, M_i;
  compute error (M_i)
  if error $(M_i) > 0.5$ **then**
    go back to step 3 and try again
  endif
  for each tuple in D_i what was correctly classified **do**
    multiply the weight of the tuple by $error\ (M_i)/(1 - error(M_i))$;//update weight
  normalize the weight of each tuple;
endfor

Algorithm 5: AdaBoost Pseudo-code

During a single iteration, the AdaBoost algorithm calls a weak base learning algorithm. An important thing to notice about this algorithm is that it will assign an equal weight for all the instances in the training set in the first iteration. After that, on each of following

iterations the weight will be increased only if the instance is misclassified and decreases the correctly classified instances and so on. By increasing the weight of the misclassified instances, the new classifier is forced to focus on correcting them with lower weight (Witten & Frank, 2005). In general, each model of AdaBoost requires $M(T+1)$ to pass through the entire training set (N. Oza, 2001). This makes it unsuitable for Data Streams where the entire data set is not available for each iteration. Recall that in the Data Stream setting, a subset of the Data Stream, typically window, is also taken into consideration. However, there are two additional challenges regarding using AdaBoost in Data Stream. First, as mentioned before, this algorithm needs to perform multiple passes on the data set. The second challenge is the need to have the size of the data set in order to calculate the weight (de Souza & Matwin, 2013). An updated version of AdaBoost is presented by Oza and Russell (N. C. Oza & Russell, 2005), they use a Poisson parameter $\lambda = 1$ to assign a weight for each training example. More details about the online version of Boosting are given in Section 3.1.5.

The AdaBoost approach has several advantages and disadvantages. On one hand, it is fast and simple. It is flexible to combine with any method to find a base classifier since it requires no previous knowledge about the learner (Freund, Schapire, & Abe, 1999). Also, it is able to identify noise and outliers. This ability comes from the way it works by focusing on the weight of the instances; the instances with high weight are often outliers. On the other hand, one drawback of the AdaBoost approach is its sensitivity to outliers; the increased number of outliers has a negative effect on the AdaBoost performance. Therefore, much research has been done to solve this problem. One piece of research by Friedman et al. (Friedman, Hastie, & Tibshirani, 2000) introduces "*Gentle Adaboost*" algorithm to minimize the error rates in the outliers. In another work, Freund introduces "*BrownBoost*" as a solution to solve the AdaBoost sensitivity to noise and outliers (Freund, 2001). In a recent work, Wang et al. introduce new approach named "noise-detection based AdaBoost (ND-AdaBoost);"it is based on correctly classifying non-noisy instances and training the noisy instances (Cao, Kwong, & Wang, 2012). Another drawback is the growing knowledge during building models. Adaboost is considered an an incremental ensemble since it updates the knowledge by adding a new classifier to the previous model. The incremental learning makes it suitable for the Data Stream mining

setting. However, as the data grows like in Data Streams and extremely large datasets, it causes an increase in the storage memory for the decision model, and it can cause over-fitting (Rastgoo, Lemaitre, Palou, Miralles, & Casale, 2012). Also, as mentioned before, to calculate the weight for each instance, the algorithm needs to have the entire training data, which is not possible in Data Streams (N. C. Oza & Russell, 2005). Overall, the accuracy of the AdaBoost algorithm is influenced by the sufficiency of the given data, the complexity of the base classifier, or the weakness of the base classifier (Freund et al., 1999; Schapire, 2003).

2.3.2 Bagging

Unlike Boosting, the Bagging algorithm can be used for both classification and regression. In 1996, Leo Breiman developed the Bagging algorithm, which is also known as Bootstrap aggregating (Breiman, 1996). It was developed to improve the algorithm accuracy by producing diverse models and then combining them to produce the final model. However, when it was first developed, the experimental result by Breiman showed that Bagging can improve the performance of unstable learners, but decrease the performance of a stable one (Breiman, 1996). In the unstable learner, any small changes in the training data cause large changes in the resulted hypothesis (Dietterichl, 2002). Moreover, in an experimental comparison of the performance of Bagging, Boosting and Randomization shows that Bagging performs better in the presence of noise (Dietterich, 2000b).

In general, the Bagging algorithm works by splitting the data into two sets: the training and testing sets. Given a training set of size N, the algorithm will run for a number of iteration t where $t = 1, \dots . T$ will be set to form the total number of the training set replications (N. Oza, 2001). These sets are randomly created by selecting uniformly n samples with replacement instances from the original dataset S, this will give the instances an equal probability to be selected (N. Oza, 2001). The resulted datasets are called bootstrap replicates and they are at the same size as the original dataset S (J Ross Quinlan, 1996). The probability of choosing any example is governed by *Binomial* distribution (Bifet & Kirkby, 2009a). However, some instances are represented in the dataset more than one time whereas around 0.368 of data points are not selected in the

training sets and will be used as test data (Ye, 2003). Finally, the base learner is applied to each bootstrap to produce an ensemble of the T hypothesis. See algorithm 6 for more details on how Bagging works (Han et al., 2006).

Algorithm 6: Bagging Algorithm

Input:

- D, a set of d training tuples;
- k, the number of models in the ensemble;
- a classification learning scheme.

Output: the ensemble – a composite model, $M*$

for I = 1 to k **do** // create k models:
create bootstrap sample, D_i, by sampling D with replacement;
use D_i and the learning scheme to derive a model, M_i;
endfor

to use the ensemble to classify a tuple, X:
let each of the k models classify $\boldsymbol{X}$ and return the majority vote;

Algorithm 6: Bagging Pseudo-code

In terms of Data Streams, there are some challenges that need to be addressed in order for this algorithm to work. Hence, as in Boosting, the Bagging algorithm has the same challenges regarding Data Stream. As mentioned before, the first challenge is that the algorithm needs to run multiple passes on the data set. The second one is the requirement of knowing the dataset size (de Souza & Matwin, 2013). One of the Bagging parameters is the size of the training set N. The Data Stream is not entirely available to be partitioned into the training and test sets as in the traditional mining. Therefore, Oza and Russell (N. C. Oza & Russell, 2005) proposed an online version of Bagging; in this version each example is assigned a weight according to the Poisson(1) distribution, instead of using sampling with replacement. Also, the algorithm simulates the bootstrap process by creating K copies of each newly arrived instance then updating each base model (N. Oza, 2001). The online version of Bagging is presented in Section 3.1.3.

Overall, the chosen base model will be trained to result in M number of models. The final prediction will be made through combining all the M trained models. Since this method can be used for both classification and regression, there are two possibilities to predict the final model. In the case of classification, voting will combine the output models whereas

the output models will be averaged in the case of regression (Sewell, 2008). The objective of the Bagging algorithm is to enhance the accuracy of one model by using the produced replications. Also, averaging the misclassification error of different models helps improve the accuracy of the final predicted model.

Similarities and Differences between Bagging and Boosting

Both classifiers use the same combining method to produce the outcome of each model. In classification, they both use voting; and in numeric attribute, they both use averaging. Also, both of them combine models of the same type. However, in Bagging, each model is built separately, whereas in Boosting each model depends on the performance of the previous model. Each new model in Boosting helps correct the incorrectly classified instances in the previous model. Also, these two algorithms need some modification to work in the Data Stream setup. Finally, in Boosting, the model attributes are weighed according to their performance, but in Bagging it is assigned to equal weight (Witten & Frank, 2005). In terms of Data Streams, both approaches need to work with the limited recourses of time and memory. Also, they need to be able to adjust to the change in the data distribution in the stream (Bifet, Holmes, Pfahringer, & Gavaldà, 2009). Therefore, several approaches were presented to adjust them in a way they will perform well with the Data Stream. The online approach presented by Oza and Russell (Bifet & Kirkby, 2009b; N. C. Oza & Russell, 2005) named OzaBag and OzaBoost are used in this thesis; for this reason more details about them will be given in Sections 3.1.3 and 3.1.5.

2.3.3 Stacking

In 1991, Wolpert introduced Stacked Generalization or (Stacking); his goal was to "minimize the generalization error rate of one or more generalizer" (Wolpert, 1992). This method can also be described as a complicated version of cross-validation. In the cross-validation process, on one hand, data is divided into k number of subsets. Each iteration uses one subset for testing and the rest of the k subsets for training the classifier (Refaeilzadeh, Tang, & Liu, 2007). On the other hand, Stacking works by combining different types of models that have been learned in the classification task.

The difficult analysis of the theory of this algorithm makes it less popular when compared to Bagging and Boosting. Also, in practice, there is no general best way of using it (Witten & Frank, 2005). It is applied to models built by different algorithms and the voting combines the output. However, this is not reliable since the algorithm does not know which classifier to trust. Instead, it uses a meta-learner which helps the algorithm decide the best way to combine the base learner output (Witten & Frank, 2005).

Algorithm 7: Stacking Algorithm

Input: Data set $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$;
First-level learning algorithm $\mathcal{L}_1, \dots, \mathcal{L}_T$;
Second-level learning algorithm $\mathcal{L}$
Process:
For $t = 1, \dots, T$;
$h_t = \mathcal{L}_t(D)$ %Train a first-level individual learner h_t by applying the first-level
end; %learning algorithm $\mathcal{L}_t$ to the original data set D
$\boldsymbol{D'} = \emptyset$ %generate a new data set
for $i = 1, \dots, m$;
for $t = 1, \dots, T$;
$z_{it} = h_t(x_i)$ %use h_t to classify the training examples x_i
end;
$D' = D' \cup \{((z_{i1}, z_{i2}, \dots, z_{iT}), y_i)\}$
end;
$h' = \mathcal{L}(D')$ %Train the second-level learner h' by applying the second-level
%learning algorithm $\mathcal{L}$ to the new data set D'
Output: $\boldsymbol{H(x) = h'(h_1(x), \dots, h_T(x))}$

Algorithm 7: Stacking Pseudo-code

Practically, the algorithm starts by splitting the original set into two subsets. The first subset is used to train the base learners and the second subset to test the learners' behavior. In the first phase, a new set of data is collected from each output model. This data represent the models' prediction of the instances' classes with their correct classification. The instances with unknown classification, which are under investigation, are not included in this first phase and the resulted data will be used in the next one. In the second phase, the outcome prediction or the models from the testing run is then used as an input to produce the correct prediction model (Sewell, 2008; Ting & Witten, 1997).

According to Wolpert's terminology, the first phase consists of level-0 data, which is the original data and level-0 models, which are the output models. The second phase includes level-1 data that is the resulted models from phase one and level-1 generalizer, which is the learning algorithm in this phase. See algorithm 7 for more details on how it works (Zhou). As in all Ensemble Learners, to work with Data Streams, it has to utilize memory (Bifet et al., 2009). Stacking works by partitioning data into subsets; in the Data Stream the algorithm has to portion an infinite number of data and store it. This requires having a large memory, which might not be practical in practice. Besides the Stacking algorithm analyzing difficulty (Witten & Frank, 2005), it is known for its ability to improve single classifier accuracy and reduce the generalization error (Wolpert, 1992). Other studies showed that Stacking is time-consuming and expensive (Fan, Chan, & Stolfo, 1996).

2.3.4 Random Subspace Method

The Random Subspace method, also known as Attribute Bagging (AB), works by using several classifiers to create the output of each individual model. Furthermore, this classifier is considered to be a wrapper method that is able to work with any learning algorithm (Bryll, Gutierrez-Osuna, & Quek, 2003). The main objective of this classifier is to improve the accuracy and stability of the classifier. It is a good choice for a classification problem that includes a larger number of attributes, compared to the number of the training set. Originally, it was developed for decision trees. In general, it can be said that AB is a method based on random forest algorithm in which a random subset of identifiers is used to grow the classification tree. However, it can be used to improve other unstable classifiers.

Basically, it works by defining the number of input variables for each classifier (represented by X_j for each training example). Then, for each classifier a number of training sets are randomly chosen without replacement, where S is a p-dimentional vector $X_j = (x_{j1}, x_{j2}, \dots, x_{jp})$. This will create the training set of p^*-dimentional random subspace of the original feature space, which will be used to build the ensemble classifier. Finally, voting produces the final output. The AB method is shown in algorithm 8 (Panov & Džeroski, 2007).

Algorithm 8: Random Subspace Method

Input: Training examples S, Number of subspaces B, Dimension of subspace p^*
Output: Ensemble E

$E \Leftarrow 0$
For $i = 1$ *to* B **do**
    $\widetilde{Si} \Leftarrow SelectRandomSubspace(S, p^*)$
    $\widetilde{Si} \Leftarrow ConstructClassifier\ (\widetilde{Si})$
    $E \Leftarrow E \cup \{C^i\}$
endfor
return E

Algorithm 8: Random Subspace Method Pseudo-code

In this method, each base learner will be trained only on a subset of the data, which was randomly selected. This will give it the advantage to adjust to Data Stream environment, where instances arrive in different times. In Data Stream, the algorithm will randomly create subsets as the data arrives and every base learner will be weighed by the subset it was built on (Hoens, Chawla, & Polikar, 2011). In general, the strength of this method comes from two facts: the fact that it uses random subsamples, as in Bagging; and that algorithms in the learning base level classifier are chosen randomly as in decision trees (Panov & Džeroski, 2007). By constructing the classifier randomly, it solves the small sample problem when the number of training sets is small. Also, the randomization performance may be better than Bagging in the case of very large datasets, which make it suitable for mining Data Streams (Dietterich, 2000a). However, one downside of this algorithm is present when sampling the original data. In order to result in good classification accuracy, a large number of attributes is needed, which cause an increase to the computational time (Y. Zhu, Liu, & Chen, 2009). Also, the computational intensity of this algorithm comes from the need to weigh each model according to the subset it was built on (Mielniczuk & Teisseyre, 2014). Hence, this algorithm is not used in this thesis because of the memory limitation and the computational complexity of it.

2.3.5 Discussion

In this section the Ensemble Learning concept is discussed. Ensemble Learning is the process of combining multiple models which aims at improving one model performance. Also, it can work with both static data and Data Stream. However, in the Data Stream setting, the classifier should be able to learn information about new instances without the need to access old data. Also, four different Ensemble approaches are presented in this chapter such as Bagging, Boosting, Stacking, and the Random Subspace Method.

As mentioned before, the main idea of Boosting is to improve the performance of the previous model by correcting the error weight for each instance. One of the most wildly used Boosting algorithms is AdaBoost. Section 2.3.1 presents this algorithm along with its strengths and weaknesses in the Data Stream setting. One of the limitations in the Data Stream setting is that the algorithm needs to perform multiple scans of the entire training data. Another limitation is the requirement of data size to calculate the weight. However, Oza and Russell solve these limitations when they created an on-line version of Boosting.

The second ensemble approach presented in this section is the Bagging approach. It increases the accuracy of a single model by combining the output of the produced diverse model. Like Boosting, this algorithm also requires the knowledge of the training set size and to perform multiple scans on the entire training set. The training set size is needed to execute the bootstrap process. Also, Oza and Russel introduced an on-line version of Bagging named OzaBag. Both OzaBoost and OzaBag are presented in the next chapter in Section 4.2.

Two more ensemble methods are presented in this chapter; namely Stacking and the Subspace Random method. However, these two methods are not used in this thesis but are presented to give more knowledge about different ensemble methods. In order to work, Stacking needs to partition the entire dataset. This requires a large amount of memory, which is not practical in practice. This algorithm is not used in this thesis because of its inability to work in an Active Learning framework. On the other hand, Random subspace method characteristics make it suitable for the Data Stream setting. It

works by creating random subsets as the data arrive in the stream then every model is weighed according to the subset it was built on. The next table shows the advantages and disadvantages of the ensemble method explained in this chapter:

Table 2: Ensemble Methods Advantages and Disadvantages

Method	Advantages	Disadvantages
Boosting	- Works incrementally	- Performs multiple passes - Requires previous knowledge of data size - Needs to be adapted to handle Data Streams
Bagging	- Produce diverse models	
Stacking	- It is hard to be applied in Data Stream setting; hence, there are no advantages to be listed	- Has memory limitations
Random Subspace method	- Trained on subsets of the data	- Has memory limitations - Has computational complexity

2.4 Summary

In summary, this chapter reviews the Data Stream mining concept. As mentioned, the main challenges for a Data Stream algorithm are its ability to process data in real-time, and to work in limited memory recourses by scanning the data only once to extract information. Four classification methods are presented in this chapter; namely C4.5, Naïve Bayes, k-Nearest Neighbor, and Hoeffding Tree. The main goal of these methods is to predict the outcome using a given input. We discussed the advantages and disadvantages of these methods, with respect to learning from Data Streams.

Also, this chapter introduces the Ensemble Learning concept and techniques that combines multiple models to improve the performance of a single classifier. Boosting, Bagging, Stacking, and the Random subspace method are the four ensemble methods discussed in this chapter. There are several weaknesses for these methods when used in Data Streams. Therefore, the next chapter will present the Active Learning concept and

techniques, which can handle Data Streams. The Active Learning techniques discussed in the next chapter includes Query by Committee, Query by Bagging, and Query by Boosting. Their strength and weakness in terms of the Data Stream setting will be discussed. Also, two online learning methods will be presented in our next chapter. These two algorithms are developed by Oza and Russell and will be used in our framework.

Chapter 3

Active Learning and Online Learning

Active Learning, also known as Query Learning, is a sub-discipline of machine learning and artificial intelligence. The basic idea of Active Learning is to allow the user to select the most informative examples by using multiple classifiers. Informative examples are the ones that cause high disagreement among classifiers (Stefanowski & Pachocki, 2009). In most of the learning algorithms, the algorithm uses a number of labeled instances as a guide to label the unlabeled ones. With large databases and Data Streams, this task is considered to be costly and time inefficient. There are different scenarios in which Active Learning is beneficial. One scenario is using a labeled instances pool; the algorithms query for information from the domain expert, the answered queries will result in building trained models. By using these models, the algorithm is now able to label the coming unlabeled instances.

The main idea of Active Ensemble Learning is using the diversity of Ensemble Learning to create a number of ensembles (models). This usually works by taking some information of the data from the user or any other source of information to start the prediction process at some point of incoming data. In other words, the algorithm will start with limited amounts of labeled data, which include the most informative instances from the coming data. Then it passes them to the learning algorithm as a training set to produce the first classifier. In each of the following iterations, the algorithm analyzes the remaining unlabeled instances and presents the prediction to the *oracle* (human expert) to label them. These labeled examples are added to the training set and used in the following iteration. This process is repeated until the user is satisfied or until a specific stopping criterion is achieved (Stefanowski & Pachocki, 2009). In this technique, a strong learner is used as a base learner to create the committee [9]. A committee contains a voted result of training different models on the current labeled data. Some of the well-known

examples of the application areas of Active Learning are Web mining, email filtering systems, and text categorization (Balasubramanian, Chakraborty, Ho, Wechsler, & Panchanathan, 2014).

There are a number of Active Learning algorithms. Although these algorithms use the same Ensemble algorithms such as Bagging, Boosting, and Stacking, they differ in the way they create the committee. Further details on these algorithms are given in the following sections.

3.1 Query Learning Methods

3.1.1 Query by Committee

The Query by Committee (QBC) algorithm, introduced in 1992 (Seung, Opper, & Sompolinsky, 1992), uses random learning algorithms to produce different copies of hypotheses. The hypotheses are randomly selected using the Gibbs algorithm, which is a randomized algorithm that selects the hypothesis randomly from a pool of hypotheses according to the prior distribution and makes predictions according to it (Freund, Seung, Shamir, & Tishby, 1997; Mamitsuka, 1998; Zhou, 2012). Algorithm 9 presents the Query by Committee approach (Melville & Mooney, 2004).

