SCUT-DS: Methodologies for Learning in Imbalanced Data Streams

by

Olubukola Olaitan

Supervisor

Professor Herna Viktor

A thesis submitted in partial fulfillment of the requirements for the

Master of Science degree in Computer Science

School of Electrical Engineering and Computer Science

Faculty of Engineering

uOttawa

University of Ottawa

Abstract

The automation of most of our activities has led to the continuous production of data that arrive in the form of fast-arriving streams. In a supervised learning setting, instances in these streams are labeled as belonging to a particular class. When the number of classes in the data stream is more than two, such a data stream is referred to as a multi-class data stream. Multi-class imbalanced data stream describes the situation where the instance distribution of the classes is skewed, such that instances of some classes occur more frequently than others. Classes with the frequently occurring instances are referred to as the majority classes, while the classes with instances that occur less frequently are denoted as the minority classes.

Classification algorithms, or supervised learning techniques, use historic instances to build models, which are then used to predict the classes of unseen instances. Multi-class imbalanced data stream classification poses a great challenge to classical classification algorithms. This is due to the fact that traditional algorithms are usually biased towards the majority classes, since they have more examples of the majority classes when building the model. These traditional algorithms yield low predictive accuracy rates for the minority instances and need to be augmented, often with some form of sampling, in order to improve their overall performances.

In the literature, in both static and streaming environments, most studies focus on the binary class imbalance problem. Furthermore, research in multi-class imbalance in the data stream environment is limited. A number of researchers have proceeded by transforming a multi-class imbalanced setting into multiple binary class problems. However, such a transformation does not allow the stream to be studied in the original form and may introduce bias. The research conducted in this thesis aims to address this research gap by proposing a novel online learning methodology that combines oversampling of the minority classes with cluster-based majority class under-sampling, without decomposing the data stream into multiple binary sets. Rather, sampling involves continuously selecting a balanced number of instances across all classes for model building. Our focus is on improving the rate of correctly predicting instances of the minority classes in multi-class imbalanced data streams, through the introduction of the Synthetic Minority Over-sampling Technique (SMOTE) and Cluster-based Under-sampling - Data Streams (SCUT-DS) methodologies. In this work, we dynamically balance the classes by utilizing a windowing mechanism during the incremental sampling process. Our SCUT-DS algorithms are evaluated using six different types of classification techniques, followed by comparing their results against a state-of-the-art algorithm. Our contributions are tested using both synthetic and real data sets. The experimental results show that the approaches developed in this thesis yield high prediction rates of minority instances as contained in the multiple minority classes within a non-evolving stream.

Acknowledgements

I am grateful to almighty God for His grace.

I would to like express a special gratitude to my supervisor for her unwavering support during my studies and the research. I am grateful for the opportunity you gave me.

I am grateful to my parents, my siblings and my husband for always being there. Your sacrifice, encouragements and supports have gone a very long way and are really appreciated. I would like to appreciate my daughter for her sacrifice too.

My appreciation also goes to my friend, Ali Pesaranghader for his help. Finally, I would like to express my gratitude to all the lecturers and staff of the University of Ottawa, friends and all those who in one way or the other contributed to making this journey a success.

Table of Contents

List of Tables

List of Algorithms

List of Acronyms

AI	Artificial Intelligence
AOG	Algorithm Output Granularity
API	Application Program Interface
ARFF	Attribute Relation File Format
ASCII	American Standard Code for Information Interchange
ATM	Automated Teller Machine
AUC	Area under Curve
CSV	Comma Separated Values
DBMS	Database Management Systems
DoS	Denial of Service
FN	False Negative
FP	False Positives
GUI	Graphical User Interface
IDE	Integrated Development Environment
KNN	K-Nearest-Neighbor
LED	Light Emitting Diode

McELM	Meta-Cognitive Extreme Learning Machine
ML	Machine Learning
MOA	Massive Online Analysis
PCA	Principal Component Analysis
PDA	Personal Digital Assistant
PR	Precision-Recall curve
ROC	Receiver Operating Characteristics
OVA	One-Versus-All
OVO	One-Versus-One
SCUT	SMOTE and Cluster-based Under-sampling
SCUT-DS	SMOTE and Cluster-based Under-sampling – Data Streams
SMOTE	Synthetic minority over-sampling technique
TN	True Negatives
TP	True Positives
UCI	University of California, Irvine
VFDT	Very Fast Decision Tree
WOS-ELM	Weighted Online Sequential Extreme Learning Machine

XRFF	Xml-attribute Relation File Format

Chapter 1

Introduction

Technology, Automation, the era of Big Data and Data Analytics have redefined the data generation frequency of our daily activities and the myriads of knowledge that may be discovered from these data [1]. These activities include shopping, health, banking, social networking and infrastructure monitoring. The recent improvement and decline in the cost of technologies that are used to monitor these events have increased their availability and applicability to our lives [2]. For example, sensor technology may be deployed for real-time health monitoring in patients [3], [4], and for monitoring the durability of infrastructure such as bridges [2].

The data produced are mined to discover interesting and useful knowledge under a field named Artificial Intelligence (AI). Machine Learning (ML) and data mining are sub-fields of AI used for these nugget discoveries. In data mining, data are usually stored in Database Management Systems (DBMS) for analysis. The knowledge discovered from these analyses have been applied to many aspects of life, including medicine, the environment, and security. Some activities generate data continuously, thus making data storage impossible because of their size. These data are referred to as data streams, and because they are generated in real time, they need to be analyzed in real time too. Examples of data streams include sensor network, Automated Teller Machine (ATM) transactions and the data generated on the web. Data stream mining involves analyzing these data as they are being produced. There are different type of tasks used in discovering patterns and knowledge from data streams, including classification, regression, clustering, frequent pattern, and outlier or anomaly analysis [5]. Classification involves the prediction of the labels of data in a stream based on the classes that were assigned to previously seen data [5]. For example, instances in the data set are categorized as belonging to certain classes based on some criteria such as business requirements and problem definition. As an illustration, examples of data from logs of network services can be labeled as being excellent, good, fair or poor based on some criteria. Past logs of such network services may be used in building models that will be used to predict the labels of incoming streams of network service log. Depending on the scenario, classification may be used in fraud detection, identifying decline in the quality of network service, network intrusion and Denial of Service (DoS). There are situations where some labels occur infrequently compared to others, because such classes of data rarely occur. For example, in fraud detection, instances that belong to the fraudulent class will be sparse in such data set because fraudulent transactions rarely happen. This results in unbalanced data sets, because some classes will occur more frequently than others. The ratio of the non-frequent class to the frequent class may be as high as 1:100 or more [6]. Data sets of this nature are referred to as imbalanced data set.

The number of classes that could be identified in a data set depends on the domain, problem definition and business logic. Data sets that contain two classes are referred to as binary-class, while those with more than two are referred to as multi-class data sets. Thus, an imbalanced data set with more than two classes is referred to as multi-class imbalanced data set. The classes with the frequently occurring examples are referred to as majority (or negative class) while those that are sparse are categorized as minority (or positive classes).

The multi-class imbalance problem occurs when the number of data in some classes (majority classes) have very high frequencies, while the other classes (minority classes) have very low frequencies [7], [8]. Different techniques have been used in data mining to address multi-class imbalance. Some of these techniques have involved decomposing the multi-class imbalance classification problem into binary-class imbalance classification problems [9], [10], [11], [12]. Class reduction may be achieved by combining a minority class with one other class in the data set, and this is called One-Versus-One (OVO). Another method, which is referred to as One-Versus-All (OVA), involves combining one minority class at a time with all the other classes in the data set. The limitation of class decomposition is that only one minority class may be studied at a time when this method is used. Sampling or algorithmic techniques, which are methods used in addressing class imbalance, may then be applied afterwards on the decomposed data sets. Class decomposition techniques are not usually efficient for multi-class imbalance data classification because the minority classes, which are of the main concern, are not analyzed in their natural forms which may lead to poor and biased predictive performance [7].

Unlike two-class imbalance classification, multi-class imbalance classification without resorting to class decomposition has not been extensively studied, especially against data streams [13], [14]. Therefore, in this thesis, the focus is on improving the recognition rate of the minority classes in multi-class imbalanced data stream without decomposition to binary classification. Here, we define the recognition rate as the total number of correctly classified instances of a particular label against the total number of instances of that label in the data set. Specifically, this thesis seeks to extend Synthetic Minority Over-sampling Technique (SMOTE) and Cluster-based Under-sampling (SCUT), which have been previously applied in a static setting [8], to a data stream. The remainder of this chapter explains the motivations for the research and the objectives of the study.

1.1 Motivations

Generally, imbalanced data set are difficult to classify because the models are built with insufficient examples from the minority classes and traditional classification algorithms tend to be biased towards the classes with many examples [15], [16]. This makes it difficult in classification task to build models with high prediction accuracy on the minority classes. Compared to binary-class imbalanced data sets, the problem is further compounded in multi-class imbalanced data set [13], [14].

Ironically, the “difficult to learn” classes are in most cases the classes of interest. Multi-class imbalance data classification problem in static data was addressed in [8], [14], while multi-class imbalanced learning in data stream without reduction to two-class was studied in [13] .

The ability to improve the recognition rate of these important classes will reduce the research gap in class imbalanced learning in data stream mining and data mining. In addition, the accurate prediction of these rare classes or events will help in quickly identifying and addressing issues in monitoring, which could be catastrophic if left to linger for long. The correct classification of the positive examples in multi-class imbalance data set in data stream mining could, for instance, aid an organization to avoid financial loss from fraudulent activities, or improve network service by being able to detect decline in quality of service quickly.

1.2 Thesis Objective

This thesis seeks to improve the recognition rate of multi-minority classes in multi-class imbalanced data learning of data streams using sampling methods. The aim is to improve the recognition rate of minority classes in data stream mining without resorting to class reduction, by building models that are unbiased towards the majority classes in the data set. This research focuses on how to increase the examples of minority instances in the training set. Minority instances may be increased by generating synthetic instances using techniques such as SMOTE [17], [18] or by keeping minority examples from past data chunks. Also in this thesis, we concentrate on how to get balanced training sets across all classes present, by reducing the majority examples. In addition, the thesis focuses on how to update models in data stream classification task. It concentrates only on minority instances, because standard classifiers are known to have high recognition rates with majority instances.

Specifically, this research proposes addressing the multi-class imbalance data classification problem in data stream mining. It aims to extend the above-mentioned work of Agrawal et al. [8] in which a hybrid sampling method, named SCUT, was used to address the multi-class imbalance problem in static data. Their method aids at balancing the frequencies of the different classes in the training data set, thus improving the classification accuracy of multi-class imbalance data sets. Specifically, we introduce two new approaches in this thesis, namely SCUT-DS and SCUT-DS++. The SCUT-DS algorithm increases the number of the minority instances in the training set by generating synthetic instances, while SCUT-DS++ increases the presence of the minority instances in the training set by generating synthetic instances and accumulating past minority instances.

The research analyses and reviews the performance of the designed methodologies against a state-of-the-art algorithm [19], which is henceforth referred to as INTER. In summary, this thesis aims to answer the following research questions. First, we explore how resilient the methodologies developed in this research, namely SCUT-DS, SCUT-DS++ and INTER are to noise. Secondly, we aim to investigate whether there is any difference in the recognition rate of the minority instances amongst the three techniques mentioned. Thirdly, we seek to identify the approach that generalizes better on incoming streams between SCUT-DS and SCUT-DS++. Finally, we aim to investigate the effect of the choice of classification algorithms on the new algorithms we created.

To this end, the algorithms developed in this thesis serve to narrow the research gap in imbalanced data sets classification, particularly in multi-class imbalanced setting. Thus, our main contributions are, first, extending SCUT to the streaming environment and, second, directly addressing multi-class imbalance classification without class decomposition. Another contribution to the literature is the oversampling approaches designed to augment the minority instances in the training sets. Also, excessive computational memory and time requirements are avoided with the sampling rates used. Specifically, the sampling rates prevented extreme sampling in highly imbalanced data streams. Lastly, our contribution led to the development of algorithms with a wide range of application, because they are flexible and require little prior domain knowledge.

The organization of the thesis is described in the next subsection.

1.3 Thesis Organization

The remainder of this thesis is organized into six chapters as follows. Chapter 2 provides relevant background on data mining and data stream mining. Chapter 3 reviews class imbalance classification. It emphasizes the categorization of imbalanced learning, challenges of imbalanced data set learning and solutions provided in the literature. Studies relevant to this thesis are discussed, to lay the foundation for the new methods. Chapter 3 also discusses the application of multi-class imbalanced stream classification. Chapter 4 focuses on our SCUT-DS and SCUT-DS++ methodologies and introduce the performance measures employed in this study. That is, the chapter presents the framework of the proposed algorithm, as well as the basis and justification for the performance evaluation and statistical analysis. Chapter 5 discusses the experimental design and the analyses of the results of the experimentation, while Chapter 6 concludes the thesis and provides suggestions for future works.

Chapter 2

Background

An abundance of data has necessitated the development of novel algorithms to mine them in order to discover useful knowledge. Today, such knowledge discovered is being applied to almost all aspects of our life. Data stream mining is a growing subfield of Machine Learning that aims to build near real-time models against continuously flowing data. This chapter presents the basic concepts and relevant background information on data stream mining. First in Section 2.1, we briefly introduce important concepts about data mining in order to set the stage. Next, in Section 2.2, we discuss data stream mining, its characteristics, and the different techniques used to adapt static data mining algorithms to the non-stationary environment. In Section 2.3, we explore forgetting mechanisms, an essential method that is used for building relevant and efficient models in data stream classification. In Section 2.4, we give insights to topics discussed in Chapter 2. Finally, in Section 2.5, we conclude the chapter.

2.1 Overview of Data Mining

The improvement and reliance on technology has led to an explosion in data generation. Data mining reduces the gap between the data generated and information derived from it. Hence, data mining helps to automate the process of discovering knowledge from data [5]. Major data mining tasks include classification, regression, clustering, frequent pattern, and outlier or anomaly analysis [5].

Classification and regression are both used for prediction; while classification is used for predicting categorical data (discrete and unordered), regression is used for predicting numerical or continuous-valued data [5]. Both tasks involve building models by setting aside a subset of data as the training data and another subset as the test data. Classification is also referred to as supervised learning, because the labeled instances in the training set are used to build models that will be used in predicting the labels of the test data [5]. The accuracy and performance of the model built is improved by methods such as cross validation, holdout and bootstrapping [5], [20]. These techniques build many learners using various samples of the training and test data and select the model with the highest accuracy.

Clustering, also known as unsupervised learning is used for discovering the grouping of instances in data sets [5]. The task is to partition the instances into groups based on their homogeneity using some measures [5], [21]. These measures include similarity or dissimilarity measures using methods such as distance (Euclidean, Mahalanobis and Manhattan distance), density and, nearest neighbors. Clustering may be used for taxonomy, for example in the fields of biology, medicine, urban planning, and market research among many others [22]. It is used to group homogenous objects together and to separate the heterogeneous ones [22]. For instance, in market research, clustering analysis may be used to identify groups of customers with similar buying patterns, so that specific promotions and advertisement may be used to target these groups in order to increase the company's revenue [23]. Many clustering algorithms have been developed in literature, they include the K-means and Expectation Maximization (EM) algorithms [5], [21].

Frequent pattern mining is another task in data mining. It is used for discovering association or correlation between instances in the data, by grouping items or sub-items that occur frequently together as a set. Frequent pattern mining has been applied too in marketing research [5].

Another data mining task is outlier analysis which is used to identify data with anomalous characteristics, i.e. data with features that are different from majority of the data contained in the data set. These data objects are then categorized as exceptions or noise. This task have been meaningfully applied in fields such as fraud and intrusion detection [5].

Traditional data mining is based on static data, where it is possible to store data in a repository and analyze the data as a whole. However, innovations such as social networking and sensor technology render full data storage problematic, or even unfeasible, because of the large volume of data that are continuously generated. Thus, there is need to develop novel ways of analyzing these data as they are being generated. Data stream mining is the area of data mining where the challenges involved with real time analysis of streaming data are researched. Data stream mining will be discussed in more detail in the next subsection.