Algorithm 9: Query by Committee Algorithm

Given: T – set of training examples
U – set of unlabeled examples
$BaseLearner$ – Base learning algorithm
k - number of selective sampling iteration
m – size of each sample

1. Repeat k times
2. Generate a committee of classifier,
3. $\forall\, x_j \in U$, compute Utility (C^*, x_j), based on the current committee
4. Select a subset S of m samples that maximize utility
5. Label examples in S
6. Remove examples in S from U and add to T
7. Return EnsembleMethod(BaseLearn, T)

Algorithm 9: Query by Committee Pseudo-code

In a specific number of iteration, a pool of unlabeled examples is used as a training set. The algorithm starts by creating a set of hypotheses in each iteration and then evaluating the utility of each unlabeled example. As a result, it will select the examples with the highest utility value and move them to the training set. The processes will continue until reaching the total number of required labels (Melville & Mooney, 2004).

It is hard to apply Query by Committee in practice, because of the algorithm's specific requirement (Mamitsuka, 1998; Seung et al., 1992). In addition, this learning algorithm tends to be difficult in data processing because it works by using the Gibbs algorithm. Also, the Gibbs algorithm is an intractable algorithm which causes increasing the algorithm's computational complexity (Freund et al., 1997). To overcome this problem, new query learning algorithms were developed using the Bagging and Boosting algorithms (Mamitsuka, 1998). Abe and Mamitsuka proposed the Query by Bagging and Query by Boosting algorithms in 1998; further details will be given in the following two sections.

3.1.2 Query by Bagging

This method combines the Query by Committee and Bagging techniques. The Query by Bagging (QBag) algorithm overcomes the problem of using the Gibbs algorithm, which is in Query by Committee, by using the Bagging algorithm and using bootstrap sampling. Therefore, the Bagging algorithm works by dividing the input data into subsamples, using identical distribution. Also, the variance of the prediction algorithm is related to arbitrary resampling (Mamitsuka, 1998).

At the beginning of the procedure and at each iteration, bootstrap sampling is performed. Recall that Bootstrap refers to sampling with the replacement of the training dataset, which gives each instance an equal probability to be selected (N. Oza, 2001). This will result in a number of hypotheses that will be used to select the next examples. Averaging the output of the resulted hypotheses makes the final prediction. Therefore, the query is run for a number of iteration and each one results in a new hypothesis. The next starting point in each iteration is where the models are uniformly divided (Mamitsuka, 1998). More details about the QBag procedure and its pseudo-code are given in algorithm 10. Adding to the previous mentioned benefits of using QBag, the algorithm has the ability to

reduce the prediction error and improve the prediction accuracy of new data compared to the QBC algorithm (Mamitsuka, 1998).

Algorithm 10: Query by Bagging algorithm

Input: Number of trials: N
Component learning algorithm: A
Number of times re-sampling is done: T
Number of query candidates: X
A set of query points: Q
Initialization: $S_1 = \langle x_1, f(x_a) \rangle$ for random x_1
For $i = 1, \ldots . . N$

1. By resampling according to uniform distribution on S_i, obtain sub-sample $S'_1, \ldots . , S'_T$ each size m.
2. Run A on each sub-sample and obtain $h_1, \ldots \ldots , h_T$.
3. Randomly generate a set of R points $C \subset Q \backslash S'_i$. with respect to uniform distribution over $Q \backslash S'_i$.
4. Pick a point $x^* \in C$ split most evenly: $x^* = arg$
$$min_x \in C \, \| \{t \leq T | h_t(x) = 1\} - |\{t \leq T | h_t(x) = 0\} \|$$
5. Query the function value at x^* and obtain $f(x^*)$.
6. Update the past data as follows $S_{i+1} = append\ (S_i, \langle x^*, f(x^*) \rangle)$

End for

Output: output as the final hypothesis:
$$h_{fin}(x) = \arg max_{y \in Y} |\{t \leq T | h_t(x) = y\}$$
Where h_t are hypotheses of the final (N^{-th}) stage.

Algorithm 10: Query by Bagging Pseudo-code

3.1.3 Query by Boosting

Similar to QBag, this method combines the Query by Committee technique with Boosting. The idea of this algorithm is to repeatedly resample the training data using sampling distribution that changes after each iteration (Mamitsuka, 1998). The Boosting algorithm enhances the performance of the weak algorithm by repeatedly improving the result of the previous model until reaching a satisfactory result. The Query by Boosting (QBoost) runs for a number of iterations with the start point of the next iteration being unknown. It is chosen depending on the properties of the resulted models (Mamitsuka, 1998).

In this algorithm, the training data will be formed using re-sampling. The re-sampling distribution will be related to the properties of each hypothesis. Consequently, weighted voting of the entire collected hypothesis influences the variance of the algorithm. Algorithm 11 shows the Pseudo-code of the QBoost algorithm (Mamitsuka, 1998). Also, as in Boosting, QBoost is sensitive to outliers and noise.

Algorithm 11: Query by Boosting algorithm

Input: Number of trials: N
Component learning algorithm: A
Number of times re-sampling is done: T
Number of query candidates: R
A set of query points: Q
Initialization: $S_1 = \langle x_1, f(x_a)\rangle$ for random x_1
For $i = 1, \ldots . . N$

1. Run AdaBoost on input (S_i, A, T) and get: $H_{fin}(x) = \arg max_{y\in Y} \sum_{ht(x)=y} \log \frac{1}{Bt}$
2. Randomly generate a set of R points $C \subset Q \backslash S'_i$. with respect to uniform distribution over $Q \backslash S'_i$.
3. Pick a point $x^* \in C$ with minimum margin:

$$x^* = \arg min_{x\in c} \left| \sum_{ht(x)=0} \log \frac{1}{Bt} - \sum_{ht(x)=1} \log \frac{1}{Bt} \right.$$

4. Query the function value at $x^*$and obtain $f(x^*)$.
5. Update the past data as follows $S_{i+1} = append\ (S_i, \langle x^*, f(x^*)\rangle)$

End for
Output: Output h_{fin} in the last stage as the output.

Algorithm 11: Query by Boosting Pseudo-code

3.2 On-line Learning Methods

3.2.1 Online Bagging - OzaBag

Oza and Russell invented a new version of Bagging (OzaBag) and Boosting (OzaBoost) (N. C. Oza & Russell, 2005). The OzaBag and OzaBoost are used for incremental on-line learning and Data Stream. In incremental on-line learning, after each new example arrives, the hypotheses are updated incrementally by the learning algorithm (N. Oza, 2001). Also, in on-line learning the algorithm will perform only one pass through the entire training set. For the Bagging method, they explain that the process of sampling bootstrap from training data can be simulated to work in the Data Stream context. They

also explain that the individual probability for choosing replicates follow Poisson(1) distribution.

The original Bagging method requires having the entire data set in order to create the bootstrap replicate. As mentioned in Section 2.3.2, the probability of choosing an example to be added to the bootstrap follows *Binomial* distribution. Also, the Bagging method requires knowing the number of examples in advance (Bifet & Kirkby, 2009a). Oza and Russell solve these two problems by suggesting that the number N of examples is infinite $N \to \infty$. Adding to this, they suggest that the *Binominal* distribution tends to $Poisson(1)$ distribution. The pseudo-code of OzaBag is listed in algorithm 12.

Algorithm 12: Oza and Russell's Online Bagging (OzaBag).

M is the number of models in the ensemble and $I(.)$ is the indicator function.
Initialize base model h_m for all $m \in \{1,2,\dots,M\}$
for all training examples **do**
  for $m = 1,2,\dots,M$ **do**
    set $K = Poisson(1)$
    for $n = 1,2,\dots,k$ **do**
      Update h_m with the current example
    end for
  end for
anytime output:
return hypothesis: $h_{fin}(x) = \arg max_{y \in Y} \sum_{t=1}^{T} I(h_t(x) = y)$

Algorithm 12: OzaBag Pseudo-code.

3.2.2 Online Boosting - OzaBoost

OzaBoost is an online version of the Boosting algorithm introduced by Oza and Russell (N. C. Oza & Russell, 2005). The goal of any online classifier is its ability to process every single example as it arrives (Bifet & Kirkby, 2009a). The OzaBoost algorithm works similarly to AdaBoost, it solves the previously mentioned challenges in AdaBoost when used in Data Stream. As mentioned in Section 2.3.1, AdaBoost requires knowing the training set size in order to calculate the weight. The AdaBoost divides the weight into two halves: half for the correctly classified and half for the incorrectly classified ones. In order to solve this, they modify it by using Poisson distribution (λ) which gives a random probability for each example to be used in training (Bifet & Kirkby, 2009a). In

this method, the Poisson distribution (λ) parameter assigned to each example will be increased if the base model misclassifies the instance and decreased if it correctly classifies it. Then, the new value will be presented to the next model (N. C. Oza & Russell, 2005). See algorithm 13 for more details about this algorithm (Bifet & Kirkby, 2009a).

As said before, OzaBoost is a modified version of Boosting. In the Data Stream setting, this algorithm uses a parallel boosting. This means that as every example arrives to the base Data Stream algorithm, the base algorithm will build different ensemble member in parallel (Bifet & Kirkby, 2009a).

Algorithm 13: Oza and Russell's Online Boosting (OzaBoost).

N is the number of examples seen.
Initialize base model h_m for all $m \in \{1,2,\dots,M\}, \lambda_m^{sc} = 0, \lambda_m^{sw} = 0$
for all training examples **do**
  set "weight" of examples $\lambda_d = 1$
  for $m = 1,2,\dots,M$ do
    set $k = Poisson(\lambda_d)$
    for $n = 1,2,\dots,k$ **do**
      update h_m with the current example
    end for
    if h_m correctly classifies the example **then**
      $\lambda_m^{sc} \leftarrow \lambda_m^{sc} + \lambda_d$
      $\lambda_d \leftarrow \lambda_d \left(\frac{N}{2\lambda_m^{sc}}\right)$
    else
      $\lambda_m^{sw} \leftarrow \lambda_m^{sw} + \lambda_d$
      $\lambda_d \leftarrow \lambda_d \left(\frac{N}{2\lambda_m^{sw}}\right)$
    end if
  end for
end for
anytime output:
Calculate $\epsilon_{m=\frac{\lambda_m^{sw}}{\lambda_m^{sc}+\lambda_m^{sw}}}$ and $\beta_m = \epsilon_m/(1-\epsilon_m)$ for all $m \in \{1,2,\dots,M\}$
return hypothesis: $h_{fin}(x) = \arg max_{y \in Y} \sum_{m:h_{m(x)=y}} \log 1/\beta_m$

Algorithm 13: OzaBoost Pseudo-code

3.3 Discussion

As mentioned in the previous chapter, ensemble techniques require multiple passes through the entire training data. Also, it requires increasing the memory size as the data increases which is not practical in Data Streams. Hence, several research projects were done to overcome these problems and produce new algorithms that are active and able to interact with the continuously arrived data. The Active Ensemble Learning process is one solution to learn from the stream. A diversity of Ensemble Learning methods is used to create a number of models. In Active Learning, the oracle (human expert) plays an important role in the learning process. Active Learning, known as Query Learning, also has several methods. In this chapter, three methods are presented; namely Query by Committee, Query by Bagging, and Query by Boosting.

The first method discussed in the chapter is QBC. However, this algorithm uses the Gibbs algorithm which causes an increase to the algorithm's computational complexity. Therefore, two query methods were developed to overcome the mentioned problem. QBag and QBoost use a combination of QBC with Bagging or Boosting algorithms. QBag has the ability to reduce the prediction error and to improve the prediction accuracy when compared to QBC.

In addition, this chapter introduced two on-line algorithms that were developed by Oza and Russell. These two algorithms were developed to enhance the performance of Bagging and Boosting and enable them to work incrementally in the Data Stream setting. Also, they work by scanning the training data once. In the case of OzaBag, the Poisson distribution is used instead of bootstrap sampling. The arrived instances are assigned with a random probability using Poisson distribution in case of OzaBoost . This will solve the problem of requiring the training data set to calculate the weight for each instance.

3.4 Summary

To summarize, this chapter presented the Active Ensemble Learning process. It also explained how Query Learning methods work. It addressed the problems in QBC that led to develop both QBag and QBoost. Adding to that, it presented two on-line learning

methods named OzaBag and OzaBoost . These two methods are used in this experiment and their results will be presented in the next chapter.

The next chapter will present the research methodology along with the methods that are used in. Also, we present the data sets that are used and the evaluation of all the used algorithms in this experiment.

Part II

Ensemble Learning through Active Learning

Chapter 4

A Framework for Online Active Ensemble Learning

This chapter presents the framework developed for this study, in order to study the strength and weaknesses of Ensemble-based learning as well as its extension to the online Active Learning paradigm. Our framework is divided into two main stages, namely Ensemble and Active Learning. Different methods and classifiers are used in this framework. Recall that Chapter 2 and 3 presented different types of algorithms of both Ensemble and Active Learning. In the Ensemble Learning stage, both Bagging and Boosting methods are used. OzaBag and OzaBoost are used in the Active Learning stage. Also, Chapter 2 presented the four classification methods used in this study.

The framework used in this thesis is inspired by a number of different research works. For example, the causality workbench is a project that aims at providing an environment for people including professors, students and researchers who are interested in creating and then testing machine learning algorithms. This website provides tutorials and papers on different types of machine learning and challenges including Active Learning ("The causality workbench, ," 2014). Also, a research paper on the de-duplication method in Active Learning was useful to come up with the final design used in this thesis (Muhivuwomunda, 2010).

The research design used in this thesis is an experimental design that tests the performance of Ensemble Learning and Active Ensemble Learning in a tightly controlled setting. Also, the resulted experimental data is used to compare the performances of Ensemble Learning versus Active Ensemble Learning, using the same method, algorithm, and parameters. Parameters include the ensemble size and setting the K value for the k-Nearest Neighbor algorithm, as introduced in Section 2.2.3. The ensemble size indicates

the total number of classifiers in the ensemble (Agrawal & Gupta, 2013). The resulted data is analyzed using the same quantitative statistical techniques. This will be explained in details in Section 4.4. Also, Cross-Validation is applied to the process to ensure the algorithms' performance.

Recall that the main goal of this framework is to evaluate the performance of Ensemble Learning and Active Ensemble Learning and to contrast their strengths and weaknesses. That is, we are testing the hypothesis that incorporating Active Learning into the Ensemble Learning framework will lead to an improvement in performance. Here, we measure the performance in terms of i) accuracy of the final mode, ii) time to train the model, and iii) the size of the Ensemble, Specifically, we are further interested in studying the number of instances that would result in a good performance of the learning algorithm. The number of instances that is considered in this evaluation is the size of the test sets. Moreover, we do evaluate the impact of increasing the ensemble size and the number of iterations on the resulted performance. That is, our goal is to study the interplay between the accuracy and efficiency of the algorithms. To this end, in Chapter 5 of this thesis, the resulted data will be used to answer the main research questions that are presented in Chapter 1.

The rest of this chapter is divided into four sections. Section 4.1 presents an overview of our Active Ensemble Learning framework. Also, it gives details about how it works along with running examples. The environment setup is discussed in Section 4.2. This includes the perpetration of the datasets and details about each one. Next, in Section 4.3, the evaluation criteria are discussed and the performance measures are presented. Finally, the statistical analysis tool that is used to validate the framework results is presented.

4.1 Overview of Active Ensemble Learning Framework

In this section we discuss the Active Ensemble Learning framework. The framework developed in this thesis uses the Ensemble Learning algorithm within the Active Learning process. The following is the Active Ensemble Learning pseudo-code followed by an explanation.

Active Ensemble Learning Pseudo-code

Input:
D: a set of d class-labeled training tuples; E:Ensembe method; N: Ensemble size
C_i: classifier; k, the number of models in the ensemble
Initialization for Ensemble Learning stage: $D_1 = \langle x_1, f(x_a)\rangle$ for random x_1
For i = 1,…, N

1- **if** E = Bagging
   then run Bagging algorithm with classifier C_i
   Else
   If E = Boosting **then run** Boosting algorithm with classifier C_i
   endif
2- **Obtain** a model
3- **Update** the past training data as follows $D_{i+1} = append\ (D_i, \langle x^*, f(x^*)\rangle)$
4- **Output** the final hypothesis h_i according to E
5- **If** D size is known and C = {C4.5 or Naïve Bayes}
   Output the final model
   Else
   If D size is unknown and C = {kNN or Hoeffding Tree }
   Continue to Active Ensemble Learning stage

Active Ensemble Learning stage:
Input:
h_i: Hypothesis obtained from the Ensemble Learning stage; D_i: Training data resulted from the Ensemble Learning stage; A: Active learning method; T_i: Test set; F_i: accumulated data; N: Ensemble size
M is the number of models in the ensemble
For all training examples do

1- **Test** T according to A on input (D_i, T_i, C_i) and obtain prediction values P_i
2- **Present** P_i to user – user choose 10 instance from each class with highest P_i value
3- **Update** F_i with output from step 2
4- **Run** A on input (F_i, A)
   If A = OzaBag **then run** OzaBag
   Else If A = OzaBoost **then run** OzaBoost
5- Output the final hypothesis according to A

Algorithm 14: Active Ensemble Learning Pseudo-code

As discussed in Chapters 2 and 3, the Ensemble Learning method allows the algorithm to combine multiple classifiers to improve the classification or prediction accuracy. However, Active Learning uses a number of labeled instances to guide the process of labeling the unlabeled ones. This process is done by presenting the informative instances to the user and using the user's feedback to build the trained models. Both learning methods have their own strengths and weaknesses; therefore, we decided to build a framework that allows combining the strengths of both methods. Also, to enable the algorithm to handle the vast amount of instances that is in Data Streams. The proposed Active Ensemble Learning framework works by using the diversity of the Ensemble Learning to create a number of models combined with the Active Learning ability to learn in online mode.

The presented pseudo-code is divided into two main stages: namely, the Ensemble Learning and Active Learning stages. In the first stage, there are four main inputs. Firstly, the training dataset, which contains a set of labeled instances and secondly, the ensemble size (recall that this size indicates the number of classifiers in each ensemble). Thirdly, the ensemble method and lastly the base classifier that are used in this particular process and the base learner classifier C_i are discussed. In this framework, we are using four different learning methods with four different classifiers. As mentioned before, the learning methods that are used are: Bagging, Boosting, OzaBag, and OzaBoost and the four classifiers used are C4.5, Naïve Bayes, kNN, and Hoeffding Tree. In each learning process we will be using one of the learning methods with one of the classifiers. The same process will be repeated four times with different datasets. In this way, each learning method will be tested on the same data set using four different classifiers. The whole process is repeated with different ensemble size. Each time the process is performed with one of the five ensemble sizes we set for this study which are 10, 20, 25, 50, and 100.

In the first stage, the algorithm will start with a limited amount of labeled data (training set D). In the following section we will discuss in detail how this training set is chosen. The algorithm will then pass these labeled instances to the learning algorithm and train the base classifier on them. Recall that Ensemble Learning methods require knowing the

training data size. Therefore, T is the training data size passed to the algorithm along with the training data. The model will be built and the final hypothesis will be the output for this stage. In this stage, there are two different cases to build the final model. The first case is when the learning method is Bagging. The Bootstrap replicates will be built and the base classifier is applied to each bootstrap to produce an ensemble of the hypothesis. Subsequently voting is used to predict the final model. The second case is using the Boosting learning method. In this case an equal weight is assigned to each instance. Next, D will be sampled with a replacement and the base classifier is trained on each subset to obtain the final model. However, the sampling distribution will be changed after each iteration, which is explained in detail in Section 2.3.1. At the end of this stage the final model is ready to be used as an input for the next stage.