2.2 Data Stream Mining

As stated above, today's applications generate data in a stream, and these data streams are analyzed in a different fashion from static data due to their inherent properties. The applications that generate data streams include weather monitoring instruments, video surveillance, sensor networks, industry production process, ATM transactions, and web logs. These data streams have the characteristics that huge amount of data are generated in an endless fashion, leaving no time for multi-scan analysis [5], [24], [25]. Based on the inherent characteristic of this data flow, stream data analysis involves faster processing with little memory within a short amount of time [25]. Also, the result of analysis must be readily available when needed [5], [25].

The challenges with data stream mining [24] include the continuous arrival of data, the need to process data in a fast and efficient way, the need to make results readily available and the potential occurrence of concept drift, where the characteristics of the underlying concepts of the data stream may change over time [26], [27].

In order to solve the first three challenges of data stream mentioned above, data stream tasks summarize previously seen instances using a data summarization technique. A data summarization technique reduces the memory requirements, the computational cost and the time required for processing. It also reduces the size of the data that is used for analysis and the amount of information lost while processing previously seen instances in extremely huge and fast streams. The achievement of the targets imposed by the first two characteristics makes the third characteristic of making results readily available a reality.

Based on a last characteristic which may be present or absent, data stream may be classified into evolving and non-evolving data stream. In contrast to evolving data stream, the underlying distribution of data in non-evolving data streams does not change. Some researches focus on the last challenge above; hence concept drift exists as a study on its own. Data may evolve because the data source has changed or the features of the label changed [26].

The approaches used in solving the first three challenges in data stream mining above are discussed in the following subsection.

2.2.1 Data Stream Mining Techniques

In order to reduce memory usage, improve performance and prediction time in data stream mining, the size of the data encountered so far is systematically reduced before being used for analysis. Moreover, since it is impossible to store the data seen so far, there are mechanisms for efficiently selecting representative data to be used for analysis in streaming environment. Usually in trying to meet the demand of efficiency imposed by data stream mining, the techniques' accuracy is often traded for storage. The representative data used for analysis give an approximate result with an error bound [28]. These techniques involve single scan, real time analysis and or multi-levels abstraction [5]. These methods can be categorized into data-based and task-based. These are going to be discussed in the next two subsections.

2.2.1.1 Data-based Methods

Data-based method may arrive at its reduced representative data by using a subset of the data set or the whole data set [28]. Representative data used for analysis may be obtained by selecting a subset from the data or by transforming the whole data set vertically or horizontally. Synopsis and aggregation falls in the latter category which uses the whole data, while sampling, load shedding and sketching belongs to the former category that uses subset of the data [5], [28]. These techniques reduce the size of the data used for analysis. The different data-based stream mining methods are discussed below.

Sampling: Random sampling is one of the techniques used in selecting subset of data from a stream that will represent the characteristics of the stream at a particular point in time [29]. Representative data are selected using statistical probability. In random sampling, statistical probability involves knowing the size of the data in order to derive the subset data. The first downside of sampling is that the size of the stream is unknown a priori. Because of this, methods such as reservoir sampling were proposed [30]. The second disadvantage is that it may not be good for applications with high fluctuation in data rate [28]. The third drawback is the presence of noise or outliers.

Reservoir sampling keeps a reservoir of a particular data size with the assumption that the elements of the reservoir are randomly replaced as new data flows in. The size of the reservoir is known a prior and is constant, unlike in random sampling where the size of the data seen so far is always recomputed. In [29], the incoming stream is scanned only once, there is no need for always computing the size of the data seen so far as required in random sampling. Therefore, the reservoir may be used for analyses at different points in time without multiple scanning of the data seen in the stream. Babcock et al [31], designed an efficient algorithm for stream mining that extended reservoir sampling to sliding window. Chaudhuri et al in [32] developed a "priority-sample" mechanism, an improvement over random sampling and reservoir sampling. The instances were prioritized and selected into the representative sample based on the assigned weight.

Load shedding: In load shedding, the stream is seen as chunks in a sequence. The data size is reduced by eliminating some chunks from the data stream. Its disadvantages are synonymous to those of sampling [28]. In addition, vital information may be lost because of a sequence that was discarded [24], [28]. Babcock et al [33] used load shedding and sampling to meet up the demand of rise in data rate in a data stream. Tatbul et al [34], also applied load shedding and sliding window on aggregates in stream. Load shedding is effective in the scenario where the stream is extremely fast and there is almost no time for data analysis.

Sketching: Sketching compacts the data by transforming them into smaller size through the use of method such as Principal Component Analysis (PCA) [28]. Such representative data comprises only features that are deemed critical, hence improving the use of such data for analysis.

Synopsis Data Structures: In synopsis data structure, representative data sets are chosen by summarizing recent stream using summarization methods such as wavelet, histograms, frequency moment and quantiles [28]. The summary consists of summary data structure and characteristics of the seen stream [28]. For example, micro-resolution using balanced binary tree stores summary information as nodes of the tree. The levels of the tree represent different levels of abstraction, the leaf nodes the tree represents most recent summary. Further analysis can then be done offline with earlier summaries [24].

Aggregation: In aggregation, recent streams are summarized using aggregate statistical functions like mean and variance. Further analysis on the stream may be done using the aggregated data. Aggregation is not effective in situations where the data distribution fluctuates often [28], because there will always be the need to re-compute the aggregates.

Although data-based task reduces the size of the data that is used for analysis, it does not take into consideration the computational resources such as memory that will be involved in the reduction task. Also, it does not consider how long it will take for the reduced data to be available for use. Data-based tasks are summarized in Table 1.

Table 1. Summary of Data-based Tasks [24].

Technique	Definition	Advantages	Disadvantages
Sampling	Subset of data is used for analysis	Error bounds guaranteed	Poor for detecting anomaly
Load Shedding	Chunk of data is ignored	Efficient for queries	Very poor for anomaly detection
Sketching	Random projection on feature set	Extremely Efficient	May ignore relevant features
Synopsis Structure	Quick transformation	Analysis task independent	Not sufficient for very fast stream

2.2.1.2 Task-based Methods

Similar to the data-based task discussed above, the task-based techniques are also used to reduce the size of data. Apart from only condensing the data, the computational challenges faced when trying to reduce the data size is considered too. These were not considered in the data-based methods. There are three examples of the task-based techniques: sliding window, approximation techniques and algorithm output granularity (AOG).

Sliding Window: This technique was born out of the idea that the most recent chunks are more important in predicting newly arriving data. Data analyses are therefore performed using the most recent historic data [24], [28], [35]. Sliding window removes the need for computing the probability of replacing elements as it is done in sampling. This method utilizes lower memory, because the size of the window is usually small [35] and known. A sliding window may not be favorable to applications with concept drift, because the recent chunk may not be related to the incoming chunk as assumed. Techniques used in sliding windows include sequence-based window and timestamp-based window.

A sliding window may be implemented as a sequence-based window of a particular data size [31]. Examples are collected as they arrive and when a particular size is reached, the stream is paused and processing begins. In a timestamp-based window instead of using data size, time is used to control when the window is cut. Instances that arrive within a timestamp are considered for processing, thus the size of the window may vary depending on the number of instances that arrive within a particular time frame.

The main issue with sequence-based window centers on the specification of the window size [31], [36]. The concerns include, how quickly or how slowly is the window filled and how reactive will the window be to changes in the underlying data distribution. In timestamp-based window the data rate may vary. The challenge with timestamp-based window is the sufficiency of the data that is accumulated at a particular time.

Approximation Algorithm: Data stream mining tasks are grouped into the class of hard problems in algorithm design [28]. Therefore, in order to design efficient algorithms for data stream mining, the developed approximation algorithm includes specified error bounds [28]. The algorithms usually have lower computational complexity at the specified error bound [24], [28].

Algorithm Output Granularity (AOG): AOG are resource-sensitive data stream mining techniques that are also effective for highly fluctuating data streams [24], [28]. AOG is sensitive to memory and time usage. In AOG, the method initially adapts the mining task to the local resources. As the resources are being used up, the earlier structures that were generated by the task are merged in order to reduce the resource requirement thereby adapting to the current available resources [24], [28].