Step four in this stage is about testing our inputs and deciding whether to continue to the next stage or to stop at this one. In order to continue to the next stage, we set two requirements. The first one is the size of the training dataset. Recall that the online Active Learning method we employ does not require knowledge of the data size. The other requirement is the base classifier type. The literature review of both C4.5 and Naïve Bayes suggest that they are not able to handle Data Streams well. Hence, if the base classifier is C4.5 or Naïve Bayes the process is stopped and the final model will be the output. As mentioned in Section 2.2.1, C4.5 requires running multiple scans on the entire data to calculate the split point of each branch. In Section 2.2.2, we discussed the Naïve Bayes feature of independence. As the literature suggests, this feature enables the classifier to learn incrementally, but it misleads the classifier from finding the correlation between the Data Stream's features. Therefore, we decided to choose these two requirements as our stopping criteria for the Ensemble Learning process. However, to validate this theory, we tested the datasets using these two classifiers in online mode. The result of this testing is presented in the next chapter and it will be shown that our results confirm this hypothesis.

In the Active Ensemble Learning stage, the model obtained from the Ensemble Learning stage is used as an input. As in Ensemble Learning stage, there are two learning methods used in this framework, namely OzaBag and OzaBoost. We decided to use these two

algorithms because they simulate the static data into a Data Stream. The inputs are the training data D_i from stage one, the first test data T_i, the base learner classifier C_i, and the Active Learning method A_i.

Using the resulted model from stage one, the test data T_i is tested against the model. The result is a prediction value for each unlabeled example in the test set. These prediction values are then presented to the user (human expert) to evaluate them and choose 10 examples from each class. The user will choose the examples with the highest prediction probability. He/she will proceed to label these examples and append them to the original training data. Adding these newly labeled examples will result in the new accumulated dataset F_i that will be tested repeatedly in the second stage. The resulted accumulated dataset F_i is then passed to the Active Learning method A_i.

If A_i is the OzaBag method, creating the Bootstrap will follow a Poisson distribution instead of the Binomial distribution that is used in the Bagging. Also, since we assume that the data size is unknown in this framework, the number of examples is set to infinite. Therefore, the Bootstrap is created by producing k copies of each newly arrived instance in the stream and updates each base model. Hence, if A_i is the OzaBoost method, a Poisson distribution is used to assign a random probability for each example in the training data. As mentioned in 2.3.1, Boosting requires knowing the training set size to calculate the weight for each example.

The steps from one to five, as shown in the pseudo-code, are then repeated until all the test subsets are tested. After each updates on the accumulated data set F_i by the user, the A_i is trained and a new model will result. The following diagram presents the learning process of this framework.

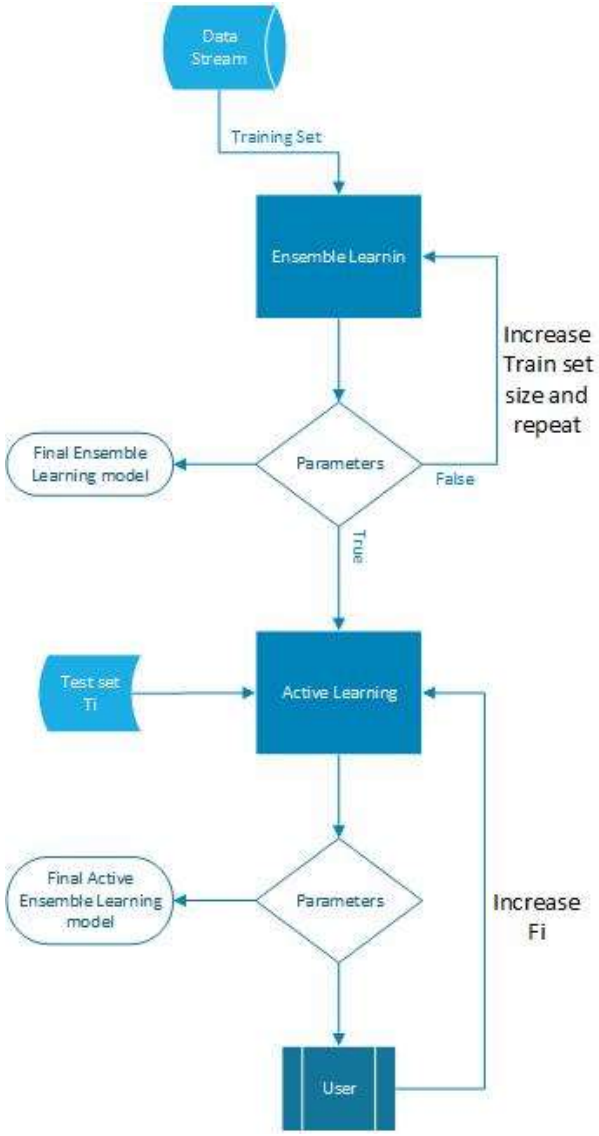

Figure 3: Active Ensemble Learning Process in Detail

4.1.1 Active Learning Using Ensemble Learning

The framework is divided into five main steps. In general, the framework starts by preprocessing the datasets to remove the unneeded attributes and normalize the datasets. The datasets are subsequently used in the Ensemble Learning steps where different methods are applied. After that, the inputs are tested to meet specific requirements to proceed to the next stage. The resulted model from the previous step is tested using different test sets and the result is then presented to the user. The data presented to the

user includes the prediction probability percentage of each unlabeled instance. Next, the user chooses the instances with the highest probability and labels and appends them to the original training set. Adding these newly labeled examples to the original training set by the user results in the accumulated dataset, which will be used in the Active Learning stage. After each iteration the accumulated dataset size increases with ten instances from each class that is in the dataset. Deciding to add ten instances from each class was based on two reasons. That is, we want the algorithm to learn from the smallest number of training examples as possible and this number was suggested in the literature (Muhivumundo & Viktor, 2011). Finally, the new training set produced by the user is used in the Active Learning process. This framework shows the process flows and algorithms used in our research. The used steps are explained in details in the following sections, and running examples are provided to clarify the process.

4.1.1.1 Step One: Preprocessing

Before using any datasets in a data mining Exercise they need to be preprocessed. In our setting the datasets are normalized using the Weka data mining environment (Hall et al., 2009). Weka is a data mining software developed in Java code by the University of Waikato. The software includes a large number of algorithms (Hall et al., 2009; Witten & Frank, 2005). As in any data mining exercise, an attribute selection is applied to our datasets. This is done due to the fact that attributes may contain redundant or irrelevant data. These kinds of attributes only slow the learning process and they provide no useful information (Piramuthu, 2004). Therefore, an attribute selection method is applied to focus only on the informative attributes. In this work, we set the attributes number to ten and this is done by experimentation. The benefit of this step is mentioned in Section 4.2.1.

The entire dataset is then divided into Test (90%) and Train (10%). As a next step, the test set is divided into 16 subsets of an equal size. The test sets are randomly selected from the original test set and divided into 16 subsets. By inspection, we decided to stop at 16 subsets because more dividing resulted in a small size of sets which may lead to overfitting. Also, the percentage of the split is chosen by experimentation, to allow us to choose between an 80:20 split or a 90:10 split as suggested by other researchers

("stackoverflow," ; Waikato). With a subsample by random selection of the dataset, several classifiers are trained and the performance compared. The performance in many cases is better when dataset is divided as 90:10; however, it remains the same in some others. For this reason, a decision is made to go with the 90:10 split.

Several equal subsets are created from the test set for later use. Each subset is randomly selected and contains the same percentage of the original data in the test set. The randomness is used to eliminate sample bias and to give each record in the dataset an equal chance to be selected for inclusion in the subsets. The details about training set size and test sets size are presented in Chapter 5. By the end of this step, the training set and the equal sized test sets will be ready for input into the next step.

The following Figure 4 shows a screenshot from the first step in this process. In this example, the training set is taken from the Spambase dataset, one of the datasets used in this study. The original size of this dataset is equal to 4,601 instances and it contains 57 attributes. After preprocessing and dividing this dataset, the result is a training set of size 230 and 10 attributes.

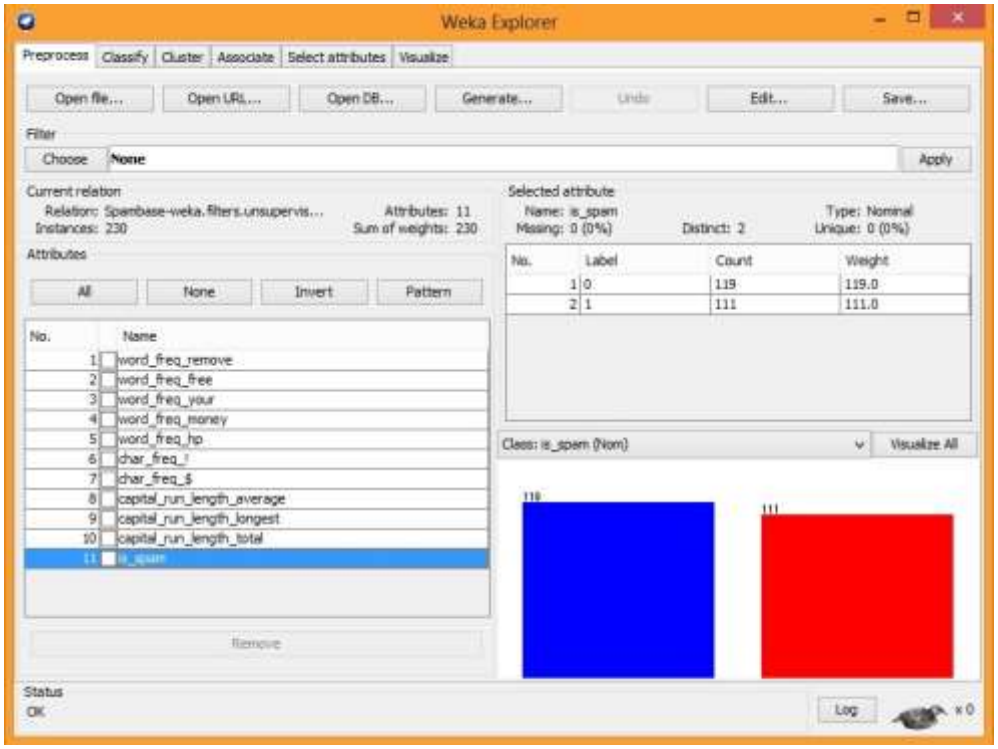

Figure 4: A Screenshot of the Running Example of Step One

4.1.1.2 Step Two: Training the Ensemble

The next part of the experiment starts with training the data using the Ensemble Learning methods. A training set contains a number of labeled data, which is used as an input to the Ensemble Learning methods. The method proceeds to create a set of hypotheses that generates the predictive model. In this step, different measure results are taken as the Ensemble Learning method results. They are later used to compare the algorithm's performance and evaluate the overall accuracy.

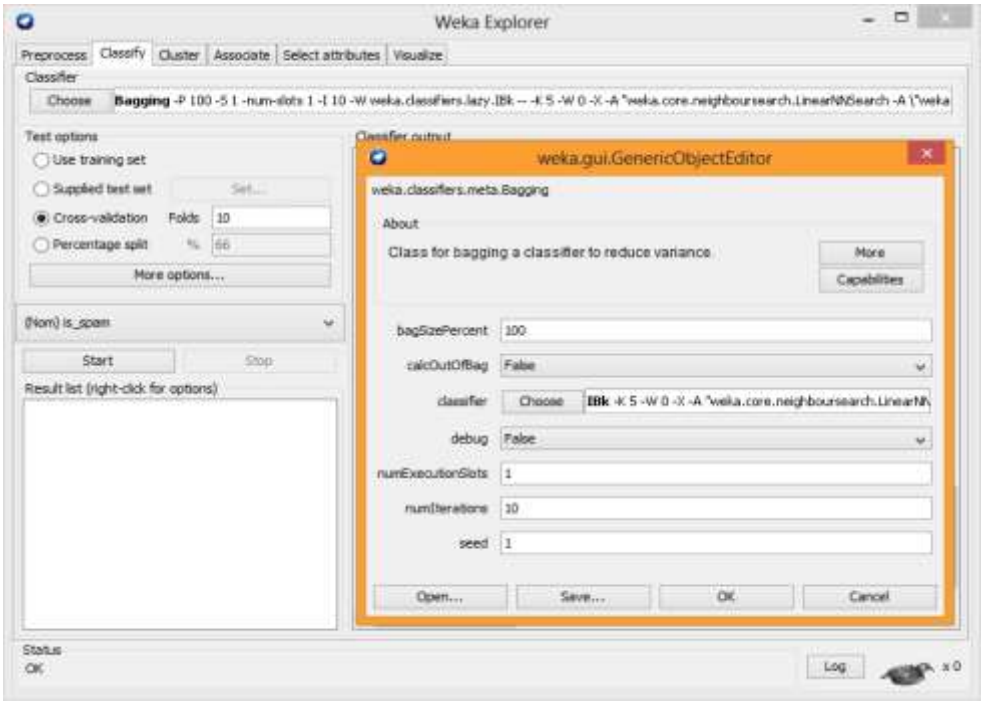

Figure 5: A Screenshot of the Running Example of Step Two

The previous Figure 5 shows how the Ensemble Learning method is trained using the training dataset. In this step, we show the example where the Bagging algorithm is trained using kNN as a base classifier. In this example, the ensemble size is set to 10 and the k value for kNN classifier is set to 5. As mentioned before, choosing the right k value is crucial; hence, 5 cross validation is the technique used to select the right value for this process. By the end of this step, the Ensemble Learning process will have finished, and the output will be the resulted model. After each learning process, we increase the size of the training set and train the same method using the same base classifier each time to evaluate the model accuracy. The entire process is repeated, each time with different

ensemble size. This is done in order to allow us to fine the optimal ensemble sizes for the four types of Ensemble used in this study. Table 3 shows an example of how the Bagging method is tested using different ensemble sizes with different base classifiers.

Table 3: Example of Changing the Ensemble Size

Dataset	Method	Classifier	Ensemble size
Waveform	Bagging	Naïve Bayes	10
			20
			25
			50
			100

Dataset	Method	Classifier	Ensemble size
Waveform	Bagging	Hoeffding Tree	10
			20
			25
			50
			100

Dataset	Method	Classifier	Ensemble size
Waveform	Bagging	C4.5	10
			20
			25
			50
			100

Dataset	Method	Classifier	Ensemble size
Waveform	Bagging	k-Nearest Neighbor	10
			20
			25
			50
			100

In this framework, "*10 fold cross-validation*" is used to test the resulted model. This process uses 100 labeled pieces of data to produce 10 equal-size sets. Each one of these sets is then divided into two groups: 90-labeled data for training and 10-labeled data for testing. The algorithm builds classifiers using the first group then applies the resulted classifier to the second group. The same process is then repeated for all the sets from two to ten. The final result is the average of the performance of the ten classifiers. Hence, in this step the final model is the Ensemble Learning model.

4.1.1.3 Step Three: Testing the Parameters

In this step of the framework, the inputs are tested and a decision will be made whether to continue to the Active Learning stage or to stop at the Ensemble Learning stage and output the final model. In this step there are two parameters. The first is the training set size. As mentioned in Chapter 3, the online learning methods do not require a previous knowledge of the training set size. Both OzaBag and OzaBoost, which are used in this Active Learning stage, set the size to infinite.

The second parameter is the base classifier type. Recall that in this framework we are using four different classifiers. These classifiers are tested individually with each one of

the four learning methods. Recall that C4.5 and Naïve Bayes have some limitation in the Data Stream setting was discussed in Chapter 2. However, as mentioned earlier, we seceded to also include these two classifiers in the Active Learning stage to validate our hypothesis. For example, Section 5.1.1.3 shows the result of the Active Learning stage. According to our framework, this section should include only the result of QBag using Hoeffding Tree and kNN as base classifiers. However, for consistency, we tested the framework using C4.5 and Naïve Bayes as well.

4.1.1.4 Step Four: Preparing Unlabeled Data for Labeling

The resulted model from the previous step is used to test against the first subset from the test set. The test set contains a number of unlabeled instances; therefore, the model gives the user a list of the predicted classes for each one with the probability of it.

4.1.1.5 Step Five: Active Learning Through User Labeling

This step is all about the user feedback. In the previous step, the algorithm provided the user with list of probabilities for all of the unlabeled instances from the test set. Next, the user will track the instances with the highest probability among the other unlabeled instances. Ten instances from each class with the highest probability are chosen, labeled, and appended to the original training set. Hence, the user labels these instances according to their prediction value. However, it is important to note that the prediction accuracy differs depending on the dataset and the method used. In some methods, the prediction percentage is as high as 100% for some instances.

The next Figure 6 shows the resulted prediction percentage for each unlabeled instance that is in the tested test set. In this case, the user will choose instances numbers 3 and 18. These two instances have the highest prediction value of 0.999% compared to the other instances from class ‘0.’ The same process is done for predicted instances in class ‘1.’ Instances numbers 8, 17, 20, and 25 are chosen since they have the highest prediction value compared to the other instances from the class ‘1’.

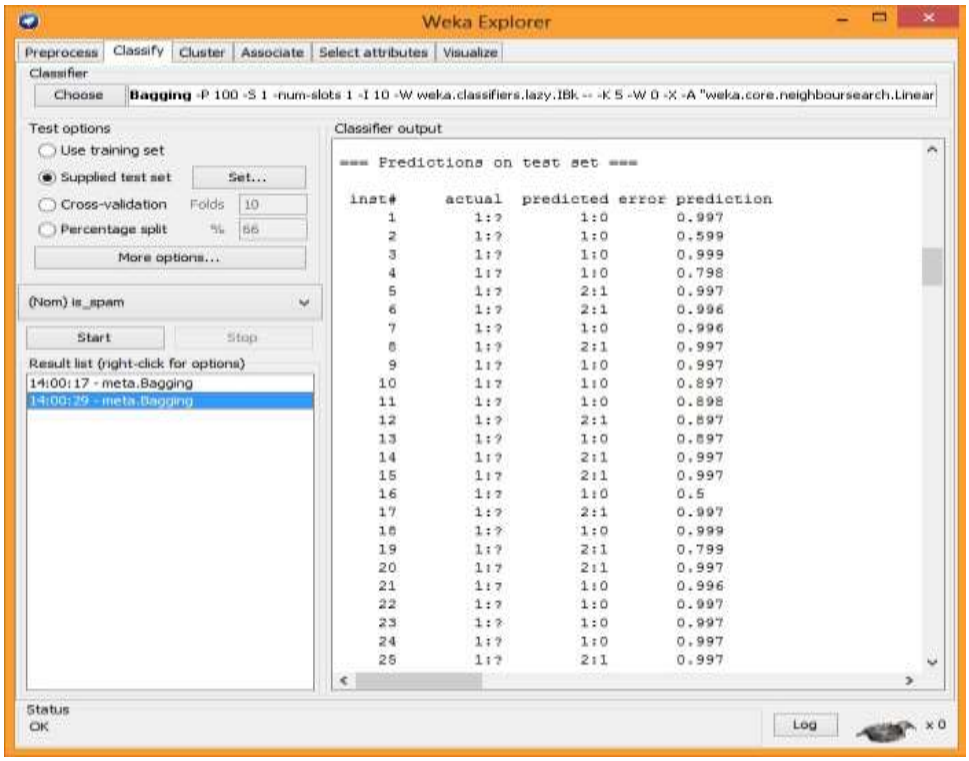

Figure 6: A Screenshot of the Running Example of Step Five

4.1.1.6 Step Six: Creating the Final Model

After updating the training set with the new-labeled instances by the user, an online learning method is applied on this accumulated training set to output the final model as shown in the next figure. The same process is repeated for all the steps until all the 16 test subsets are tested. The following Figure 7 shows the final model of the Active Ensemble Learning process.