The task-based techniques discussed above are cognizant of the computational requirements in data stream mining. Although as noted above, the choice of the optimal window size in sliding window is non-trivial neither is the setting of standardized error bound and resource limit in both Algorithm Approximation and AOG. In addition, Algorithm Approximation methods may not be able to adapt the data to the resource requirement while the resource aware component in AOG may increase computational cost [24]. Even though sliding window assumes that the most recent window is relevant to analyzing incoming streams, there are methods used in literature to override this approach. Some instances from previously seen examples may be accumulated or the number of chunks made relevant to the task may be increased. Sliding window is good for data streams with non-evolving underlying data distribution. Table 2 below summarizes task-based methods.

Table 2. Summary of Task-based Techniques [24]

Technique	**Definition**	**Advantages**	**Disadvantages**
Approximation Algorithms	Algorithms with error bounds	Efficient	Resource adaptivity with data rates not always possible
Sliding Window	Analyzing most recent streams	General	Ignores part of stream and may not be good for concept drift.
Algorithm Output Granularity	Highly resource aware technique with memory and fluctuating data rates	General	Cost overhead of resource aware component

The mining approaches discussed above form the basis of the algorithms developed in data stream mining. They are used to reduce the data size that is used for analysis in data stream mining tasks in order to lower the computational requirement and to hasten the output of results. Classification in data stream mining is presented in the following subsection.

2.2.2 Classification in Data Stream Mining

Recall that stream data have some inherent characteristics that make methods used in batch mining not to be directly applicable. Hence the algorithms in data stream mining are designed to be resource-aware and less computationally intensive. Generally, techniques used in data stream mining tasks must be resilient and adaptable in incorporating new information from newly arriving streams and outdating irrelevant information from previous streams. Data stream mining considers the concept of evolving data as a study on its own, thus there exist classification of non-evolving data streams and classification of data streams with concept drift. The focus in this thesis is on classification of data streams without concept drift. Similar to the static environment, classification in the streaming environment involves model building and testing. Various types of classification algorithms have been developed for data stream learning in literature, notable amongst these categories are decision trees, Bayesian, meta-learners, function and drift classification techniques [25].

Decision tree classification algorithms, as the name suggests, make use of a tree data structure. The internal nodes of the tree are used to represent the tests on the attributes while the corresponding branches denote the result of the tests [37]. The leaf nodes are the final outcome of the test and these are represented with the classes [5], [37]. The tree, based on each example in the training set, branches into nodes and finally into classes through the use of splitting criteria or attribute selection measure. The splitting rules determine the attribute on which the internal node of the tree will branch [5]. Thus, a decision tree may be said to be a mapping between the class label and the instance's attributes [5], [37]. These mappings are the models that are used for predicting incoming streams [37]. In incremental decision tree techniques, because all data cannot be stored in memory, a bound is usually used to establish when to branch [37]. The bound is used to determine the number of examples after which a split may occur. An example of such bound is the Hoeffding bound, which is used in the Hoeffding Tree (HT) classification algorithm [37].

The next classification algorithms to be discussed are those that fall under the Bayesian group. These algorithms make prediction using the Bayesian conditional probability [25]. These learning algorithms assume that the features of the instances are not dependent [25]. Bayesian classification methods use the training data to compute the Bayesian prediction, which is later used for predicting the labels of incoming stream. Naïve Bayes (NB) is an example of a Bayesian algorithm.

The last algorithm that we will be focusing on is the ensemble classification approach, which uses more than one base-classifiers [38], [39]. The classifiers are built using different samples of the training set [5], [38]. The models are then combined to predict the unseen instances. There is a final vote or averaging amongst the classifiers' prediction to determine the final prediction of newly arriving instances [5], [39]. The accuracy of the ensemble techniques is better than that of the base classifiers and accuracy is improved when models are built using varied samples [5], [39]. Bagging and Boosting are categories of ensembles or meta-learners. In Bagging, the final prediction is based on the vote with the majority. The votes are assumed to have equal weights. Boosting, on the other hand assigns weight to the training instances. The weights of the wrongly predicted instances are higher than those of the correctly predicted instances [5], [38], [39]. Weights are updated so that the models may concentrate on the difficult examples [5], [38], [39]. The final prediction in boosting is the averaging of the combination of the weighted predictions [5], [39]. Boosting is said to be able to generate more diverse models than bagging [39]. Example of ensembles methods in data stream mining are OzaBoost and OzaBag [38].

We discuss the techniques used for evaluating classification in data stream mining in the following subsection.

2.2.3 Evaluation of Learning in Data Stream Mining

Evaluation is used to test the performance of the model built on unseen data and to prevent over-fitting [5]. Evaluation aids in determining the best model that may be used for prediction [39]. In incremental learning, models are updated as data flows in [40]. The classifier in traditional classification is not updated, thus, it is not capable of predicting new instances that have different characteristics from the earlier data set that was used in building the model [26]. The evaluation methods used in the static environment are not applicable to the data stream environment. The reasons being that, in data stream, concepts cannot be learned at once because data are not stored, data distribution may change over time and multiple scans are not feasible [41]. Recall that in batch setting, the evaluation methods were said to be computationally intensive. Thus, in order to eliminate multiple scans in data stream classification, single holdout which is synonymous to holdout with one cross validation in static environment [5] is used [39].

Researchers are still working towards a set of standard evaluation methods for use in data stream classification [20], [39]. To date, two major approaches were implemented in data stream learning in [25], [39]. One of the methods divides every sample or chunk of examples encountered in the stream into training and test set [25], [39]. This type of evaluation is referred to as periodic holdout evaluation [39], [42]. As its name suggests, the size of data to be used for training and testing is specified. The instances seen in the stream are accumulated until they are equal to the total value specified as chunks D_i, as shown below in Figure 1. Di is separated into training data and test data based on the specified number of instances. Learning is done using the training instances and the model is tested with the training instances. Learning and testing continues periodically when the threshold is reached in this fashion across the stream. New models are always built per specified data size. Since the test and training instances are changed every time, there is no need to implement a forgetting mechanism. Forgetting mechanism is used to discard encountered instances in order to reduce data size or remove examples that are considered irrelevant for classification. Periodic holdout evaluation method is depicted in Figure 1 below. The disadvantage of this method is that not all the instances get the opportunity to influence the building of the model and also to be used in testing the model.

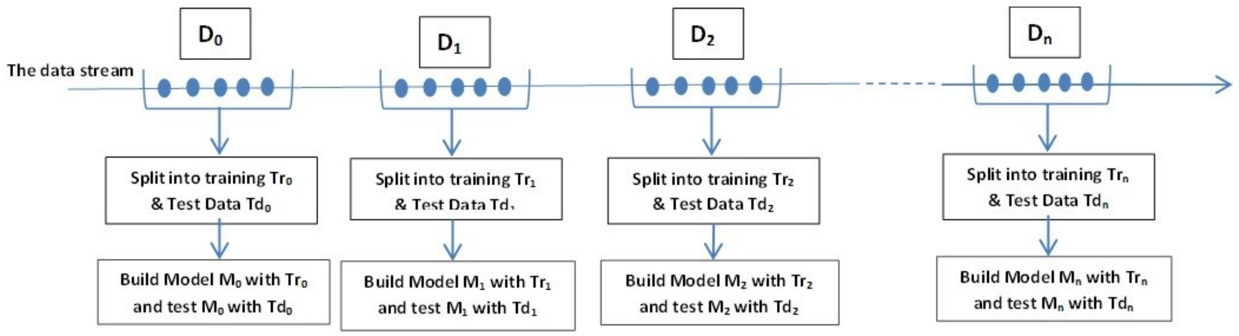
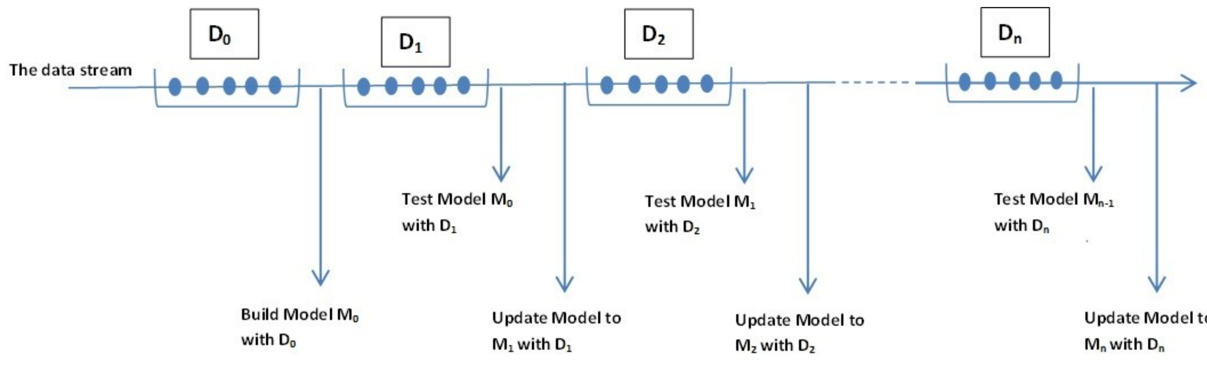

The data stream
D_0
D_1
D_2
D_n
Split into training Tr_0 & Test Data Td_0
Split into training Tr_1 & Test Data Td_1
Split into training Tr_2 & Test Data Td_2
Split into training Tr_n & Test Data Td_n
Build Model M_0 with Tr_0 and test M_0 with Td_0
Build Model M_1 with Tr_1 and test M_1 with Td_1
Build Model M_2 with Tr_2 and test M_2 with Td_2
Build Model M_n with Tr_n and test M_n with Td_n

Figure 1. Periodic Holdout Evaluation

In the other technique, "Interleaved-test-then-train" evaluation, rather than split examples seen into training and test sets, instances are first tested with the model and later used to train the learner in order to improve the prediction performance of the model [39]. Thus, testing is interleaved with training across the stream [25]. This may be done on an "instance-by-instance" or "chunk-by-chunk" basis [19]. The past examples are forgotten either by using sliding window or some other heuristics called fading factors, because instances are accumulated unlike in the periodic holdout evaluation where they are changed. Sliding window forgetting mechanism, forgets instances by using window of specified size while fading factor forgetting uses some heuristics to determine the instances to be forgotten. The advantage of this evaluation approach is that it makes use of all the data for testing and also because they are later trained, no individual example has significant influence on the model built [25], [39]. Interleaved-then-train evaluation method is also known as “prequential” evaluation technique [39]. Gama et al developed “prequential” that is "predictive sequential" evaluation that uses sliding windows or fading factors as its forgetting techniques [40]. Also in [19], an Interleaved-then-train evaluation technique which uses the “chunk-by-chunk” version was designed. We refer to this evaluation algorithm as INTER in this thesis. INTER is further discussed in the next section.

2.2.4 Overview of INTER

Recall that INTER is the benchmark evaluation algorithm used in this thesis. As mentioned in the section above, INTER is based on sliding window, where the data are accumulated until a certain threshold is reached [19], [25]. Initially, the first chunk is used to build the first model. Subsequently, recently seen chunk is first tested on the most recent model and afterwards used in updating the model. Therefore, as presented in Figure 2 below, window D_0 will be used to build the model M_0 that will be used to test window D_1. Likewise, chunk D_1 will be used to update the model to M_1, this will be used to test chunk D_2. This training and testing continues until the stream is exhausted. INTER is illustrated with the Figure 2 below. Any classification algorithm of choice may be applied in building and testing the model.



The data stream
D_0
D_1
D_2
D_n
Build Model M_0 with D_0
Test Model M_0 with D_1
Update Model to M_1 with D_1
Test Model M_1 with D_2
Update Model to M_2 with D_2
Test Model M_{n-1} with D_n
Update Model to M_n with D_n

Figure 2. Interleave-Test-then-Train Evaluation (Chunk-by-Chunk Version) [19]

Recall that evaluation is used in testing the performance of the models built, thus the choice of the classification algorithm used may have an influence on accuracy. Gao et al in [16] categorized the models built in stream learning as discriminative and "generative" models. Generative models are based on the assumption that a generalized data space exists for describing the class' distribution in a data stream. Discriminative models on the other hand, do not generalize but establish a distinct data boundary that clearly defines the data space of each class in the stream based only on its training set. Classifiers like decision trees build discriminative models while NB build generative models [16], [43]. For instance, NB updates its models based on the condition of Bayesian probability [44] thus it generalizes whereas decision trees maps instances seen to the tree. Hence decision trees rely heavily on the examples seen. Although discriminative models are expected to be better than generative models, unfortunately, the training instances that establishes clear boundary are difficult to obtain since prescient data distribution of the incoming stream are not available [16], [43]. Relying only on examples in recent window to build a descriptive model may result in low accuracy because the instances may be insufficient to build a discriminative model [43] that will be able to generalize on incoming streams. The incoming stream may comprise of newly introduced concepts.

In this section we discussed some pertinent aspects in stream mining to this project. In the next section we will discuss forgetting methods, where a subset of previously seen examples are discarded in data stream mining, in order to reduce the size of the training data.

2.3 Forgetting Techniques in Data Stream Learning

Recall that some earlier encountered instances in the stream are removed from the training set before building the models. This is done in order to improve latency in data stream mining. Forgetting as discussed, is used to achieve the task of choosing representative data by reducing the size of the training data and retaining only relevant concepts that will be related to incoming streams [37]. The instances that are forgotten must be instances that are no longer relevant to the task at hand. For example, in sliding window discussed under Section 2.1.1.2, older chunks are assumed to be irrelevant, thus they are discarded. But in actual fact, the forgetting mechanism used will depend on the type of task.

As an illustration, it is not all the time that it is only the recent chunk that is relevant to the incoming streams, examples in past chunks maybe relevant in predicting later chunks. In this thesis, for example, the sizes of instances of some classes in the recent window may be too small to build unbiased models. Therefore, minority instances in earlier chunks are accumulated and added to recent training sets in order to augment instances of those classes. The various types of forgetting approaches in literature can be grouped into window-based or fading factor-based. The window-based technique includes fixed sized windowing-based forgetting and adaptive windowing-based forgetting while the fading factor-based technique is also known as the weighing-based forgetting. They are further discussed below.

Fixed Sized Windowing-Based Forgetting: This type of forgetting mechanism is based on keeping a constant window size [43], [44]. The training instances or window becomes updated when the instance in incoming stream becomes equal to the size of the window. The challenge here is the size of the window to keep so that enough concepts may be learned. Maintaining a small window leads to frequent updating of the model and not learning sufficient concepts at a time, this may have impact on accuracy. Whereas large window has higher accuracy because the model is built using more examples, but the problem is the time taken to accumulate such a large window [43], [44].

The phenomenon of the optimal window size to maintain is referred to as “Stability-plasticity dilemma”. "Stability-plasticity dilemma" concept [44], [45] tries to balance the rate of updating a model with new knowledge and the rate of forgetting old knowledge. Plasticity refers to maintaining a smaller window which will enable us to learn new concepts quickly and forget old concepts faster [44]. The keeping of smaller windows will require new models to be built often [44]. Stability on the other hand, refers to the maintenance of a bigger window, which gives the allowance of accumulating more concepts within a window [44]. Higher plasticity results into high rate of forgetting likewise, higher stability requires longer time and it reduces the accuracy of the classifier in evolving streams [44].

Adaptive Windowing-Based Forgetting: In adaptive windowing-based forgetting, the window size is reduced when a change is detected or a condition occurs otherwise it is left to reach a certain threshold before the training instances are changed [44]. For instance, the imbalance ratio may be used to trigger when to update the model [46].

Weighing-based forgetting: Instances are assigned weights according to how relevant they are to incoming streams. Earlier examples are retained or forgotten when training instances are updated based on the weight assigned to them. The weight may be made to fade with time or based on how related instances in recent chunks are to incoming concepts [44].

Learners are made adaptive by updating the underlying training set used in building them [26]. Because training sets are selected using forgetting mechanism, the learner is relevant and resilient to predicting incoming stream. Therefore, the model is said to be adaptive.

We discussed how some of the previously seen instances are discarded and only relevant examples are left in order to reduce the size of the training set in this section. Building on the discussion above, we intend to use a pseudo-fixed windowing-based forgetting mechanism, where the training sets are updated after each window. Unlike the fixed-window based forgetting technique, the size of the training set is not the same as that of the specified window. Past examples in some earlier encountered windows are retained, thus making the size of the training set to be larger than the size of window specified. In the next section we discuss some insights.