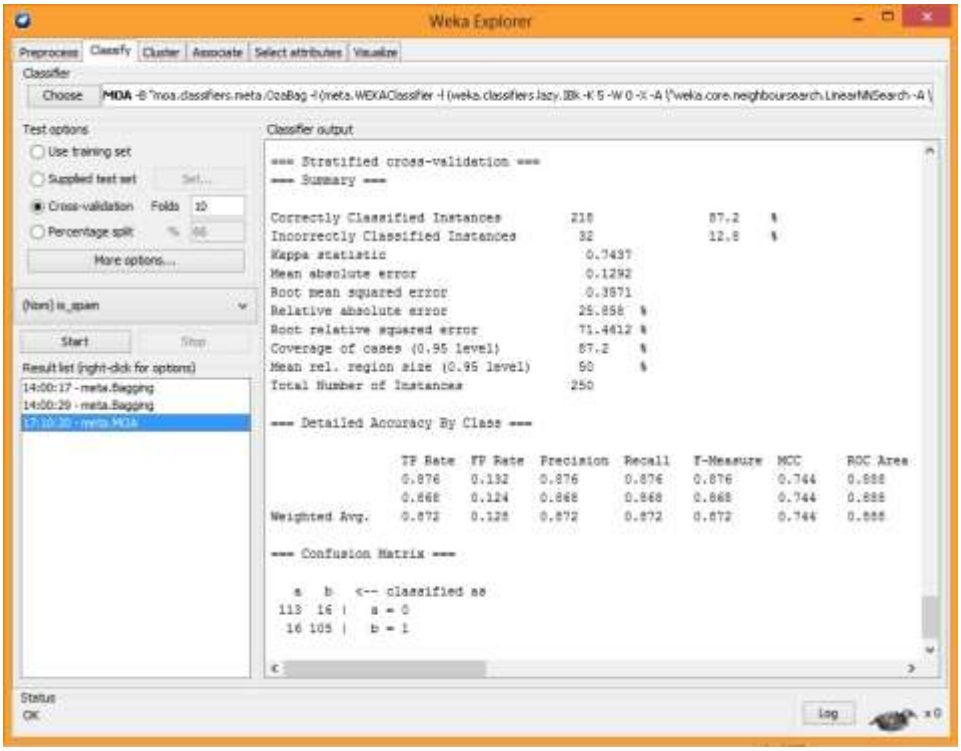

Figure 7: A Screenshot of the Running Example of Step Six

4.2 Environment Setup

This experiment took place using a machine equipped with an Intel(R) Core(TM) i5-2410M CPU @ 2.30 GHz processor and 8.00 GB of RAM. For this experiment, both the Weka and MOA data mining environment are used. MOA Massive Online Analysis is Data Stream Analytical software (Bifet et al., 2010). Recall that Weka includes a large number of machine learning algorithms, including the Bagging and Boosting algorithms. The Bagging algorithm is implemented under meta classifiers with the name Bagging, whereas Boosting algorithm is implemented under meta classifiers with the name AdaBoostM1 (Hall et al., 2009; Witten & Frank, 2005). To be precise, version 3.7.9 of Weka is used. As mentioned in Section 2.3.4, the Random Subspace Method and Stacking algorithms are not used in this experiment because of their computational complexity and memory limitation. However, they will both be considered in our future work.

The Online Bagging is implemented with the name OzaBag, and the Online Boosting is implemented under the name OzaBoost (Bifet et al., 2010; N. C. Oza & Russell, 2005).

The MOA framework was developed by Bifet et al. to serve a specific purpose, which is providing an environment for implementing and examining Data Stream algorithms (Bifet et al., 2010). The framework provides the user with the capability of generating Data Stream and to creating Data Stream from static data. However, MOA is added as an extension to Weka. In this way, both software classifiers can be used from Weka whenever needed (Bifet et al., 2010).

As mentioned in Section 2.2.3, when using kNN, choosing the right k value is very critical since it greatly influences the final model that it being built. For this reason, cross-validation is used to choose the right value. Cross-validation is a technique to select the right k number between a minimum and maximum value (IBM). In this framework, the range V is set between 1 to 5. As suggested by Chosh (Ghosh, 2006), the best max value for V for the large dataset is 5 or 10. The k values are calculated by computing the average error rate or sum-of square error of k:

$$CV_k = \sum_{v=1}^{v} e_v / V \qquad \textbf{(4. 1)}$$

where e_v is the error rate or sum of square error. Then optimal value of k is selected by using the following equation:

$$\hat{\boldsymbol{k}} = \boldsymbol{arg}\{\boldsymbol{minCV_k}: \boldsymbol{k_{min}} \leq \boldsymbol{k} \leq \boldsymbol{k_{max}}\} \qquad \textbf{(4.2)}$$

Moreover, it is important to be careful when determining the ensemble size, since this will also influence the final results. To this end, a similar procedure for each learning algorithm on the same dataset with different ensemble sizes was performed. This helps understand the effect of changing the size on the performance. In general, the process on the sizes of 10, 20, 25, 50 and 100 was executed. The sizes were determined by experimenting.

4.2.1 Data Preparation

After selecting the data sets, they must go under the preparation process since it follows that the quality of the data affects the quality of the mining process. Recall that, in our research, the data are first normalized then the relevant attributes are selected.

4.2.1.1 Normalization

The normalization process works by scaling all the numeric attributes to fit within small specific range [0,1]. In this thesis this process is done using the Weka unsupervised filter. There are several benefits of the normalization process. In this thesis, both numeric and categorical attribute characteristics are used. Therefore, by normalizing the datasets the numeric attributes are brought to the same scale as the categorical data [0,1]. Also, by normalizing the numeric attributes, the attributes with large scales are prevented from biasing the solution (Oracle, 2008).

4.2.1.2 Attribute Selection

Using Attribute selection (known also as feature selection), a reduced version of a dataset is produced. The aim of this process is to reduce the size of the dataset, which results in shortening the algorithm learning time and improving the model's accuracy (Bryll et al., 2003). Also, it removes the redundant or irrelevant attributes from the dataset. Redundant attributes are the ones that will add no more value to the learning process, whereas irrelevant attributes are the ones that are not useful for specific tasks. Although the user can do this process manually, it may cause confusion for the mining algorithm. Also, the domain expert does not know some dataset behavior. For this reason, it will be hard to decide which important attribute to keep and which to remove. Choosing to remove the wrong attribute may result in poor quality algorithm performance (Han et al., 2006). Adding to the previous benefits of using feature selection, the reduced data set will maintain a probability distribution of the classes as close to the original one as possible. However, this will not affect the performance quality.

The attribute selection method used in this experiment is Ranker with the Information Gain Attribute evaluator. This method is chosen by inspection since it treats each attribute individually so the attributes values do not influence each other. The Ranker

method tends to rank attributes according to their individual evaluation. The Information Gain Attribute Evaluator uses the Information Gain method to evaluate attributes according to their classes. Also, this method can handle missing value (Witten & Frank, 2005).

4.2.2 Datasets

In this experiment four datasets were used. These datasets had been chosen because they are benchmarking datasets frequently used in Data Stream and online learning research. The default task for the used data sets is classification. Moreover, these datasets were chosen from various sources with different types of attribute characteristics. Different ranges of dataset sizes in this experiment were used. The following table shows a summary of the original datasets:

Table 4: Summary of the Original Datasets

Name	Size	#Attributes	Classes	Data Distribution	Dataset Characteristic	Attribute characteristics	Missing value
Spambase	4601	57	2	Class 0 = 39.4% Class 1 = 60.5%	Multivariate	Integer, Real	Yes
Waveform	5000	40	3	Class 0 = 33.84% Class 1 = 33.06% Class 2 =33.1%	Multivariate, Data-Generator	Real	No
Chess	3196	36	2	Class 0 = 52% Class 1 = 48%	Multivariate	Categorical	No
Australian	690	14	2	Class 0 = 44.5% Class 1 = 55.5%	Multivariate	Categorical, Integer, Real	Yes

Spambase Dataset

The Spambase dataset includes classifying emails into Spam or Non-spam (UCI-Machine Learning Repository). This dataset includes a collection of e-mails from the postmaster

and individuals who had filed spam. Also, the collection of non-spam came from filed work and personal emails.

Specifically, the database contains information of 4,601 emails that are classified into spam or non-spam. Therefore, there are two classes: class 0 which is equal to 39.4% and class 1 which is equal to 60.5% of the entire dataset. In total, there are 57 attributes.

Waveform Dataset

The original dataset introduced in this paper includes missing values (Breiman, Friedman, Stone, & Olshen, 1984). However, the donor to the UCI repository updated the dataset and removed all the missing values. This dataset includes 5000 records that are divided between three classes. The first class, class 0 is equal to 33.84%, class 1 is equal to 33.06% and class 2 is equal to 33.1% of the total size of the dataset. There are 40 attributes in total.

Chess (King-Rock vs. King-Pawn) Dataset

The Chess dataset was originally generated and described by Alen Shapiro in 1989 (UCI-Machine Learning Repository). The overall size of this dataset is 3196 with 36 attributes. Records are divided into two classes: class 0 is equal to 52% and class 1 is equal to 48%. Also, there are no missing values in this dataset. Moreover, this dataset is considered to be a game area.

Australian (Credit Card Approval) Dataset

According to UCI, this dataset source is confidential (UCI-Machine Learnign Repository). However, it contains information about credit card applications. All the attributes' names and values are changed to protect the confidentiality of the data. This dataset is chosen because it has been used in two papers that are related to this thesis (J. Ross Quinlan, 1987; John Ross Quinlan, 1993). It includes 6,90 records divided into two classes: Class 0 includes 44.5% of the data and class 1 includes 55.5%.

As mentioned earlier, all of the datasets used in theses are divided into training and test sets. The following table shows a summary of the data set sizes of the test set, as well as the training set sizes at the beginning and the end of training.

Table 5: A Summary of Training and Test Subsets

	Spambase	Waveform	Chess	Australian
Number of attributes	10	10	10	10
Training set size	230	500	320	69
Each test subset size	273	281	180	38
Size of training set in first iteration	230	500	320	69
Size of training set in last iteration	550	980	640	389

4.3 Criteria of Evaluation

In order to be able to evaluate the performance of Active Learning and Ensemble Learning, the accuracy of the algorithms that are used within each process should be considered. Recall that in our research the Naïve Bayes, C4.5, k-Nearest Neighbor, and Hoeffding Tree algorithms are used. A number of performance measures that are generally used to evaluate Machine Learning methods were applied. The following is the performance measures and explanations of each one of them.

4.3.1 Performance Measures

Accuracy

To measure the accuracy of the model, the incorrectly classified instances percentage is used. In this case, the lower percentage means a better performance.

Mean Absolute Error

The Mean Absolute Error (MAE) uses each predicted value by the model and compares it to the actual value, and then it averages the absolute error to estimate the accuracy of the model. In other words, it shows how close the model prediction is to the actual value [3]. Therefore, the lower the percentage, the better the model. It is calculated by the following equation:

$$MAE = \frac{|\rho i - \alpha i| + \cdots + |\rho n - \alpha n|}{n}$$

(4.3)

F-Measure

This measure is a combination of the precision and recall measures. It can also be described as a weighted average of the recall and precision measures. The calculation equation is as follows:

$$F - Measure = \frac{Precision \times Recall}{Precision + Recall}$$

(4.4)

Precision

Precision is defined as the fraction of relevant instances that have been retrieved by the algorithm. In other words, a high percentage means that the algorithm returned more relevant instances than the irrelevant ones [5]. It is calculated using the following equation:

$$Precision = \frac{TP}{TP + FP}$$

(4.5)

Recall

This measure shows the percentage of the relevant results that have been returned by the algorithm. The higher the number, the better the performance [5]. The following is the calculating equation for this measure:

$$Recall = \frac{TP}{TP + FN}$$

(4.6)

4.4 Statistical Analysis

The statistical analysis is done using the Friedman test. This test is a non-parametric test that performs a comparative test on data without including mean, standard deviation, or any other typical parameters. It tests the difference between related samples that

populated using the same parameter but under different conditions (MedCalc). In this experiment, the datasets were tested using the same algorithms but with different ensemble size.

This test works by ranking the algorithms for each dataset. An average rank will be assigned to each algorithm starting with the best performance, which is assigned the rank value of 1. After ranking the algorithms, a decision to reject or accept the null hypothesis will be made. The null hypothesis states that there is no significant difference between the algorithms' performances (Theodorsson-Norheim, 1987). After calculating the Friedman test value, if the result is lower than 0.05, the critical level, the null hypothesis will be rejected, meaning that there is a difference between the tested variables.

4.5 Summary

This chapter presented the Active Ensemble Learning framework used in this study. The Active Ensemble Learning pseudo-code is presented in Section 1 followed by a discussion and examples to illustrate each step in the framework. Mainly the framework is divided into two main stages. To continue to the second stage, two parameters are tested which are the training set size and the classifier type.

Section 2 provides the reader with insight into the environment setup. This includes explanation about the preparation process of the data. The second half of this section presents the datasets that are used in this study. In Table 5, more details about each dataset are presented. Some of these details are the training sets' sizes before and after the learning process. This information is useful in the algorithms' evaluation, which is presented in the next chapter.

Sections 3 and 4 present the performance measures and the statistical analysis that are used to evaluate each model. As mentioned in this chapter, this framework is tested using four learning algorithms using four different classifiers. The resulted data will be presented in the next chapter along with their evaluation.

Chapter 5

Evaluation and Results

This chapter presents the experimental results when testing the Active Ensemble Learning framework, recall that our goal is to explore the suitability of combining Ensemble methods with the online Active Learning paradigm. Further, we study the effects of ensemble sizes and test set sizes on the overall accuracy and the time to build the model.

This chapter is organized as follows: The following Sections 5.1.1 to 5.1.4 will present the result for each method using different base classifiers. In each of the following subsections, the performance of each individual method when used with each classifier is discussed. After that, an overall evaluation will be presented for both Ensemble Learning and Active Ensemble Learning.

5.1 Experimental Results

This section presents the results against each one of the four datasets.

5.1.1 Spambase Dataset

5.1.1.1 Bagging Results

The Bagging method and the rest of the following methods are evaluated using four classifiers. As stated before, these classifiers are Naïve Bayes, C4.5, Hoeffding Tree, and k-Nearest neighbor. More details about these classifiers are given in Chapter 2.

Table 6 shows a summary of the results for the Bagging method on the Spambase dataset. Recall that cross-validation is applied to the process to ensure

the model accuracy. The best accuracy for the Naïve Bayes classifier is achieved by using the ensemble size of 20, whereas the best accuracy is at the size of 50 for the C4.5 and size 25 for k-nearest neighbor algorithm.

In the Hoeffding Tree classifier, three of the ensemble sizes resulted in the same error rate of 4%. However, the MAE percentage for ensemble size 50 is equal to 0.0426%, which is the lowest percentage among the other two results of 0.441% and 0.444%. Therefore, we can understand that the best accuracy for this classifier is achieved when building the model with the ensemble size of 50.

Table 6: Spambase Dataset- Bagging Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Spambase	4.5455	0.0459	0.9550	0.9550	0.9550
20		4.0000	0.0451	0.9600	0.9600	0.9600
25		4.1818	0.0447	0.9580	0.9580	0.9580
50		4.9091	0.0523	0.9510	0.9510	0.9510
100		5.6364	0.0572	0.9440	0.9440	0.9440
C4.5						
10	Spambase	4.1818	0.0663	0.9580	0.9580	0.9580
20		4.0000	0.0632	0.9600	0.9600	0.9600
25		3.8182	0.0602	0.9620	0.9620	0.9620
50		3.4545	0.0635	0.9650	0.9660	0.9650
100		4.3636	0.0833	0.9560	0.9570	0.9560
Hoeffding Tree						
10	Spambase	4.1818	0.0436	0.9580	0.9580	0.9580
20		3.8182	0.0433	0.9620	0.9620	0.9620
25		4.0000	0.0441	0.9600	0.9600	0.9600
50		4.0000	0.0426	0.9600	0.9600	0.9600
100		4.0000	0.0444	0.9600	0.9600	0.9600
k-Nearest neighbor						
10	Spambase	5.8182	0.0656	0.9420	0.9420	0.9420
20		8.1818	0.0927	0.9180	0.9180	0.9180
25		5.4545	0.0681	0.9450	0.9450	0.9450
50		5.6364	0.0628	0.9440	0.9440	0.9440
100		6.1818	0.0708	0.9380	0.9380	0.9380

In this method one can conclude that the C4.5 has the best performance among others with its ensemble size of 50. Moreover, the first model for this algorithm returned a percentage of 8.6957% of incorrectly classified instances. As the test continues with increasing the training set size, the accuracy is improved and the percentage drops down to 3.4545%, which indicates an improvement of an overall percentage of 5.2412%. Table 7 shows the result for Bagging with C4.5 classifier as a base learner and using the ensemble size of 50. Another thing to notice from this table is the classifier's ability to improve its accuracy using a small number of instances, which will be discussed at the end of Section 5.1.1.

Table 7: Spambase Dataset-The Results of Bagging Algorithm with C4.5 at Ensemble Size of 50

#instances	Incorrect	Mean abs errors	F-Measure	Precision	Recall
230	8.6957	0.1504	0.913	0.913	0.913
250	9.2000	0.1453	0.908	0.909	0.908
270	7.7778	0.1253	0.922	0.923	0.922
290	6.8966	0.1221	0.931	0.932	0.931
310	7.4194	0.1132	0.926	0.926	0.926
330	7.5758	0.1104	0.924	0.924	0.924
350	6.5714	0.1003	0.934	0.934	0.934
370	6.2162	0.0982	0.938	0.938	0.938
390	7.1795	0.0962	0.928	0.928	0.928
410	5.1220	0.0833	0.949	0.949	0.949
430	5.5814	0.0865	0.944	0.945	0.944
450	4.6667	0.0784	0.953	0.954	0.953
470	4.4681	0.0775	0.955	0.956	0.955
490	4.0816	0.0736	0.959	0.959	0.959
510	3.7255	0.0688	0.963	0.963	0.963
530	3.9623	0.0671	0.96	0.961	0.96
550	3.4545	0.0635	0.965	0.966	0.965

Table 8: Spambase Dataset-The Results of Boosting with C4.5 at Ensemble Size of 100

#instances	Incorrect	Mean abs errors	F-Measure	Precision	Recall
230	10.8696	0.1067	0.891	0.892	0.891
250	8.8000	0.0873	0.912	0.913	0.912
270	10.3704	0.1031	0.896	0.898	0.896
290	7.2414	0.0724	0.927	0.928	0.928
310	7.4194	0.0739	0.926	0.927	0.926
330	7.2727	0.0723	0.927	0.928	0.927
350	6.5714	0.0655	0.934	0.935	0.934
370	5.4054	0.0541	0.946	0.946	0.946
390	5.1282	0.0513	0.949	0.949	0.949
410	6.8293	0.0681	0.932	0.933	0.932
430	5.1163	0.0511	0.949	0.949	0.949
450	5.5556	0.0541	0.944	0.945	0.944
470	4.6809	0.047	0.953	0.954	0.953
490	4.0816	0.0408	0.959	0.96	0.959
510	3.3333	0.0334	0.967	0.967	0.967
530	3.9623	0.0393	0.96	0.96	0.96
550	2.7273	0.0269	0.973	0.973	0.973

5.1.1.2 Boosting Results

Table 9: Spambase Dataset - Boosting Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Spambase	4.9091	0.0698	0.9510	0.9510	0.9510
20		6.1818	0.0810	0.9380	0.9380	0.9380
25		4.7273	0.0683	0.9530	0.9530	0.9530
50		10.3636	0.1671	0.8960	0.8990	0.8960
100		5.4545	0.0759	0.9450	0.9460	0.9450
C4.5						
10	Spambase	3.8182	0.0461	0.9620	0.9620	0.9620
20		4.7273	0.0481	0.9530	0.9530	0.9530
25		3.2727	0.0325	0.9670	0.9670	0.9670
50		3.6364	0.0364	0.9640	0.9640	0.9640
100		2.7273	0.0269	0.9730	0.9730	0.9730
Hoeffding Tree						
10	Spambase	6.3636	0.0732	0.9360	0.9370	0.9360
20		6.3636	0.0946	0.9360	0.9360	0.9360
25		4.7273	0.0586	0.9530	0.9530	0.9530
50		5.4545	0.0655	0.9450	0.9460	0.9450
100		5.0909	0.0621	0.9490	0.9490	0.9490
k-Nearest neighbor						
10	Spambase	4.9091	0.0676	0.9510	0.9510	0.9510
20		4.9091	0.0591	0.9510	0.9510	0.9510
25		8.3636	0.0858	0.9160	0.9170	0.9160
50		4.9091	0.0559	0.9510	0.9510	0.9510
100		8.5455	0.0874	0.9150	0.9150	0.9150

Using the k-Nearest Neighbor algorithm as a base learner for Boosting, resulted in the same accuracy for the ensemble size of 10, 20 and 50. However, the MAE result is slightly different. Using ensemble size of 10, the MAE is equal to 0.0676%, whereas it is equal to 0.0591%, using ensemble size of 20; and it is equal to 0.0559% with the size of 50. As mentioned in Section 4.3.1, the lower the MAE, the better performance, making ensemble size of 50 is the best result for this algorithm.

However, on one hand, for the C4.5 algorithm, the best result achieved was after using the ensemble size of 100. On the other hand, both the Naïve Bayes and Hoeffding Tree algorithm returned the best result when using the ensemble size of 25.

In general, it can be said that the best accuracy for the Boosting algorithm is achieved using the C4.5 algorithm as a base learner. This algorithm experiment resulted in 10.8698% incorrectly classified instances at the first built model and ended up with only 2.7273%, which shows an improvement of 8.1425%. The result of Boosting algorithm using C4.5 as a base learner with the ensemble size of 100 are listed in Table 7.

5.1.1.3 Query by Bagging Results

In this method, the Naïve Bayes and C4.5 algorithm result in high percentages of incorrectly classified instances. Also, the MAE percentage is always equal to 0.5% in both algorithms even with different sizes of ensembles. The random result and the poor accuracy of the model show that the model is overfitting the data. See Chapter 2 for more information about the overfitting problem and what causes it in C4.5. Moreover, we mentioned in Chapter 2 that the Naïve Bayes feature of independency could mislead the classification process. In this case, we can understand that the classifier is not working. As mentioned before, the literature suggests that these two classifiers are not suitable for a Data Stream setting.

Table 10: Spambas Dataset-Query by Bagging Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Spambase	49.2727	0.5000	0.3410	0.2570	0.5070
20		49.2727	0.5000	0.3410	0.2570	0.5070
25		49.2727	0.5000	0.3410	0.2570	0.5070
50		49.2727	0.5000	0.3410	0.2570	0.5070
100		49.2727	0.5000	0.3410	0.2570	0.5070
C4.5						
10	Spambase	49.2727	0.5000	0.3410	0.2570	0.5070
20		49.2727	0.5000	0.3410	0.2570	0.5070
25		49.2727	0.5000	0.3410	0.2570	0.5070
50		49.2727	0.5000	0.3410	0.2570	0.5070
100		49.2727	0.5000	0.3410	0.2570	0.5070
Hoeffding Tree						
10	Spambase	3.8182	0.0459	0.9620	0.9620	0.9620
20		4.1818	0.0473	0.9580	0.9580	0.9580
25		4.3636	0.0451	0.9560	0.9570	0.9560
50		4.3636	0.0453	0.9560	0.9570	0.9560
100		4.1818	0.0460	0.9580	0.9590	0.9580
k-Nearest neighbor						
10	Spambase	6.0000	0.0604	0.9400	0.9400	0.9400
20		8.5455	0.0859	0.9150	0.9150	0.9150
25		6.1818	0.0623	0.9380	0.9380	0.9380
50		5.8182	0.0586	0.9420	0.9420	0.9420
100		7.4545	0.0750	0.9250	0.9260	0.9250

Since Naïve Bayes and C4.5 algorithms result in poor accuracy for this method, the evaluation is applied to Hoeffding Tree and k-Nearest Neighbor. In this method, the best accuracy is achieved when using Hoeffding Tree with ensemble size of 10. As Table 10 shows, there is an improvement in the accuracy with a percentage of 7.0514%, which means that the accuracy has dropped down from 10.8696% in the first model to 3.8182% in the last one. Table 12 shows the results of Boosting using C4.5 classifiers as base learners at the ensemble size of 100.

5.1.1.4 Query by Boosting Results

Like QBag, Naïve Bayes and C4.5 classifiers resulted in a poor accuracy model. Almost half of the dataset were incorrectly classified. Therefore, these two algorithms will not be taken into consideration when evaluating this method.

The k-Nearest Neighbor returned the best accuracy for the QBoost method. At the beginning of this experiment, the incorrectly classified instances percentage was equal to 11.7391%, but then decreased to 2.7273 % at the final model. Therefore, the improvement percentage for this model is equal to 9.0118%. Table 13 shows the result for QBoost using k-Nearest Neighbor as a base learner with the ensemble size of 50.

Table 11: Spambase Dataset-QBoost Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Spambase	49.2727	0.5000	0.3410	0.2570	0.5070
20		49.2727	0.5000	0.3410	0.2570	0.5070
25		49.2727	0.5000	0.3410	0.2570	0.5070
50		49.2727	0.5000	0.3410	0.2570	0.5070
100		49.2727	0.5000	0.3410	0.2570	0.5070
C4.5						
10	Spambase	49.2727	0.5000	0.3410	0.2570	0.5070
20		49.2727	0.5000	0.3410	0.2570	0.5070
25		49.2727	0.5000	0.3410	0.2570	0.5070
50		49.2727	0.5000	0.3410	0.2570	0.5070
100		49.2727	0.5000	0.3410	0.2570	0.5070
Hoeffding Tree						
10	Spambase	5.8182	0.2089	0.9420	0.9430	0.9420
20		6.3636	0.2255	0.9360	0.9370	0.9360
25		4.9091	0.2383	0.9510	0.9510	0.9510
50		5.4545	0.2132	0.9450	0.9460	0.9450
100		5.6364	0.2220	0.9440	0.9440	0.9440
k-Nearest neighbor						
10	Spambase	3.2727	0.0475	0.9530	0.9530	0.9530
20		5.6364	0.0574	0.9440	0.9440	0.9440
25		8.5455	0.0841	0.9150	0.9150	0.9150
50		2.7273	0.0463	0.9550	0.9550	0.9550
100		8.0000	0.0894	0.9110	0.9110	0.9110

Sections 5.1.1.1 to 5.1.1.4 presented the data of the four methods using different base classifiers applied to the Spambase dataset. Both Bagging and Boosting resulted in best accuracy with C4.5 as a base learner. However, Boosting resulted in better accuracy compared to Bagging in the Ensemble Learning process.

In the Active Learning stage, QBag and QBoost resulted in best accuracy with different base classifiers. For instance, the QBag's accuracy is higher using Hoeffding Tree, whereas the accuracy is higher when the kNN classifier is used as base learner for QBoost. Therefore, we conclude that QBoost has the best model for this stage.

When comparing between Ensemble Learning and the Active Learning accuracies, we find that using the Active Learning did not add much value to the process in this dataset. In both cases, the error rate is equal to 2.7273%, when using Boosting and QBoost. However, it is important to notice that this error rate resulted from different base classifiers. Adding to this, the learning time of the Active Learning process increased to 188.342 seconds, whereas in the Ensemble Learning process the learning time is 0.529 seconds.

This implies that Active Learning is resource intensive in terms of training time. However, on the other hand, the QBag algorithm produces an accurate model with an ensemble size of only 10 classifiers, when compared to the other approaches. The reader should also notice that we need a relatively small data set, of only 550 instances out of the original 4601 (11.9%), in order to build this accurate model.

Next, we turn our attention to the Waveform dataset.

Table 12: Spambase Dataset- The Results of Boosting with C4.5 at Ensemble Size of 100

#instances	Incorrect	Mean abs errors	F-Measure	Precision	Recall
230	10.8696	0.1179	0.891	0.892	0.891
250	9.6000	0.1072	0.904	0.905	0.904
270	8.8889	0.0934	0.911	0.912	0.911
290	7.5862	0.0842	0.924	0.924	0.924
310	7.7419	0.0848	0.922	0.923	0.922
330	6.6667	0.0735	0.933	0.933	0.933
350	6.5714	0.0774	0.934	0.934	0.934
370	6.2162	0.0762	0.938	0.938	0.938
390	5.6410	0.0619	0.943	0.944	0.943
410	5.6098	0.0613	0.944	0.944	0.944
430	5.3488	0.0587	0.947	0.947	0.947
450	4.8889	0.0511	0.951	0.951	0.951
470	4.6809	0.0502	0.953	0.953	0.953
490	4.2857	0.0487	0.957	0.957	0.957
510	3.9216	0.0433	0.961	0.961	0.961
530	3.7736	0.0450	0.962	0.963	0.962
550	3.8182	0.0459	0.962	0.962	0.962

Spambase Dataset- The Results of QBoost with k-Nearest Neighbor at Ensemble Size of 50

#instances	Incorrect	Mean abs errors	F-Measure	Precision	Recall
230	11.7391	0.1485	0.853	0.853	0.853
250	11.2000	0.1262	0.876	0.876	0.876
270	11.1111	0.1246	0.877	0.877	0.877
290	10.0000	0.1016	0.900	0.900	0.900
310	8.7097	0.0983	0.903	0.903	0.903
330	9.0909	0.0920	0.909	0.909	0.909
350	6.2857	0.0802	0.921	0.922	0.921
370	7.5676	0.0769	0.924	0.924	0.924
390	5.8974	0.0756	0.926	0.926	0.926
410	2.4390	0.0620	0.939	0.940	0.939
430	3.2558	0.0553	0.946	0.946	0.946
450	2.0000	0.1008	0.900	0.900	0.900
470	2.7660	0.0567	0.945	0.945	0.945
490	2.8571	0.0486	0.952	0.952	0.952
510	2.3529	0.0597	0.941	0.941	0.941
530	3.0189	0.0511	0.950	0.950	0.950
550	2.7273	0.0463	0.955	0.955	0.955

5.1.2 Waveform Dataset

5.1.2.1 Bagging Results

Similar to Spamebase dataset, the best model accuracy is achieved when using the C4.5 classifier. Using the ensemble size of 25, the error rate becomes 11.0204%. By looking at Table 14, it is shown that the improvement of the accuracy between the first model and the last one is equal to 10.3789%. However, the best accuracy for Naïve Bayes, Hoeffding Tree, and k-Nearest Neighbor classifiers is achieved at the ensemble sizes of 25, 100, and 10. Table 15 shows the results of the Bagging using C4.5 as base classifier with the ensemble size of 25.

Table 13: Waveform Dataset-Bagging Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Waveform	16.3265	0.1079	0.8330	0.8360	0.8370
20		16.2245	0.1098	0.8330	0.8380	0.8380
25		15.8163	0.1081	0.8380	0.8410	0.8420
50		15.9184	0.1076	0.8370	0.8400	0.8410
100		15.9184	0.1070	0.8370	0.8400	0.8410
C4.5						
10	Waveform	14.0816	0.1294	0.858	0.859	0.859
20		11.7347	0.1076	0.882	0.882	0.883
25		11.0204	0.1069	0.889	0.89	0.89
50		13.1633	0.1237	0.868	0.87	0.868
100		11.3265	0.1054	0.886	0.887	0.887
Hoeffding Tree						
10	Waveform	15.9184	0.1080	0.8370	0.8400	0.8410
20		16.0204	0.1074	0.8360	0.8390	0.8400
25		16.0204	0.1081	0.8360	0.8390	0.8400
50		15.8163	0.1073	0.8380	0.8410	0.8420
100		15.5102	0.1067	0.8410	0.8440	0.8440
k-Nearest neighbor						
10	Waveform	11.1224	0.0845	0.8880	0.8890	0.8890
20		12.0408	0.0851	0.8790	0.8800	0.8800
25		16.0204	0.1095	0.8400	0.8400	0.8400
50		11.2245	0.0792	0.8880	0.8870	0.8880
100		11.6327	0.0798	0.8830	0.8830	0.8840

Table 14: : Waveform Dataset- Result of Bagging with C4.5 as Base Classifier at Ensemble Size of 25

#instances	Incorrect	Mean abs errors	F-Measure	Precision	Recall
500	26.8000	0.1905	0.705	0.755	0.732
530	26.4151	0.1816	0.712	0.752	0.736
560	24.6429	0.1726	0.735	0.767	0.754
590	23.3898	0.1644	0.752	0.774	0.766
620	22.9032	0.1589	0.757	0.777	0.771
650	22.1538	0.1516	0.766	0.783	0.778
680	21.3235	0.1462	0.776	0.791	0.787
710	20.2817	0.1411	0.788	0.800	0.797
740	20.1351	0.1354	0.790	0.800	0.799
770	20.0000	0.1323	0.793	0.800	0.800
800	19.0000	0.1279	0.803	0.810	0.810
830	18.5542	0.1241	0.809	0.814	0.814
860	17.6744	0.1191	0.818	0.822	0.823
890	17.3034	0.1165	0.822	0.826	0.827
920	16.7391	0.1133	0.828	0.832	0.833
950	16.4211	0.1107	0.831	0.835	0.836
980	15.8163	0.1081	0.838	0.841	0.842

5.1.2.2 Boosting Results

The best model accuracy for Boosting is achieved using the ensemble size of 25 with k-Nearest Neighbor classifier as a base learner. The incorrectly classified instances are equal to 8.8776% with an improvement of 15.7224%.

On this dataset, as shown in Table 16, k-Nearest Neighbor is performing better compared to the previous dataset. Overall, the best accuracy of each classifier is achieved at different sizes of the ensemble. The best accuracy of Naïve Bayes is at the ensemble size of 100. C4.5 performs best at the size of 25. Lastly, Hoeffding Tree performance is best at the size of 25.

Table 15: Waveform Dataset-Boosting Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Waveform	16.2245	0.2094	0.8310	0.8450	0.8380
20		16.9388	0.2294	0.8230	0.8420	0.8310
25		17.7551	0.2358	0.8150	0.8310	0.8220
50		18.9796	0.2504	0.8020	0.8190	0.8100
100		16.1224	0.2114	0.8310	0.8470	0.8390

C4.5						
10	Waveform	14.4898	0.0939	0.8550	0.8550	0.8550
20		11.3265	0.0737	0.8860	0.8860	0.8870
25		12.7551	0.0837	0.8710	0.8730	0.8720
50		12.0408	0.0807	0.8790	0.8800	0.8800
100		11.5306	0.0770	0.8840	0.8850	0.8850
Hoeffding Tree						
10	Waveform	16.6327	0.2044	0.8280	0.8390	0.8340
20		16.6327	0.2016	0.8270	0.8420	0.8340
25		15.2041	0.1893	0.8420	0.8580	0.8480
50		18.4694	0.2444	0.8080	0.8310	0.8150
100		16.6327	0.1928	0.8270	0.8400	0.8340
k-Nearest neighbor						
10	Waveform	9.4898	0.0789	0.9050	0.9050	0.9050
20		11.2245	0.0787	0.8880	0.8880	0.8880
25		8.8776	0.0783	0.9110	0.9120	0.9110
50		10.4082	0.0765	0.8960	0.8960	0.8960
100		10.4082	0.0970	0.8960	0.8960	0.8960

5.1.2.3 Query by Bagging Results

Table 16: Waveform Dataset-QBag Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Waveform	65.5102	0.4444	0.1770	0.1190	0.3450
20		65.5102	0.4444	0.1770	0.1190	0.3450
25		65.5102	0.4444	0.1770	0.1190	0.3450
50		65.5102	0.4444	0.1770	0.1190	0.3450
100		65.5102	0.4444	0.1770	0.1190	0.3450
C4.5						
10	Waveform	65.5102	0.4444	0.1770	0.1190	0.3450
20		65.5102	0.4444	0.1770	0.1190	0.3450
25		65.5102	0.4444	0.1770	0.1190	0.3450
50		65.5102	0.4444	0.1770	0.1190	0.3450
100		65.5102	0.4444	0.1770	0.1190	0.3450
Hoeffding Tree						
10	Waveform	16.2245	0.1092	0.8350	0.8360	0.8380
20		16.0204	0.1090	0.8370	0.8380	0.8400
25		16.3265	0.1093	0.8340	0.8350	0.8370
50		16.3265	0.1093	0.8340	0.8350	0.8370
100		16.0204	0.1078	0.8370	0.8380	0.8400
k-Nearest neighbor						
10	Waveform	11.2245	0.0753	0.8870	0.8880	0.8880
20		12.5510	0.0841	0.8740	0.8740	0.8740
25		16.0204	0.1072	0.8400	0.8400	0.8400
50		11.6327	0.0780	0.8840	0.8830	0.8840
100		11.6327	0.0780	0.8830	0.8830	0.8840

Similar to Spambase dataset, Naïve Bayes and C4.5 resulted in a poor accuracy. More than half of the datasets were incorrectly classified. However, the best accuracy is achieved using k-Nearest Neighbor at ensemble size of 10. Also, it is fair to say that the classier performs well at the size of 50 and 100 since the difference between the incorrect percentages is less than 0.5%.

Hoeffding Tree accuracy was very close using all ensemble sizes. However, the size of 100 shows the best accuracy since it resulted in lower MAE rate. The following Table 17 shows a summary of the results.

5.1.2.4 Query by Boosting Results

Using C4.5 and Naïve Bayes as base learners resulted in classifying all the instances correctly. For these two classifiers, the resulted error is zero, which suggests that they are overfitting the data. Both k-Nearest Neighbor and Hoeffding Tree classifiers perform best using an ensemble size equal to 25. However, k-Nearest resulted in a lesser percentage of incorrectly classified instances.