2.4 Discussion

In this section, we discussed data stream mining and relevant concepts. In data stream mining, it is impossible to learn all concepts at once because data are not stored, thus the models must be updated. Also because we cannot accumulate all seen instances, we need to use forgetting mechanisms to retain only relevant training instances. We talked about data stream mining techniques, which are used to reduce the size of representative data. In addition, we discussed evaluation techniques in data stream mining which consider computational requirements and thus involve single scan. In this thesis, we will employ sliding window as the data stream mining technique with a pseudo window-based forgetting mechanism as mentioned above. Unlike the sliding window and window-based forgetting methods, we do not assume that only instances in the recent window are important. Thus, we accumulate some minority instances in past chunks since their occurrence is infrequent. "Interleave-test-then-train" evaluation is chosen as the evaluation method because models may be able to generalize better since they will be trained using almost all seen instances. We conclude this chapter in the next section.

2.5 Summary

In this chapter, we reviewed the concept of data stream mining. We discussed the properties and approaches that were used to extend algorithms in data mining to the stream setting. We also talked about forgetting mechanism which is a phenomenon that is synonymous to stream mining. In the next chapter we will be concentrating on multi-class imbalance learning, which is our main concern in this research. We will do a literature review on relevant topics and related approaches used in literature.

Chapter 3

Multi-Class Imbalance Classification

The ability of traditional classification algorithms to correctly predict the labels of minority instances becomes a challenging task in highly skewed data set. Intuitively, this is due to the fact that the classifier was not presented with sufficient examples from some classes in building the models that will be used in predicting unseen instances.

In this chapter we give a general view of the imbalanced data set classification problem in Section 3.1. Section 3.2 discusses the taxonomy of imbalanced data set learning based on the definition of the problem and the application domain. In Section 3.3, we elaborate on the possible factors that may make the classification of skewed data sets difficult. Section 3.4 presents the different categories of solutions to imbalanced data classification. Relevant algorithms such as SMOTE and SCUT are discussed alongside. In section 3.5, we review the literature on methodologies that are relevant to this thesis. We conclude the chapter by discussing the areas of application of multi-class imbalance data stream learning to real life.

3.1 Imbalanced Data Set Classification

The distribution of classes in real world data set is often not balanced, in that examples of some labels occur much frequently than others, thereby resulting in imbalanced data scenario. This may occur in both static and dynamic environment [36]. Imbalanced data sets can feature highly skewed data sets where the imbalance ratio of one class to another class may be as high as 1:

100, 1:1000 or 1:10,000 or more [7], [6], [18], [36], [47], [48]. Imbalance ratio, for example, is the ratio of the number of instances in one class to the number of instances in another class. An example of a multi-class data set is the yeast data set in the UCI repository [49]. The data set contains ten classes with different proportion of instances per class. The distribution of instances of the classes in the yeast data set is present in Table 3 below.

Table 3 Yeast Data set Description[49]

Class	No of Instance
CYT	463
NUC	429
MIT	244
ME3	163
ME2	51
ME1	44
EXC	37
VAC	30
POX	20
ERL	5

This situation is common in applications where some events rarely happens, such as in intrusion detection, medical diagnosis and fraud detection. As an illustration, in the yeast data set above, examples of the ERL, POX, VAC, EXC classes are infrequent compared to classes like CYT and NUC, thus the data set is said to be imbalanced. Imbalance can also occur artificially in an application if some instances are infrequent, for instance due to lack response from survey [50]. Similarly, in a stream which is analyzed in chunks, instances of some classes may not be frequent in a particular chunk, thereby making the chunk to be imbalanced. Class distribution has effect on accuracy and error rate [47] in classification, the recognition rates of the labels with low distribution are usually poor. These labels with infrequent examples are usually the classes of interest [41].

The goal in imbalance learning is to improve the prediction rates of the positive classes [7], since the minority instances are usually the examples we are keen in identifying correctly. Classifiers generally tend to be overwhelmed with the lopsided training sets that contain many examples from the majority classes and few from the minority labels [41], [47]. Apart from the small representation, other factors such as overlapping classes, noise, classes being defined by many feature spaces and data sets with multiple minority and majority classes may hinder standard classifiers from correctly predicting instances of these minority labels [41]. Hence, in order to solve these issues and achieve the aim of imbalance learning, different approaches have been proposed in literature.

The understanding of the problem domain and the creation of instances for the minority classes may aid in improving the recognition rate of the positive instances in imbalance learning. The system or the availability of examples of the minority classes may influence the taxonomy of imbalance learning [51]. Based on these, imbalanced classification may be grouped into one-class learning, binary-class learning or multi-class learning task. The method used in general to solve imbalance learning task may be at the data level (which involves sampling), algorithmic (this involves adjustment to the classification algorithm) or both [6]. Imbalance learning has been successfully applied in many fields.

Imbalance classification has been applied in traditional data mining to fault diagnosis, medical diagnosis, "e-mail foldering", face recognition and detection of oil spill [52]. Studies have increased in concept drift and imbalance classification problem but both research areas have been mostly studied separately [36]. Compared with the traditional learning, imbalanced stream learning has not received great attention. The categories of imbalance learning and approaches to solving imbalance classification are discussed in the next sections.

3.2 Taxonomy of Imbalanced Data Classification Tasks

Based on the approaches in literature to solving imbalanced data classification problems, both in static and streaming environment, the tasks may be grouped into three, namely one-class classification (OCC), binary-class classification (BCC) and multi-class classification (MCC). Each of these tasks may be dependent on the domain, data collection design [51] or availability of instances for some classes. For example, the system may require categorizing network quality into good and poor (this will be a BCC task) or into good, average and poor (this will be a MCC task), or good and others which will be modeled as an OCC task. The illustration above is domain dependent. The different learning tasks are further explained below.

3.2.1 One-Class Classification

In OCC, instances are not labeled as positive or negative. There is only one target class which is the class that is most represented [53]. Because instances of this target class are abundantly available and greatly out numbers examples of other instances, the model is built using these target data. The goal of OCC is to use the learner to define the boundary between the target instances and the others [54], [55]. The task in OCC may be to identify instances that fall outside the boundary which is called outlier detection, novelty detection or it may be to learn about the concept of the majority, which is referred to as concept learning [54] , [55]. OCC is good for highly imbalanced data in the presence of noise [6].

An example of research in static mining in which OCC was used is in [56], where Japkowicz built an auto-associator using neural network with only one example. The goal is for the auto-associator to differentiate between the concept it learned and others. The limitation of OCC is that it cannot be applied directly to some data mining algorithms, like the previously introduced

Naïve Bayes and Decision Trees, which are based on having more than one class [53] [55]. In addition, not all multi-class classification tasks may be modeled as OCC.

3.2.2 Binary-Class Classification

Recall that most research in both traditional and data stream mining fall into this category although there are many multi-class real-life classification problem [7], [8], [57]. The data set is seen as belonging to either the positive class or the negative class. Recall that in OCC, examples either belong to the only class, for example, the majority in novelty detection or it does not belong to it. In contrast, in binary-class learning, instances are categorized as belonging to either of two classes, the positive or the negative class. The task in two-class learning is to build a learner that will be able to successfully predict which of the two classes the test instances belong to. Figure 3 below is use to portray what two-class imbalanced data set looks like. The blue rectangle represents the positive class while the green portion is used to represent the negative class.

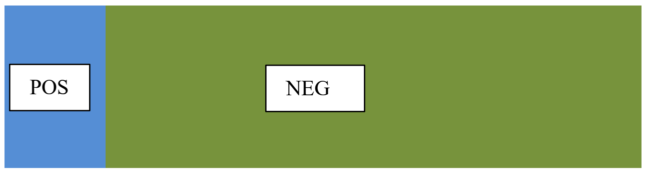


Figure 3 Two-class Data set Scenario

3.2.3 Multi-Class Classification

This approach is good for mining applications or systems where being specific is very critical, that is labels have a higher level of granularity. Examples of multi-class classification in traditional setting include protein fold and weld flaw [7]. Figure 4, below depicts a pictorial form of protein fold which is an example of a multi-class imbalanced data set. The different colors represent the labels present in the data set.