Table 17: Waveform Dataset - QBoost Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Hoeffding Tree						
10	Waveform	16.6327	0.1514	0.8290	0.8360	0.8340
20		17.1429	0.1617	0.8230	0.8310	0.8290
25		15.9184	0.1604	0.8350	0.8460	0.8410
50		18.2653	0.1635	0.8110	0.8260	0.8170
100		16.7347	0.1566	0.8280	0.8330	0.8330
k-Nearest neighbor						
10	Waveform	12.3469	0.0830	0.8760	0.8760	0.8770
20		13.4694	0.0907	0.8650	0.8650	0.8650
25		11.7347	0.0790	0.8820	0.8820	0.8830
50		13.1633	0.0886	0.8680	0.8680	0.8680
100		15.1020	0.1014	0.8480	0.8490	0.8490

By considering the best model for each learning method, C4.5 resulted in the best model for only the Bagging learning method, after only 19.6% (980) instances from the full training set. The kNN classifier resulted in having the best accuracy when used with Boosting, QBag, and QBoost, as a base learner. In the Ensemble Learning process,

Boosting resulted in the best model as in the Spambase dataset. However, the Active Learning process results show that using Active Learning did not improve the accuracy. On the contrary, the model accuracy for both Bagging and Boosting decreased compared to the result of Ensemble Learning.

Moreover, using the Active Learning causes to increase the learning time. For example, in this dataset, both Boosting and QBoost resulted in the best accuracy by using kNN as a base classifier. However, the learning time in the Ensemble Learning is equal to 0.618 seconds, which increased to 478.902 seconds in the Active Learning. Again, QBag produces a slightly less accurate ensemble, using only 10 classifiers.

In the next section we evaluate the Chess dataset results.

5.1.3 Chess Dataset

5.1.3.1 Bagging Results

Table 18: Chess Dataset- Bagging Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Chess	9.8438	0.1076	0.9010	0.9020	0.9020
20		9.8438	0.1100	0.9010	0.9020	0.9020
25		10.0000	0.1069	0.9000	0.9010	0.9000
50		9.6875	0.1092	0.9030	0.9040	0.9030
100		10.4688	0.1154	0.8950	0.8960	0.8950
C4.5						
10	Chess	2.6563	0.0462	0.9730	0.9740	0.9730
20		2.6563	0.0455	0.9730	0.9740	0.9730
25		3.5938	0.0635	0.9640	0.9640	0.9640
50		2.6563	0.0456	0.9730	0.9740	0.9730
100		2.8125	0.0491	0.9720	0.9720	0.9720
Hoeffding Tree						
10	Chess	9.6875	0.1029	0.9030	0.9060	0.9030
20		6.0938	0.0913	0.9390	0.9400	0.9390
25		8.1250	0.0949	0.9180	0.9210	0.9190
50		10.7813	0.1042	0.8920	0.8940	0.8920
100		8.7500	0.0976	0.9120	0.9150	0.9130
k-Nearest neighbor						
10	Chess	3.4375	0.0524	0.9660	0.9660	0.9660
20		3.2813	0.0512	0.9670	0.9670	0.9670
25		3.4375	0.0517	0.9660	0.9660	0.9660
50		5.4688	0.0861	0.9450	0.9450	0.9450
100		5.4688	0.0750	0.9450	0.9450	0.9450

From the following table, we can notice the performance of C4.5 classifier. Using ensemble sizes 10, 20, 50, and 100 resulted in almost the same error rate. However, size 20 is considered to be the best accurate model since it resulted in lower MAE rate.

Both Hoeffding Tree and k-Nearest Neighbor perform better at ensemble size of 20. However, size 50 is the best for Naïve Bays. Table 21 shows the entire result of using C4.5 as a base learner with ensemble size of 20. It is fair to say that an ensemble size of 20 performs well for most of the classifier in this dataset when using Bagging method.

5.1.3.2 Boosting Results

Table 19: Chess Dataset- Boosting Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Chess	5.0000	0.0580	0.9500	0.9500	0.9500
20		6.2500	0.0638	0.9370	0.9380	0.9380
25		5.9375	0.0569	0.9410	0.9410	0.9410
50		4.0625	0.0551	0.9590	0.9600	0.9590
100		7.6563	0.0933	0.9230	0.9240	0.9230
C4.5						
10	Chess	4.5313	0.0611	0.9550	0.9550	0.9550
20		3.1250	0.0382	0.9690	0.9690	0.9690
25		2.5000	0.0376	0.9750	0.9750	0.9750
50		3.1250	0.0374	0.9690	0.9690	0.9690
100		2.6563	0.0379	0.9730	0.9730	0.9730
Hoeffding Tree						
10	Chess	3.4375	0.0461	0.9660	0.9660	0.9660
20		3.9063	0.0486	0.9610	0.9610	0.9610
25		3.1250	0.0439	0.9690	0.9690	0.9690
50		2.9688	0.0454	0.9700	0.9700	0.9700
100		3.7500	0.0485	0.9630	0.9630	0.9630
k-Nearest neighbor						
10	Chess	3.1250	0.0418	0.9690	0.9690	0.9690
20		2.9688	0.0420	0.9700	0.9700	0.9700
25		3.5938	0.0426	0.9640	0.9640	0.9640
50		2.8125	0.0391	0.9720	0.9720	0.9720
100		3.7500	0.0426	0.9620	0.9630	0.9630

Similar to Spambase, Boosting with C4.5 as a base learner resulted in best accuracy with incorrect classified instances equal to 2.5 but with a different ensemble size. It is interesting to notice that this dataset of C4.5, Hoeffding Tree, and k-Nearest Neighbor resulted in almost the same error rate with a fraction of less than 0.5.

Table 22 shows the improvement in C4.5 at size 25. The incorrect percentage dropped from 5.3125% to 2.5%. This shows an improvement of 2.8125. This small percentage shows the constancy in the performance of this classifier.

Table 20: Chess Dataset- Bagging with C4.5 at Ensemble Size of 20

#instances	Incorrect	Mean abs errors	F-Measure	Precision	Recall
320	5.3125	0.0911	0.947	0.948	0.947
340	5.0000	0.0872	0.950	0.951	0.950
360	4.7222	0.0817	0.953	0.953	0.953
380	4.4737	0.0753	0.955	0.956	0.955
400	4.2500	0.0725	0.957	0.958	0.958
420	4.0476	0.0699	0.959	0.960	0.960
440	3.8636	0.0662	0.961	0.962	0.961
460	3.6957	0.0641	0.963	0.963	0.963
480	3.5417	0.0617	0.965	0.965	0.965
500	3.4000	0.0588	0.966	0.966	0.966
520	3.2692	0.0556	0.967	0.968	0.967
540	2.9630	0.0537	0.970	0.971	0.970
560	3.0357	0.0523	0.970	0.970	0.970
580	2.9310	0.0500	0.971	0.971	0.971
600	2.8333	0.0493	0.972	0.972	0.972
620	2.7419	0.0460	0.973	0.973	0.973
640	2.6563	0.0455	0.973	0.974	0.973

Chess Dataset-Boosting with C4.5 at Ensemble Size of 25

#instances	Incorrect	Mean abs errors	F-Measure	Precision	Recall
320	5.3125	0.0765	0.947	0.947	0.947
340	5.8824	0.0728	0.941	0.941	0.941
360	5.0000	0.0692	0.950	0.950	0.950
380	4.7368	0.0652	0.953	0.953	0.953
400	4.2500	0.0636	0.957	0.957	0.958
420	4.7619	0.0582	0.952	0.952	0.952
440	3.8636	0.0546	0.961	0.961	0.961
460	3.4783	0.0515	0.965	0.965	0.965
480	3.5417	0.0506	0.965	0.965	0.965
500	4.2000	0.0495	0.958	0.958	0.958
520	3.4615	0.0474	0.965	0.965	0.965
540	3.1481	0.0443	0.969	0.969	0.969
560	3.2143	0.0442	0.968	0.968	0.968
580	3.4483	0.0434	0.966	0.966	0.966
600	3.1667	0.0397	0.968	0.969	0.968
620	2.9032	0.0398	0.971	0.971	0.971
640	2.5000	0.0376	0.975	0.975	0.975

5.1.3.3 Query by Bagging Results

As in Waveform and Chess datasets, the best classifier is k-Nearest Neighbor with an ensemble size of 10. The improvement in the accuracy of this classifier is equal to 3.75%. Hoeffding Tree performs well at size 20. However, C4.5 and Naïve Bayes were not able to correctly classify almost half of the dataset.

In this case, the Active Learning resulted in improving the model accuracy compared to the Ensemble Learning results. By considering the results in Table 19, we can see that using QBag with the kNN as the same base classifier resulted in better accuracy. However, in this study we evaluate the performance of the learning process regardless of the type of the base classifier.

Table 21: Chess Dataset-QBag Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Chess	47.5000	0.5000	0.3610	0.2760	0.5250
20		47.5000	0.5000	0.3610	0.2760	0.5250
25		47.5000	0.5000	0.3610	0.2760	0.5250
50		47.5000	0.5000	0.3610	0.2760	0.5250
100		47.5000	0.5000	0.3610	0.2760	0.5250
C4.5						
10	Chess	47.5000	0.5000	0.3610	0.2760	0.5250
20		47.5000	0.5000	0.3610	0.2760	0.5250
25		47.5000	0.5000	0.3610	0.2760	0.5250
50		47.5000	0.5000	0.3610	0.2760	0.5250
100		47.5000	0.5000	0.3610	0.2760	0.5250
Hoeffding Tree						
10	Chess	8.5938	0.1112	0.9140	0.9160	0.9140
20		7.1875	0.1027	0.9280	0.9280	0.9280
25		8.1250	0.1032	0.9190	0.9200	0.9190
50		9.3750	0.1032	0.9060	0.9080	0.9060
100		7.3438	0.0980	0.9260	0.9270	0.9270
k-Nearest neighbor						
10	Chess	2.8125	0.0438	0.9720	0.9720	0.9720
20		3.2813	0.0429	0.9670	0.9670	0.9670
25		2.9688	0.0443	0.9700	0.9700	0.9700
50		5.3125	0.0810	0.9470	0.9470	0.9470
100		5.4688	0.0694	0.9450	0.9450	0.9450

5.1.3.4 Query by Boosting Results

Similar to the previous datasets, QBoost performs well when using k-Nearest Neighbor as a base learner. However, in this dataset, the incorrectly classified instances are equal to 2.5 at an ensemble size of 10.

It is also possible to consider Hoeffding Tree another best classifier since the error percentage between them is a small fraction. As in all previous datasets, C4.5 and Naïve Bayes are performing poorly with misclassifying almost half of the dataset.

Table 22: Chess Dataset-QBoost Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Chess	47.5000	0.5000	0.3610	0.2760	0.5250
20		47.5000	0.5000	0.3610	0.2760	0.5250
25		47.5000	0.5000	0.3610	0.2760	0.5250
50		47.5000	0.5000	0.3610	0.2760	0.5250
100		47.5000	0.5000	0.3610	0.2760	0.5250
C4.5						
10	Chess	47.5000	0.5000	0.3610	0.2760	0.5250
20		47.5000	0.5000	0.3610	0.2760	0.5250
25		47.5000	0.5000	0.3610	0.2760	0.5250
50		47.5000	0.5000	0.3610	0.2760	0.5250
100		47.5000	0.5000	0.3610	0.2760	0.5250
Hoeffding Tree						
10	Chess	3.7500	0.2331	0.9620	0.9630	0.9630
20		5.3125	0.2806	0.9470	0.9480	0.9470
25		3.5938	0.2817	0.9640	0.9640	0.9640
50		4.6875	0.3164	0.9530	0.9530	0.9530
100		4.0625	0.3370	0.9590	0.9590	0.9590
k-Nearest neighbor						
10	Chess	2.5000	0.0762	0.9750	0.9750	0.9750
20		3.7500	0.0752	0.9630	0.9630	0.9630
25		3.9063	0.0781	0.9610	0.9610	0.9610
50		3.7500	0.0783	0.9630	0.9630	0.9630
100		4.3750	0.0776	0.9560	0.9560	0.9560

As a conclusion to these dataset results, we can say that Boosting resulted in better accuracy in the Ensemble Learning process and QBoost in the Active Learning process. However, there is no improving in the result. In both learning types, the error rate result is equal to 2.5%. In the Ensemble Learning process, C4.5 resulted in best accuracy.

However, unlike the previous datasets, the Hoeffding Tree model accuracy is best when used as base classifier for the QBoost process.

Also, an interesting result about this dataset is the running time. In this dataset, the running time for both Boosting and QBoost is the same and equal to 0.032 second. However, this learning time is for both C4.5 and Hoeffding Tree classifiers. In general, the Active Learning process neither added any accuracy to the final model nor increased the learning time. In this dataset, both QBag and QBoost need smaller ensembles, when compared to the two Ensemble methods, to build their models. The number on instances in the final training set is 640, which is 20.0% of the full data set.

Next, we evaluate the Australian dataset results.

5.1.4 Australian Dataset

5.1.4.1 Bagging Results

Table 23:Australian Dataset-Bagging Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Australian	5.9126	0.0561	0.9410	0.9410	0.9410
20		5.9126	0.0556	0.9410	0.9410	0.9410
25		5.3985	0.0540	0.9460	0.9460	0.9460
50		5.3985	0.0539	0.9460	0.9460	0.9460
100		5.6555	0.0561	0.9430	0.9430	0.9430
C4.5						
10	Australian	3.5990	0.0461	0.9640	0.9640	0.9640
20		4.1131	0.0613	0.9590	0.9590	0.9590
25		3.8560	0.0546	0.9610	0.9620	0.9610
50		3.8560	0.0465	0.9610	0.9620	0.9610
100		3.5990	0.0461	0.9640	0.9640	0.9640
Hoeffding Tree						
10	Australian	5.9126	0.0610	0.9410	0.9410	0.9410
20		5.9126	0.0574	0.9410	0.9410	0.9410
25		5.6555	0.0634	0.9430	0.9430	0.9430
50		5.9126	0.0565	0.9410	0.9410	0.9410
100		5.6555	0.0570	0.9430	0.9430	0.9430
k-Nearest neighbor						
10	Australian	3.3419	0.0443	0.9670	0.9670	0.9670
20		3.3419	0.0367	0.9670	0.9670	0.9670
25		3.5990	0.0400	0.9640	0.9640	0.9640
50		3.0848	0.0374	0.9690	0.9690	0.9690
100		3.0848	0.0377	0.9690	0.9690	0.9690

The result of this dataset is different compared to the previous ones. In this dataset, k-Nearest Neighbor is performing better than the other classifier at an ensemble size of 50. However, the accuracy of both C4.5 and k-Nearest Neighbor is almost the same which is ≈ 3%.

Therefore, if we decide to look at this dataset separately, we can consider kNN the best base learner for Bagging in this dataset. Whereas, if we take all the four datasets into consideration, we can say that C4.5 performs the best as base learner in this case.

Another thing to notice is the incorrect percentage in kNN. Both ensemble sizes of 50 and 100 resulted in the same percentage. However, size 50 is taken because it resulted in a better MAE result with a fraction of 0.0003%.

5.1.4.2 Boosting Results

Table 24: Australian Dataset-Boosting Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Australian	7.7121	0.0821	0.9230	0.9230	0.9230
20		5.9126	0.0583	0.9410	0.9410	0.9410
25		5.1414	0.0541	0.9490	0.9490	0.9490
50		4.6272	0.0505	0.9540	0.9540	0.9540
100		4.1131	0.0417	0.9590	0.9590	0.9590
C4.5						
10	Australian	2.3136	0.0244	0.9770	0.9770	0.9770
20		3.0848	0.0316	0.9690	0.9690	0.9690
25		2.0566	0.0206	0.9790	0.9790	0.9790
50		3.8560	0.0386	0.9610	0.9610	0.9610
100		2.8278	0.0263	0.9720	0.9720	0.9720
Hoeffding Tree						
10	Australian	4.3702	0.0496	0.9560	0.9560	0.9560
20		7.7121	0.0800	0.9230	0.9230	0.9230
25		6.1697	0.0690	0.9380	0.9380	0.9380
50		6.1697	0.0724	0.9380	0.9380	0.9380
100		4.8843	0.0511	0.9510	0.9510	0.9510
k-Nearest neighbor						
10	Australian	3.0848	0.0494	0.9690	0.9690	0.9690
20		3.3419	0.0480	0.9670	0.9670	0.9670
25		3.0848	0.0478	0.9690	0.9700	0.9690
50		3.0848	0.0473	0.9690	0.9690	0.9690
100		2.5707	0.0430	0.9740	0.9750	0.9740

In this case C4.5 classifier accuracy is the best at ensemble size of 25. The performance of C4.5 as a base learner with Boosting is also best in Spamebase and Chess datasets. However, kNN is also performing well with an error percentage slightly higher than C4.5. Both Naïve Bayes and Hoefdding Tree misclassified only at about 4%.

Using C4.5 as base learner resulted in a misclassification of 2.0566%. By looking at Table 27, the error rate decreased by 0.3125%. As said before, this means that the classifier performed well from the beginning of the experiment to the end.

Table 25: Australian Dataset-Boosting with C4.5 at Ensemble Size of 25

#instances	Incorrect	Mean abs errors	F-Measure	Precision	Recall
320	6.2500	0.0937	0.937	0.938	0.938
340	7.0588	0.0904	0.929	0.929	0.929
360	7.2222	0.0914	0.928	0.928	0.928
380	7.1053	0.0940	0.929	0.929	0.929
400	7.0000	0.0905	0.930	0.930	0.930
420	8.0952	0.0881	0.919	0.919	0.919
440	7.2727	0.0793	0.927	0.928	0.927
460	6.5217	0.0787	0.935	0.935	0.935
480	6.0417	0.0751	0.940	0.940	0.940
500	7.0000	0.0768	0.930	0.930	0.930
520	5.9615	0.0714	0.940	0.940	0.940
540	7.0370	0.0709	0.930	0.930	0.930
560	5.5357	0.0668	0.945	0.945	0.945
580	4.6552	0.0596	0.953	0.953	0.953
600	5.8333	0.0639	0.942	0.942	0.942
620	5.4839	0.0615	0.945	0.945	0.945
640	5.9375	0.0569	0.941	0.941	0.941

Table 26: Australian Dataset-Bagging with kNN with Ensemble Size of 100

#instances	Incorrect	Mean abs errors	F-Measure	Precision	Recall
320	6.5625	0.0900	0.934	0.934	0.934
340	5.8824	0.0849	0.941	0.941	0.941
360	5.5556	0.0824	0.944	0.944	0.944
380	5.0000	0.0778	0.950	0.950	0.950
400	4.7500	0.0715	0.953	0.953	0.953
420	4.7619	0.0689	0.952	0.952	0.952
440	5.0000	0.0694	0.950	0.950	0.950
460	5.2174	0.0662	0.948	0.948	0.948
480	4.3750	0.0618	0.956	0.956	0.956
500	4.4000	0.0598	0.956	0.956	0.956
520	3.8462	0.0525	0.962	0.962	0.962
540	5.5556	0.0799	0.944	0.944	0.944
560	5.3571	0.0783	0.946	0.946	0.946
580	5.6897	0.0802	0.943	0.943	0.943
600	4.8333	0.0728	0.952	0.952	0.952
620	5.1613	0.0759	0.948	0.948	0.948
640	5.4688	0.0694	0.945	0.945	0.945

5.1.4.3 Query by Bagging Results

As in all the previous datasets, C4.5 and Naïve Bayes accuracy is poor since it correctly classifies less than half of the dataset. However, kNN accuracy is the best with a misclassified rate equal to 3.0848%. The kNN model accuracy is almost constant with all ensemble sizes. Table 28 shows the result of using kNN and a base learner with an ensemble size of 100.

Table 27: Australian Dataset-QBag Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Naïve Bayes						
10	Australian	51.9280	0.5000	0.3120	0.2310	0.4810
20		51.9280	0.5000	0.3120	0.2310	0.4810
25		51.9280	0.5000	0.3120	0.2310	0.4810
50		51.9280	0.5000	0.3120	0.2310	0.4810
100		51.9280	0.5000	0.3120	0.2310	0.4810
C4.5						
10	Australian	51.9280	0.5000	0.3120	0.2310	0.4810
20		51.9280	0.5000	0.3120	0.2310	0.4810
25		51.9280	0.5000	0.3120	0.2310	0.4810
50		51.9280	0.5000	0.3120	0.2310	0.4810
100		51.9280	0.5000	0.3120	0.2310	0.4810

Hoeffding Tree						
10	Australian	5.9126	0.0659	0.9410	0.9410	0.9410
20		5.9126	0.0633	0.9410	0.9410	0.9410
25		5.9126	0.0638	0.9410	0.9410	0.9410
50		5.9126	0.0614	0.9410	0.9410	0.9410
100		5.9126	0.0601	0.9410	0.9410	0.9410
k-Nearest neighbor						
10	Australian	3.3419	0.0345	0.9670	0.9670	0.9670
20		3.3419	0.0344	0.9670	0.9670	0.9670
25		3.8560	0.0396	0.9610	0.9620	0.9610
50		3.5990	0.0370	0.9640	0.9640	0.9640
100		3.0848	0.0319	0.9690	0.9690	0.9690

5.1.4.4 Query by Boosting Results

By comparing the results of all the previous datasets, we can say that kNN works best for the QBoost method. In this dataset, the best ensemble size is at 20, which resulted in a misclassification of only 3.856% of the whole dataset.

Also, similar to a Waveform dataset, Naïve Bayes and C4.5 are both overfitting the data. They both resulted in a perfect model with no misclassification at all. More discussion about the performance of all the classifiers will be given in the next section.

Table 28: Australian Dataset-QBoost Results

Ensemble size	Dataset	Incorrect	Mean abs errors	F-Measure	Precision	Recall
Hoeffding Tree						
10	Australian	7.1979	0.2022	0.9280	0.9280	0.9280
20		7.4550	0.2494	0.9250	0.9260	0.9250
25		8.4833	0.2638	0.9150	0.9150	0.9150
50		8.7404	0.2965	0.9120	0.9120	0.9120
100		7.7121	0.2666	0.9230	0.9230	0.9230
k-Nearest neighbor						
10	Australian	4.6272	0.0530	0.9490	0.9490	0.9490
20		3.8560	0.0497	0.9520	0.9520	0.9520
25		5.6555	0.0581	0.9430	0.9440	0.9430
50		4.8843	0.0503	0.9510	0.9510	0.9510
100		4.1131	0.0528	0.9490	0.9490	0.9490

In the Ensemble Learning evaluation of this dataset, we find that kNN resulted in the best model accuracy for the Bagging learning method and C4.5 for the Boosting method. However, the Active Learning results show that QBag resulted in a higher accuracy compared to QBoost.

In general, the Active Learning did not improve the accuracy in this dataset. The error rate remains the same using Bagging and QBag but the running time increased from 0.017 seconds to 0.044 seconds, against 14.5% of the full data set. In this case, QBoost uses the smallest ensemble.

The next section presents the statistical significance testing of the results.

5.2 Friedman Test - Statistical Evaluation

As mentioned before, this test ranked the algorithms based on the average mean of the data and proceeded to calculate the Friedman test result. As a result, if this result is higher than 0.05, it is not possible to reject the null hypothesis. This means that in this dataset, there are no significant differences among the used algorithms. The following are the results of applying the Friedman test on all of the resulted data:

Table 29: Friedman Test Results of Bagging

	Spambase	Waveform	Chess	Australian
Naïve Bayes	2.9	3.6	3.8	3.3
C4.5	1.6	1.4	1.2	2
Hoeffding Tree	1.5	3.1	3.2	3.7
k-Nearest Neighbor	4	1.9	1.8	1
Asypm, Sig.	0.004	0.022	0.004	0.002

In the Bagging algorithm, the results show that there is a significant difference in the performance of each algorithm. Also, it shows that C4.5 ranked the best in three data sets which validates the results mentioned in the previous section.

Table 30: Friedman Test Results of Boosting

	Spambase	Waveform	Chess	Australian
Naïve Bayes	3	3.6	4	3.2
C4.5	1	2	2	1.4
Hoeffding Tree	3.1	3.4	2.3	3.8
k-Nearest Neighbor	2.9	1	1.7	1.6
Asypm, Sig.	0.24	0.004	0.021	0.006

The result of Boosting algorithm in Spambase datasets indicates that there is no difference between the different algorithms used. However, in the ranking comparison, C4.5 resulted in best performance for two of the datasets. Also, it ranked as the second best in the other two datasets.

Table 31: Friedman Test Results of QBag

	Spambase	Waveform	Chess	Australian
Naïve Bayes	3.5	3.5	3.5	3.5
C4.5	3.5	3.5	3.5	3.5
Hoeffding Tree	1	2	2	2
k-Nearest Neighbor	2	1	1	1
Asypm, Sig.	0.002	0.002	0.002	0.002

The Query by Bagging results show that there is a significant difference between the four algorithms. However, kNN ranked as the best performance for the three datasets.

Table 32: Friedman Test Results of QBoost

	Spambase	Waveform	Chess	Australian
Naïve Bayes	3.5	-	3.5	-
C4.5	3.5	-	3.5	-
Hoeffding Tree	1.6	2	1.6	2
k-Nearest Neighbor	1.4	1	1.4	1
Asypm, Sig.	0.004	0.025	0.004	0.025

As in the QBag results, QBoost results show a significant difference between the classifiers. Also, in this learning method the kNN ranked the best for the three classifiers.

5.3 Time Evaluation

The averaged time presented in this section is measured by seconds. For each dataset, the sum of the building time for each model at each iteration was averaged. Next, for each algorithm, the sum of each data set was averaged. The following is the result for each algorithm. From tables 35 to 38, one infers that using kNN with QBoost is time consuming. The averaged time increases as the dataset size increases. The other algorithms are relatively the same. Recall that the kNN classifier is time consuming

because of the way it constructs it model. In this classifier, the distance between new instances to all the labeled sets is calculated and added to the distance matrix after each arrivals of a new instance. Also, in every calculation, the algorithm needs to scan the entire already stored data to finish the calculation. However, the Hoeffding Tree time is comparable in the all learning methods. As mentioned in Chapter 2, this classifier learns incrementally. It trains each instance as it arrives in the stream without the need to access previous data.

Table 33: Bagging Averaged Classification Time

Ensemble Size	HT	kNN	C4.5	NB
10	0.043	0.009	0.060	0.011
20	0.063	0.019	0.097	0.015
25	0.067	0.016	0.122	0.020
50	0.116	0.028	0.218	0.029
100	0.228	0.033	0.423	0.054

Table 34: Boosting Averaged Classification Time

Ensemble Size	HT	kNN	C4.5	NB
10	0.048	0.269	0.080	0.034
20	0.048	0.429	0.180	0.030
25	0.054	0.580	0.209	0.034
50	0.052	0.955	0.410	0.030
100	0.070	2.302	0.799	0.029

Table 35: QBag Averaged Classification Time

Ensemble Size	HT	kNN	C4.5	NB
10	0.029	0.017	0.008	0.008
20	0.047	0.021	0.013	0.010
25	0.053	0.021	0.013	0.011
50	0.099	0.033	0.018	0.032
100	0.193	0.066	0.026	0.043

Table 36: QBoost Averaged Classification Time

Ensemble Size	HT	kNN	C4.5	NB
10	0.041	84.605	0.005	0.004
20	0.060	160.309	0.007	0.005
25	0.071	177.303	0.006	0.005
50	0.138	335.459	0.010	0.010
100	0.281	588.104	0.017	0.014

5.4 Discussion and Lessons Learned

The following Table 39 shows a summary of the best accuracy result for each learning method. Next, we discuss the results and lessons learned.

Table 37: A Summary of the Datasets Results (Highest Accuracy)

	Ensemble Learning		Active Ensemble Learning		Comments
	Bagging	Boosting	QBag	QBoost	
Spambase	C4.5 3.4545	C4.5 2.7273	HT 3.8182	kNN 2.7273	- Error rate increases or remain the same. - A large increase in the running time especially with the kNN classifier.
Waveform	C4.5 11.0204	kNN 8.8776	kNN 11.2245	kNN 11.7347	
Chess	C4.5 2.6563	C4.5 2.5000	kNN 2.8125	HT 2.5000	
Australian	kNN 3.0848	C4.5 2.0566	kNN 3.0848	kNN 3.8560	

5.4.1 Ensemble Learning Evaluation and Discussion

The main goal of Ensemble Learning is to improve the model performance by combining multiple classifiers. In this study Bagging and Boosting learning methods were used with four different classifiers as base learners to improve the prediction accuracy. The results of this experiment are presented in Section 5.2.1 to Section 5.2.4. Therefore, in this section an overall evaluation of the two ensemble methods are presented and the result of each one using different base classifiers are compared.

In general, Bagging works by creating random samples which are selected uniformly with replacements. As mentioned in Section 2.3.2, this will give each example an equal chance to be selected. Also, in Bagging, each model will be built separately by using voting to combine the results to produce the final model.

By considering the best classifier accuracy for each method, unlike Boosting, the Bagging performance is better in the presence of noise. By comparing the results of Bagging and Boosting, it can be concluded that the incorrectly classified instances percentage is always better in Bagging, which might be because of the presence of noise.

In evaluation of the result of the Bagging method, the C4.5 performance is best in three datasets with the exception being the Australian dataset. In this dataset, the best performance is achieved using kNN. However, kNN is better than C4.5 by only ≈ 0.5. Also, this dataset is the only small dataset used in this experiment. By taking these facts into account, our experimental results suggest that C4.5 is the best classifier, in terms of accuracy, to be used as a base learner for Bagging when considering a larger dataset.

However, in practice this classifier needed more time to produce the final model compared to the other ones. Recall that the time evaluation was presented in Section 5.3. In our opinion, this time is needed due to the fact that C4.5 requires running multiple scans to partition the data then updating the final tree with each new arrival instance. Moreover, as mentioned in Section 2.3.2, the Bagging method itself requires time to create the bootstraps to pass them to the classifier. To verify our conclusion, we used the Friedman test, which shows that the result of the used classifiers are significantly different and C4.5 model accuracy is also significantly different than the other classifiers. Recall that the result of this test was presented in Section 5.2.

The main idea of Boosting is to improve the performance of the previous model by reducing the weight error of each instance. However, as mentioned in Section 2.3.1, this method is very sensitive to outliers. As in Bagging, C4.5 performed better in three datasets, whereas kNN performed best in the Waveform dataset. In this method, kNN outperformed C4.5 by $\approx 2.45\%$, which is considered close to the C4.5 performance. Like in Bagging, this result is verified using the Friedman test. As mentioned in Section 2.2,

C4.5 is a batch learner, which trains the classifier on batches. In this experiment, a number of instances are added to the training set after each iteration then the set is passed to the algorithm, which is very similar to C4.5 process.

To conclude, in the Ensemble Learning process, C4.5 outperforms the other classifiers in six out of eight cases. For this reason, we conclude that both Bagging and Boosting perform best using C4.5. In a comparison between the two learning methods, we find that Boosting always resulted in higher model accuracy. Finally, in the Ensemble Learning process we conclude that the best ensemble size for both learning methods is equal to 25 since it resulted in the best model accuracy in most of the cases.

5.4.2 Active Ensemble Learning Evaluation and Discussion

The main idea of Active Learning is to use a diversity of Ensemble Learning to create a number of models. In this learning method, a committee is created, which contains a voted result of training different models on the current labeled data. In this study when using QBag the Bagging ensemble algorithm was used to create the models using four different base learners presented in Section 2.2. Also, the Boosting ensemble algorithm is used to create the models when using QBoost methods.

Another important criterion about Active Learning is the number of labeled data (incremental training set) that is used to train the classifier. As Table 5 in Section 4.2.2 presents the size of the training sets used in each iteration, the small size of training sets used to train the classifiers is clear. As mentioned before, an accurate Data Stream mining algorithm needs to adapt and classify new instances using small amounts of training data.

In this framework, an online version of QBag and QBoost are used. The online algorithm is used because it has the ability to process each single example as it arrives in the stream. Before starting to evaluate the result of QBag and QBoost, it is good to notice the performance of the algorithm when both Naïve Bayes and C4.5 are used as a base learner. In these two cases, the algorithm accuracy is either very bad or results in a perfect model that is able to classify all the records that suggest having the problem of overfitting. For this experiment, it seems that the methods do not do better than random guessing. The reason of this performance comes from the way these two classifiers work.

In the case of a C4.5 classifier, in order for this classifier to work it needs to have the entire data set stored to calculate the split point for each branch. Also, it requires running multiple scans on the entire dataset. Adding to this, overfitting causes the algorithm to result in a model that has a zero error percentage, which is not realistic in practice. As mentioned in Section 2.2, overfitting can happen because of the empty branches or noisy data.

In the case of Naïve Bayes, although this classifier assumes the independency of each feature, this assumption can mislead the learning process. In this study it is assumed that because of the law of number of features, the classifier did not have enough information to be able to find the correlation between these features.

For both QBag and QBoost, surprisingly, the kNN classifier performance is always best except for two sets of ensemble sizes (2 out of 20 experiments) where the Hoeffding Tree outperformed it. By this result, it can be conclude that Active Ensemble Learning performs best when using kNN as a base classifier. This happens because of the nature of kNN. As mentioned in Section 2.3, it is a lazy learner that does not process the instances until the arrival of new unlabeled one. However, observing new instance requires only updating the distance database. Therefore, the increase of the averaged classification time as the ensemble size increases, as in Section 5.3, is noticeable. As in the Ensemble Learning evaluation, by using the Friedman test, as presented in Section 5.2, the result is validated.

The following observation is noteworthy. Hoeffding trees have become the benchmark algorithm against which other Data Stream algorithms are evaluated. Our results suggest that this trend may not be for the best. Rather, as our results have shown, the simple kNN algorithm outperformed Hoeffding trees against our benchmarking datasets.

To conclude our results for the Active Ensemble Learning process, the kNN results in having the best model accuracy when used with both QBag and QBoost. Also, in most cases the algorithms are able to build an accurate model, using a small number of training sets and a small ensemble size.

In this study we conclude that the best ensemble size is equal to 10 for the Active Learning process. Thus, Active Learning leads to smaller, more compact ensemble sizes than traditional Ensemble Learning. This is an important result (e.g. in a cost-sensitive learning setting where the model size is of importance). It is also important to stress that a benefit of our approach is that the training size is small. That is, we are able to build accurate models against very small, incrementally growing training sets. This is an important advantage, especially in a big data setting. However, on the other hand, using the Active Learning methods brings no improvement in accuracy. The error rates either stay the same or increase in some cases. Also, using the Active Learning methods causes an increase to the running time in most cases. It follows that research is needed to find a balance between these requirements.

Moreover, the presented framework provides valuable **guidelines** to data mining practitioners who aim to determine when to use the Active Learning process. Considering the evaluated result and the datasets used in this work, we conclude that if the ensemble size is important to the work under consideration, then the Active Learning is a good choice. That is, in a cost-sensitive learning setting where the size of the models are of importance, then going the Active Ensemble route may be worthwhile. However, if the main goal of learning is to provide highly accurate models, then the Active Learning does not bring much improvement against the datasets we used. The reason should notice that the datasets we used are relatively balanced and does not contain a high level of noise. We are thus interested in determining whether this result will also hold in an imbalanced data setting and will explore this in our future research. Active Learning is time consuming because of the user-in-the-loop. For this reason, if the time is important in the work we might not want to consider following an Active Learning process. However, on the other hand, having the user-in-the-loop is very helpful especially in the case of datasets where we do not have all the data labels or where the labels arrive late. Again, we aim explore this issue in our future work.

5.5 Conclusion

This chapter presented our experimental results against four benchmarking datasets. In our experiments, we studied the interplay between ensemble sizes, base classifiers and the Ensemble Learning paradigm.

To conclude, using C4.5 as a base learner resulted in the best model accuracy for Ensemble Learning. In the case of Active Ensemble Learning, the kNN classifier is the best base learner in terms of accuracy. Also, the ensemble size that resulted in best model accuracy, by majority, is size 25 for the Ensemble Learning and 10 for the Active Learning.

Overall, by considering the results of both Ensemble and Active Learning, as discussed in the previous sections, we conclude that the Active Ensemble Learning did not bring improvements in terms of the accuracies of the final models for the datasets we used. Both approaches are able to produce highly accurate models against small, incrementally increasing datasets. This is one of the strengths of the proposed methods, especially in the Data Stream setting. However, in most cases, Active Learning tended to slow the learning process down, in that it added more time to build the final model. Further, the Active Ensemble Learning requires a continuous effort from the user. Recall that the user plays an important role in the Active Learning process, in that a domain expert is constantly required to verify the labels. Our results thus indicate that, for the datasets we used, the added effort and computation time does not bring much in terms of improving the overall accuracy of the final model. These results stand in contrast to our hypothesis, which was that Active Learning would result in increased accuracy. Our further analysis leads us to believe that the application of Active Learning is more suited to domains with noisy data, imbalanced data, late arriving labels, and/or many missing values. We plan to explore this further. Interestingly, our results do indicate that the Active Learning process requires smaller ensembles. This is a promising result, especially from a cost-sensitive learning point of view. In the next chapter we present our thesis contributions and future work.

Chapter 6

Conclusions

In this thesis the Ensemble Learning paradigm was extended to include Active Learning, as applied to Data Streams. Also, a discussion on the Ensemble Learning methods and the Active Learning methods was presented, highlighting their strengths and weaknesses in regards to the Data Stream setting.

6.1 Contributions

Recall that one of the main motivations of this thesis was to investigate the performance of the Ensemble Learning and the Active Learning with online mode. We were interested in finding the benefit of using the Active Ensemble Learning and to see the value that Active Learning will add to the Ensemble Learning paradigm. For this reason, in Chapter 4 the developed framework was presented and a discussion on the way it works was detailed. The framework used was divided into different stages. In the last stage of it, the dataset is tested in online mode using Active Ensemble Learning techniques.

In this study, four different classifiers were tested using the Active Ensemble Learning framework. One of the main goals of this thesis was to find out when to use which Ensemble Learning methods, as well as to explore the suitability of these four base classifiers. Our results indicated that the kNN algorithm produced the models with the highest accuracy in the Active Learning setting. This stands in contrast with the current state-of-the art, in which many researchers consider Hoeffding trees to be the benchmarking golden standard, and needs more investigation. Another prominent goal was to determine when the Active Ensemble Learning framework is beneficial in terms of improving the accuracy of the final models. Overall, it is concluded that Active Ensemble Learning did not add a noticeable value to the model's accuracy. In most cases it resulted in the same accuracy as in the Ensemble Learning or increased the error rate.

However, Active Learning leads to smaller ensembles, which may be beneficial in a cost-sensitive learning setting.