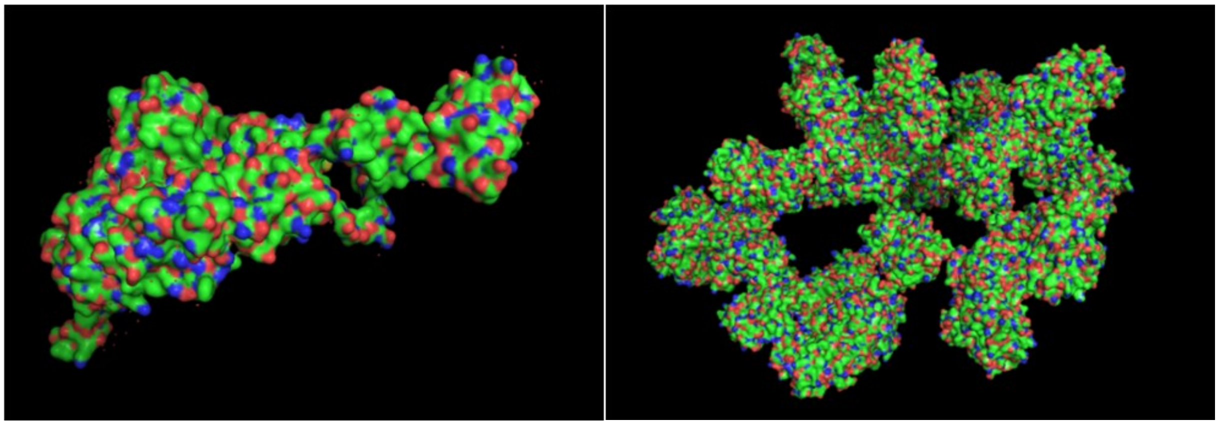


Figure 4. Proteins 1TYV and 1XFX from the P22 Tailspike and Calmodulin families [8].

Multi-class classification is used in predicting the labels of instances of a data set with more than two classes. The multi-class imbalanced data set classification problem in some studies, are decomposed into two-class classification [7], [8], [41], [57], [58], [59]. In order to model the MCC task as BCC task, classes in the data sets are merged using OVA or OVO and the decomposed data set is then presented to the classification algorithms. Examples of studies where multi-class imbalanced classification were reduced to two-class in imbalanced learning stream mining includes [16], [43], [60], [61], although in [43] it was said that the method may be easily extended to multi-class learning.

The downside of class reduction is that the classes cannot be studied at once [7], [57]. In addition, the imbalance ratio increases because the classes are combined and the predictions may be biased. Moreover, some areas may be left unlearned because the classifier was not presented with the real data distribution. [7], [57]. Interestingly, not all multi-class classification tasks may be learned effectively by reduction. For example, it is difficult in cost-sensitive learning, which is one of the techniques used in addressing imbalanced learning, to specify a combined cost-matrix for multiple classes for penalizing wrong predictions when classes are decomposed [7], [57]. In addition, the combinatory of the classes using OVA or OVO may become too cumbersome, if comprehensive combinations are done so as to arrive at a rich conclusion.

The authors in [61] argued that class reduction reduces the complexity of the problem because the decomposed data set comprises of lower number of classes. It makes the decision boundary of such data sets to be easier to separate and to identify classes [61]. This may not be true because of the higher imbalance ratio and the reduced data set may have more complex structure. The class that is being studied at the particular time may overlap with other classes. Consider the case in Figure 5 below where classes are represented using different shapes and colors. The three-class learning task is reduced to binary class learning using One Versus All (OVA). Assume that the minority label depicted with the blue circle is the class that is being studied at the moment; hence the minority class with the red crosses is merged with the majority class with the green stars. The classification problem is likely to be compounded because there is an increase in the imbalance ratio and the structure of the data space has become complex too. Unfortunately, the minority class of interest is not well separated from the majority class.

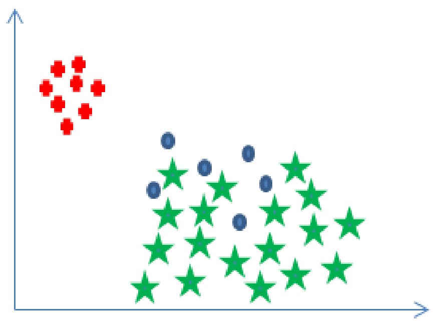


Figure 5. Issue with Class Reduction.

Wang and Yao concluded that class reduction is not necessary, because learning via reduction is not be equivalent to direct learning without class decomposition [7]. It was proven in [7] that class reduction does not aid multi-class imbalance learning [57].

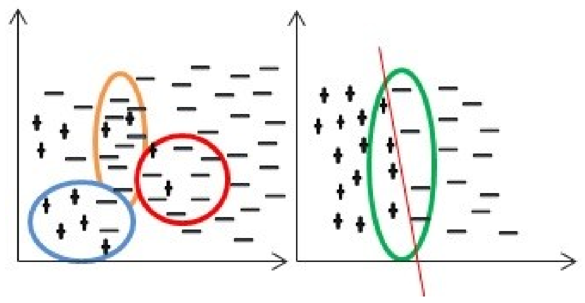
There are few studies in literature that are modeled as MCC in data stream mining, Some of them include MINAS developed by Ribeiro de Faria et al to detect novelty in multi-class data stream [62]. Another example is [57], where cost-sensitive boosting was applied to a multi-class imbalanced data stream without decomposing the data set. Although the multi-class learning technique proposed by Agrawal et al in [8] was in static environment, it is suitable to be extended to the streaming setting and we follow this research in this thesis. The next session discusses the conditions that make imbalanced data set learning difficult using standard classifiers.

3.3 Factors That Makes Learning Difficult

There are several factors that make learning ineffective. The first is the proportion of the classes' instances in the data set, this is referred to as between-class imbalance [8], [53]. Imbalanced data are not difficult to learn simply because of fewer examples alone. Other factors such as overlapping of concepts, small disjuncts and noise affect the ability to build a good model that has high recognition rate of the minority labels [8]. These factors are further elaborated below.

3.3.1 Small Class Representation

Positive examples are under-represented in imbalanced data sets. The classification algorithms are presented with barely enough examples of the minority classes in order to build the models; this means that sufficient concepts about the minority classes are not learned. Thus, the learners' ability to predict accurately instances of the minority classes is low. In [7], Wang and Yao discovered that the performance of classifiers decreases as imbalance ratio increases, i.e. the highly skewed the data set is, the lower the performance of the classifier. Therefore, learners built using few examples make the testing phase difficult [47], [52] because the range of examples used for building the model is not large.



(a) (b)

Figure 6. Imbalanced Data set (a) versus Balanced Data set (b).

Figure 6 above is used to visualize examples of data sets with imbalanced and balanced instances. Figure 6 shows the scenario where the class boundary of the labels may be either well separated as shown in Figure 6(b) or not as shown in Figure 6(a). Figure 6(a) above, depicts the situation where the positive class represented by the + sign is under-represented and the negative class represented by the – sign is well represented, while Figure 6(b) shows the two classes have equal representation. Because the negative examples are more than the positive examples, the model, during training will learn more about the majority class than the minority class. When it comes to prediction, because the model is built using more varied examples of the majority labels which spread across almost all the data regions, it is most likely to classify positive instances as belonging to the negative class. The condition of a class not being well represented like the other class is referred to as between-class imbalance [8]. Between-class imbalance is solved by selecting training samples with equal number of instances from the minority and majority classes [15].

3.3.2 Inseparable or Overlapping Classes

Overlapping of classes is another factor that may make learning difficult. In overlapping, the class boundary does not show sharp demarcation between the classes present [52]. The ability to distinguish between overlapping classes becomes hard, because the data space will be described by the model as belonging to the class that is most represented [15]. Therefore, the minority will be classified as belonging to the majority in the data space [47], [52]. Using Figure 6 above to illustrate this phenomenon, in Figure 6(a), the orange circle shows when a clear boundary cannot be drawn between the two classes, while the green circle in Figure 6(b), shows when a demarcation may be made between the two labels using the red line.