Another contribution was to study and model the time needed to classify new examples. It was evident that the classifiers used require constant time and memory. As the data size increased, the required memory increased as well. In some cases, the time also increased. Also, it was found that the user role is critical in the case of the Active Learning process. It requires constant effort by the user to evaluate the unclassified instances, to choose the one with the highest prediction probability, and then to, classify the selected instances and appending them to the stream. The part of the framework where it required the user to finish needed the most effort and time in the process. This might be unrealistic in the case of classifying a large Data Streams. Even if the user dedicates their time to the process, the human error still exists because of possible distractions or fatigue as time goes by. Also, the nature of the data and complexity may affect people's decisions.

We further analyzed the optimal size of the training set. As discussed earlier, a Data Stream contains a large amount of data. This data should be handled in a way that captures the important information from the smallest training size as possible. By nature, the Ensemble Learning methods allow the algorithm to train different classifiers at the same time, which helps reduce the learning process time. Therefore, it was decided to use the Ensemble Learning methods within the Active Learning process. In this way it was possible to combine the strength of both methods. The result as shown in Chapter 5, shows the classifiers' ability to train the classifier with a small size of training sets. Also, we were able to increase the performance of the classifier by adding only 10 new classified instances from each class.

Another contribution of this thesis is the evaluation of the Ensemble size, as each of the Ensemble and Active Learning methods were tested with different base classifiers. It was decided to repeat the whole process with different ensemble sizes. After considering methods results from both methods, it is concluded that the Ensemble Learning models were best using the ensemble size of 25, whereas the Active Learning resulted in the best models using the ensemble size of 10.

6.2 Future Work

The data sets we used in this study were balanced, contained correct class labels, and were subject to little noise. We are interested in extending our work to include other datasets with higher levels of imbalance. Further, it would be worthwhile to consider datasets with incorrect, missing, or late arriving labels. In such a setting, the run time overhead associated with Active Learning may be less relevant in order to ensure highly accurate models. The fact that Active Learning leads to smaller ensembles also needs further attention.

We also plan to continue additional experiments to analyze the performance of the Random Subspace method and the Stacking method. Recall that this thesis did not evaluate their performances, because of their computational complexity and the memory limitation.

Data Streams are susceptible to Concept Drift. Our work did not address this issue and we plan to do so in the future.

As mentioned before, the fact that the kNN algorithm outperformed the Hoeffding trees algorithm needs further attention. On the other hand using C4.5 and Naïve Bayes in an Active Learning setting did not work well. In some dataset the accuracy results were unrealistic since they resulted in a zero-error percentage. In other datasets, the error rate increased to almost 50%. Therefore, as future work, it is interesting to look into creating other online versions of these two algorithms, as well as to explore just-in-time model building.

Bibliography

[1] Aggarwal, C. C. (2007). *Data streams: models and algorithms* (Vol. 31): Springer.

[2] Agrawal, G. L., & Gupta, H. (2013). Optimization of C4. 5 Decision Tree Algorithm for Data Mining Application. *International Journal of Emerging Technology and Advanced Engineering, 3*(3), 341-345.

[3] Ali, W., Shamsuddin, S. M., & Ismail, A. S. (2012). Intelligent Naïve Bayes-based approaches for Web proxy caching. *Knowledge-Based Systems, 31*(0), 162-175. doi: http://dx.doi.org/10.1016/j.knosys.2012.02.015

[4] Amores, J., Sebe, N., & Radeva, P. (2006). Boosting the distance estimation: Application to the k-nearest neighbor classifier. *Pattern Recognition Letters, 27*(3), 201-209.

[5] Arbib, M. A. (2003). *The handbook of brain theory and neural networks*: MIT press.

[6] Balasubramanian, V. N., Chakraborty, S., Ho, S.-S., Wechsler, H., & Panchanathan, S. (2014). Active Learning. In V. N. Balasubramanian, S.-S. Ho & V. Vovk (Eds.), *Conformal Prediction for Reliable Machine Learning* (pp. 49-70). Boston: Morgan Kaufmann.

[7] Battula, B. P., & Prasad, R. S. (2013). An Overview of Recent Machine Learning Strategies in Data Mining. *International Journal of Advanced Computer Science & Applications, 4*(3).

[8] Beniwal, S., & Arora, J. (2012). Classification and feature selection techniques in data mining. *International Journal of Engineering Research & Technology (IJERT), 1*(6).

[9] Bhatia, N. (2010). Survey of nearest neighbor techniques. *arXiv preprint arXiv:1007.0085*.

[10] Bifet, A. (2009). *Adaptive learning and mining for data streams and frequent patterns*. (PhD thesis), Universitat Polit´ecnica de Catalunya.

[11] Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010). Moa: Massive online analysis. *The Journal of Machine Learning Research*, *11*, 1601-1604.

[12] Bifet, A., Holmes, G., Pfahringer, B., & Gavaldà, R. (2009). Improving adaptive bagging methods for evolving data streams *Advances in Machine Learning* (pp. 23-37): Springer.

[13] Bifet, A., & Kirkby, R. (2009a). Data stream mining a practical approach. 68-69.

[14] Bifet, A., & Kirkby, R. (2009b). Data stream mining a practical approach.

[15] Bouckaert, R. R. (2006). Voting massive collections of bayesian network classifiers for data streams *AI 2006: Advances in Artificial Intelligence* (pp. 243-252): Springer.

[16] Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123-140.

[17] Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*: CRC press.

[18] Bryll, R., Gutierrez-Osuna, R., & Quek, F. (2003). Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern recognition*, *36*(6), 1291-1302.

[19] Cao, J., Kwong, S., & Wang, R. (2012). A noise-detection based AdaBoost algorithm for mislabeled data. *Pattern recognition*, *45*(12), 4451-4465.

[20] The causality workbench, . (2014). 2013, from http://www.causality.inf.ethz.ch /activelearning.php

[21] Corporation, E. (2014). Digital Universe. Retrieved 2014, from http://canada .emc.com/leadership/programs/digital-universe.htm

[22] Cunningham, P., & Delany, S. J. (2007). k-Nearest neighbour classifiers. *Mult Classif Syst*, 1-17.

[23] de Souza, E. N., & Matwin, S. (2013). Improvements to Boosting with Data Streams *Advances in Artificial Intelligence* (pp. 248-255): Springer.

[24] Dietterich, T. G. (2000a). Ensemble methods in machine learning *Multiple classifier systems* (pp. 1-15): Springer.

[25] Dietterich, T. G. (2000b). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning, 40*(2), 139-157.

[26] Dietterichl, T. G. (2002). Ensemble learning. *The handbook of brain theory and neural networks*, 405-408.

[27] Domingos, P., & Hulten, G. (2000). *Mining high-speed data streams.* Paper presented at the Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining.

[28] Fan, D. W., Chan, P. K., & Stolfo, S. J. (1996). *A comparative evaluation of combiner and stacked generalization.* Paper presented at the Proceedings of AAAI-96 workshop on Integrating Multiple Learned Models.

[29] Freund, Y. (2001). An adaptive version of the boost by majority algorithm. *Machine learning, 43*(3), 293-318.

[30] Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence, 14*(771-780), 1612.

[31] Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences, 55*(1), 119-139.

[32] Freund, Y., Seung, H. S., Shamir, E., & Tishby, N. (1997). Selective sampling using the query by committee algorithm. *Machine learning, 28*(2-3), 133-168.

[33] Friedman, J., Hastie, T., & Tibshirani, R. (2000). Special invited paper. additive logistic regression: A statistical view of boosting. *Annals of statistics*, 337-374.

[34] Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2010). Data stream mining *Data Mining and Knowledge Discovery Handbook* (pp. 759-787): Springer.

[35] Ghosh, A. K. (2006). On optimum choice of< i> k</i> in nearest neighbor classification. *Computational Statistics & Data Analysis, 50*(11), 3113-3123.

[36] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter, 11*(1), 10-18.

[37] Han, J., Kamber, M., & Pei, J. (2006). *Data mining: concepts and techniques*: Morgan kaufmann.

[38] Hoeglinger, S., & Pears, R. (2007). *Use of hoeffding trees in concept based data stream mining.* Paper presented at the Information and Automation for Sustainability, 2007. ICIAFS 2007. Third International Conference on.

[39] Hoens, T. R., Chawla, N. V., & Polikar, R. (2011). *Heuristic updatable weighted random subspaces for non-stationary environments.* Paper presented at the Data Mining (ICDM), 2011 IEEE 11th International Conference on.

[40] Hulten, G., Domingos, P., & Spencer, L. (2005). Mining massive data streams. *Journal of Machine Learning Research 1*.

[41] Hulten, G., Spencer, L., & Domingos, P. (2001). *Mining time-changing data streams.* Paper presented at the Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining.

[42] IBM. Crossvalidation for Selection of k (nearest neighbor algorithms). 2013, from http://pic.dhe.ibm.com/infocenter/spssstat/v20r0m0/index.jsp?topic=/com.ibm.spss.statistics.help/alg_knn_training_crossvalidation.htm

[43] Ibrahim, H. E., Badr, S. M., & Shaheen, M. A. (2012). Adaptive Layered Approach using Machine Learning Techniques with Gain Ratio for Intrusion Detection Systems. *International Journal of Computer Applications, 56*.

[44] Ikonomovska, E., Loskovska, S., & Gjorgjevik, D. (2007). *A survey of stream data mining.* Paper presented at the Proceedings of 8th National Conference with International Participation, ETAI.

[45] Jin, R., & Agrawal, G. (2003). *Efficient decision tree construction on streaming data.* Paper presented at the Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.

[46] Khan, M., Ding, Q., & Perrizo, W. (2002). k-nearest neighbor classification on spatial data streams using P-trees *Advances in Knowledge Discovery and Data Mining* (pp. 517-528): Springer.

[47] Ko, A. H.-R., Sabourin, R., & de Souza Britto Jr, A. (2007). A new dynamic ensemble selection method for numeral recognition *Multiple Classifier Systems* (pp. 431-439): Springer.

[48] Krempl, G., Brzezinski, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., . . . Stefanowski, J. (2014). Open Challenges for Data Stream Mining Research. 1-10.

[49] Law, Y.-N., & Zaniolo, C. (2005). An adaptive nearest neighbor classification algorithm for data streams *Knowledge Discovery in Databases: PKDD 2005* (pp. 108-120): Springer.

[50] Lowd, D., & Domingos, P. (2005). *Naive Bayes models for probability estimation.* Paper presented at the Proceedings of the 22nd international conference on Machine learning.

[51] Mamitsuka, N. A. H. (1998). *Query Learning Strategies using Boosting and Bagging.* Paper presented at the Machine Learning: Proceedings of the Fifteenth International Conference (ICML'98).

[52] Mannila, H. (1996). *Data Mining: Machine Learning, Statistics, and Databases.* Paper presented at the SSDBM.

[53] Mazid, M. M., Ali, S., & Tickle, K. S. (2010). *Improved C4. 5 algorithm for rule based classification.* Paper presented at the Proceedings of the 9th WSEAS international conference on Artificial intelligence, knowledge engineering and data bases.

[54] MedCalc. MedCalc. 2014, from http://www.medcalc.org/manual/friedman_test.php

[55] Melville, P., & Mooney, R. J. (2003). *Constructing diverse classifier ensembles using artificial training examples.* Paper presented at the IJCAI.

[56] Melville, P., & Mooney, R. J. (2004). *Diverse ensembles for active learning.* Paper presented at the Proceedings of the twenty-first international conference on Machine learning.

[57] Mena-Torres, D., & Aguilar-Ruiz, J. S. (2014). A similarity-based approach for data stream classification. *Expert Systems with Applications, 41*(9), 4224-4234. doi: http://dx.doi.org/10.1016/j.eswa.2013.12.041

[58] Mielniczuk, J., & Teisseyre, P. (2014). Using random subspace method for prediction and variable importance assessment in linear regression.

Computational Statistics & Data Analysis, 71(0), 725-742. doi: http://dx.doi.org/10.1016/j.csda.2012.09.018

[59] Muhivumundo, D., & Viktor, H. (2011). *Detecting Data Duplicates through Active Ensemble-based Learning*. Paper presented at the the IEEE African Conference on Software Engineering and Applied Computing (ACSEAC 2011), Cape Town, South Africa.

[60] Muhivuwomunda, D. (2010). *Data De-Duplication through Active Learning.* (Master degree), University of Ottawa.

[61] Munro, R., & Chawla, S. (2004). An Integrated Approach to Mining Data Streams. 1-8. http://www.robertmunro.com/research/munro04integrated.pdf

[62] Oracle. (2008). Oracle® Data Mining Concepts. Retrieved 30/08/2014, from http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/algo_svm.htm#i1006563

[63] Oza, N. (2001). *Online Ensemble Learning.* (PhD), University of California, Berkeley.

[64] Oza, N. C., & Russell, S. (2005). *Online bagging and boosting.* Paper presented at the Systems, man and cybernetics, 2005 IEEE international conference on.

[65] Panov, P., & Džeroski, S. (2007). *Combining bagging and random subspaces to create better ensembles*: Springer.

[66] Perrizo, W., Ding, Q., Khan, M., & Denton, A. (2007). An efficient weighted nearest neighbour classifier using vertical data representation. *International Journal of Business Intelligence and Data Mining, 2*(1), 64-78.

[67] Piramuthu, S. (2004). Evaluating feature selection methods for learning in data mining applications. *European journal of operational research, 156*(2), 483-494.

[68] Polikar, R. (2006). Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE, 6*(3), 21-45.

[69] Quinlan, J. R. (1987). Simplifying decision trees. *International journal of man-machine studies, 27*(3), 221-234.

[70] Quinlan, J. R. (1993). *C4. 5: programs for machine learning* (Vol. 1): Morgan kaufmann.

[71] Quinlan, J. R. (1996). *Bagging, boosting, and C4. 5.* Paper presented at the AAAI/IAAI, Vol. 1.

[72] Quinlan, J. R. (1996). Improved use of continuous attributes in C4. 5. *arXiv preprint cs/9603103*.

[73] Quinlan, K. (1999). Decision Tree Discovery.

[74] Rastgoo, M., Lemaitre, G., Palou, X. R., Miralles, F., & Casale, P. (2012). *Pruning adaboost for continuous sensors mining applications.* Paper presented at the Workshop on Ubiquitous Data Mining.

[75] Read, J., Bifet, A., Pfahringer, B., & Holmes, G. (2012). Batch-incremental versus instance-incremental learning in dynamic and evolving data *Advances in Intelligent Data Analysis XI* (pp. 313-323): Springer.

[76] Refaeilzadeh, P., Tang, L., & Liu, H. (2007). *On comparison of feature selection algorithms.* Paper presented at the Proceedings of AAAI Workshop on Evaluation Methods for Machine Learning II.

[77] Repository, UCI. Australian Dataset Retrieved 2013 http://archive.ics.uci.edu /ml/datasets/Statlog+(Australian+Credit+Approval)

[78] Repository, UCI. Chess Dataset. Retrieved 2012 https://archive.ics.uci.edu/ ml/datasets/Chess+(King-Rook+vs.+King-Pawn)

[79] Repository, UCI. Spambase dataset. Retrieved 2012 https://archive.ics.uci.edu /ml/datasets/Spambase

[80] Ruggieri, S. (2002). Efficient C4. 5 [classification algorithm]. *Knowledge and Data Engineering, IEEE Transactions on, 14*(2), 438-444.

[81] Rutkowski, L., Pietruczuk, L., Duda, P., & Jaworski, M. (2013). Decision trees for mining data streams based on the McDiarmid's bound. *Knowledge and Data Engineering, IEEE Transactions on, 25*(6), 1272-1279.

[82] Schapire, R. E. (2003). The boosting approach to machine learning: An overview *Nonlinear estimation and classification* (pp. 149-171): Springer.

[83] Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison, 52*, 55-66.

[84] Seung, H. S., Opper, M., & Sompolinsky, H. (1992). *Query by committee.* Paper presented at the Proceedings of the fifth annual workshop on Computational learning theory.

[85] Sewell, M. (2008). Ensemble learning. *RN*, *11*(02).

[86] Shi, H., Wang, Z., Webb, G. I., & Huang, H. (2003). A new restricted Bayesian network classifier *Advances in Knowledge Discovery and Data Mining* (pp. 265-270): Springer.

[87] stackoverflow. 2013, from http://stackoverflow.com/questions/13610074/is-there-a-rule-of-thumb-for-how-to-divide-a-dataset-into-training-and-validatio

[88] Stefanowski, J., & Pachocki, M. (2009). Comparing performance of committee based approaches to active learning. *Recent Advances in Intelligent Information Systems*, 457-470.

[89] Su, M.-Y. (2011). Using clustering to improve the KNN-based classifiers for online anomaly network traffic identification. *Journal of Network and Computer Applications, 34*(2), 722-730. doi: http://dx.doi.org/10.1016/j.jnca.2010.10.009

[90] Theodorsson-Norheim, E. (1987). Friedman and Quade tests: BASIC computer program to perform nonparametric two-way analysis of variance and multiple comparisons on ranks of several related samples. *Computers in Biology and Medicine, 17*(2), 85-99. doi: http://dx.doi.org/10.1016/0010-4825(87)90003-5

[91] Ting, K. M., & Witten, I. H. (1997). Stacked Generalization: when does it work?

[92] Trambadiya, B. (2012). A Comparative study of Stream Data mining

[93] Algorithms *International Journal of Engineering and Innovative Technology (IJEIT), 2*(3).

[94] Tsymbal, A. (2004). The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*.

[95] Waikato, T. U. o. wikispaces. from http://weka.wikispaces.com/How+do+I+divide+a+dataset+into+training+and+test+set%3F

[96] Wang, H., Fan, W., Yu, P. S., & Han, J. (2003). *Mining concept-drifting data streams using ensemble classifiers.* Paper presented at the Proceedings of the

ninth ACM SIGKDD international conference on Knowledge discovery and data mining.

[97] Wang, P., Zhang, P., & Guo, L. (2012). *Mining Multi-Label Data Streams Using Ensemble-Based Active Learning.* Paper presented at the SDM.

[98] Witten, I. H., & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*: Morgan Kaufmann.

[99] Wolpert, D. H. (1992). Stacked generalization. *Neural networks, 5*(2), 241-259.

[100] Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., . . . Philip, S. Y. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems, 14*(1), 1-37.

[101] Ye, N. (2003). *The handbook of data mining* (Vol. 24): Lawrence Erlbaum Associates Mahwah, NJ.

[102] Zhang, R., Jagadish, H. V., Dai, B. T., & Ramamohanarao, K. (2010). Optimized algorithms for predictive range and KNN queries on moving objects. *Information Systems, 35*(8), 911-932. doi: http://dx.doi.org/10.1016/j.is.2010.05.004

[103] Zhou, Z.-H. Ensemble Learning http://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/springerebr09.pdf

[104] Zhou, Z.-H. (2012). *Ensemble methods: foundations and algorithms*: CRC Press.

[105] Zhu, X., & Yang, Y. (2008). A lazy bagging approach to classification. *Pattern recognition, 41*(10), 2980-2992. doi: http://dx.doi.org/10.1016/j.patcog.2008.03.008

[106] Zhu, X., Zhang, P., Lin, X., & Shi, Y. (2007). *Active learning from data streams.* Paper presented at the Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on.

[107] Zhu, Y., Liu, J., & Chen, S. (2009). Semi-random subspace method for face recognition. *Image and Vision Computing, 27*(9), 1358-1370. doi: http://dx.doi.org/10.1016/j.imavis.2008.12.009

[108] Zliobaite, I., Bifet, A., Pfahringer, B., & Holmes, G. (2014). Active learning with drifting streaming data. *IEEE transactions on neural networks and learning systems*, *25*(1), 27-39.