In [47], different combinations of sampling techniques were combined and it was discovered that class overlapping is a threat to imbalanced learning. Algorithms such as decision tree, neural network, Multi-Layer Perceptrons (MLP) are sensitive to imbalanced training sets [53], [63]. Decision trees, as an example, usually have problem with class overlap [47]. Pruning is the technique that is used for trimming leaf nodes with fewer examples in order to prevent model over-fitting. Pruning will results into categorizing some sub-concepts that would have fit into the pruned leaves as sub-concepts of the leaves with more examples.

3.3.3 Small Disjuncts

A small disjunct occurs when a class is represented with more than one sub-concepts it is also referred to as within-class imbalance [6], [8], [53], and it may also make learning difficult. Small disjuncts may be visualized in 2D as the situation where a particular class spreads and form clusters which represents sub-concepts. This makes it difficult for learners to recognize instances that actually owns the data space if they have poor representation in a particular cluster [52]. The minority class may have very little data representation in some of these subareas, thus making it difficult for the learner to learn the characteristics of the minority classes in the data space hence classifying them as belonging to the majority in the cluster.

The blue, orange and red circles in Figure 6(a) are sub-concepts belonging to both labels identified in the data set. To illustrate why it may be difficult to learn when classes in a data set are defined by more than one sub-concept, assume that we train a learner with the data set in Figure 6(a). If we later test the model we built with a test instance that is located in the data space represented by the red circle in Figure 6(a) above. The instance is very likely to be predicted as belonging to the negative class because the model during training was presented with more instances of the negative class from this region. But, if it were to be a test data from the blue circle, it has a higher likelihood of being classified as a positive instance. In order to address within-class imbalance, it was suggested in [15], [64] to use clustering to identify the sub-concepts within the class [15], [63].

In order to solve the class imbalance classification problem, the challenges which includes small representation of classes, overlapping between classes and small disjunct in imbalanced data set must be addressed [63]. Another notable factor that may affect class imbalance classification is the effect of the choice of classification algorithms on the imbalanced data set, this may vary. For instance, Batista et al, [47] found that batch decision trees are more affected by imbalance class distribution than Naïve Bayes, because of pruning.

Other challenges to learning with imbalanced data sets include noise. Noise causes distortion and increases difficulty in learning [53]. Data cleaning are found to be effective for extremely imbalanced data set with noise [47]. Data cleaning methods such as Tomek link and Edited Nearest Neighbour (ENN) Rule may be used for removing noisy examples [47]. They are also effective in the removal of instances that fall on the class boundary, thus enhancing the learning ability of the models built because noise are excluded from the training sets[47].

In addition, in streaming environment, the inherent characteristics of the stream may also pose some challenges. For instance, because of storage and computational performance not all the encountered positive instances may be stored to increase their proportion in the training set [16]. Due to this, treating an example as a noise in a chunk may be wrong; it may be a new concept that is infrequent in the chunk. Categorizing an example as a noise or new concept must be put into consideration when trying to balance plasticity and stability so that sufficient concepts are learned in order to build meaningful models.

Some of the methods proposed for solving imbalanced data set classification such as random oversampling and random under-sampling do not consider some the issues that make learning difficult. For instance, random oversampling does not consider small disjuncts and overlapping in classes. It randomly replicates previous examples; likewise random under-sampling removes instances without the use of heuristics. The instances removed may be a new concept that is yet to be learned [63]. Although pruning may be used to eliminate small disjuncts, it will result into the removal of the leaf nodes with lower than a certain number of instances [63]. This approach may be ineffective because it will leave the areas that were pruned to remain unlearned, therefore cluster-based oversampling is proposed for eliminating small disjuncts [63].

3.4 Techniques for Imbalanced Data Classification

The solution proposed for class imbalance learning in literature may be grouped into three, namely sampling, algorithmic and hybrid [52], [57], [65], [66]. The sampling approach tries to improve on the training set while the algorithmic approach creates or modifies existing learning algorithms using some lopsided criteria that improves the recognition rate of the minority class. Hybrid combines both methods. The advantage of sampling over the algorithmic approach is that no knowledge is required about the domain because weight or cost is not involved, therefore it has wider applicability and flexibility [52], [65], [66]. The taxonomy of the approaches used in addressing imbalanced data sets in literature is shown in Figure 7 below.

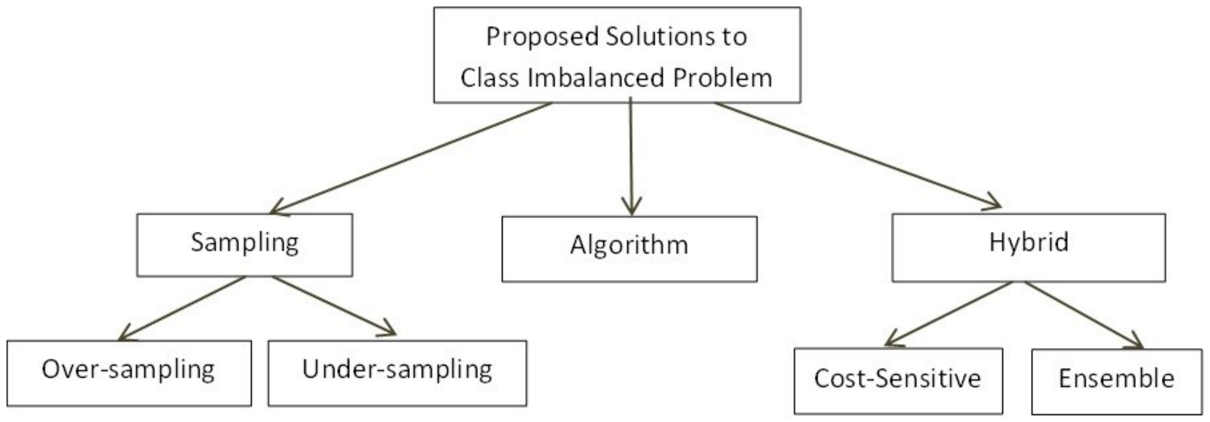


Figure 7. Taxonomy of the Proposed Approaches in Literature for Solving Imbalanced Classification Problem.

The methods used in imbalanced data sets classification are explained in the following sections.

3.4.1 Sampling or Data Techniques

The data approach is the most used technique for solving imbalanced data set classification [16]. Sampling may be applied generally to all data sets, it is not domain specific [67]. In sampling, data preprocessing approaches are used for selecting balanced instances across all classes in the training data, data may be added, removed or both, [15], [48], [52]. The addition or removal of instances can be done randomly or by using some heuristics. Thus sampling may be categorized as random under-sampling, random over-sampling, “informed over-sampling”, “informed under-sampling” and hybrid sampling. Sampling increases the prediction accuracy of the minority classes. Area under Curve (AUC) is used for visualizing the performance of classification algorithms. In [60], Weiss and Provost determined the impact of class distribution on the classifiers’ performance using AUC and found that balanced training data usually outperforms the unbalanced training sets. All sampling approaches have known advantages and drawbacks [8]. The various sampling methods are explained below.

3.4.1.1 Random Under-sampling

Random under-sampling involves the removal of some instances from the negative classes in order to balance its representation in the training set with the minority labels [47]. Under-sampling generally have been found to lead to loss of critical information [15], [52], [67], [68] although the computational time is improved because the data size has been reduced [68]. Random under-sampling reduces the example space and results into loss of information [8], this may lead to the removal of clusters which are sub-concepts, leaving those sub-concepts unlearned [15], [41]. Highly imbalanced data set may cause too much information loss. Under-sampling may also be used to speed up computational time and to reduce memory usage [53].

3.4.1.2 Random Oversampling

In random oversampling, instances of the positive class are replicated so that their representation matches that of the negative class [47], [68]. Random oversampling, because data are only duplicated may confuse the classifier [6] and no new concepts are learned. Over-sampling causes over-fitting [15], [7], [8], [52], [68] and it increases the processing time because the size of the training data have been increased [8], [68]. In [18], random over-sampling was found to perform worse when compared with random under-sampling. But according to Batista et al, oversampling is said to be more effective than under-sampling because oversampling does not result into information loss [47]. Figure 8 below gives a visual representation of random oversampling and random under-sampling.

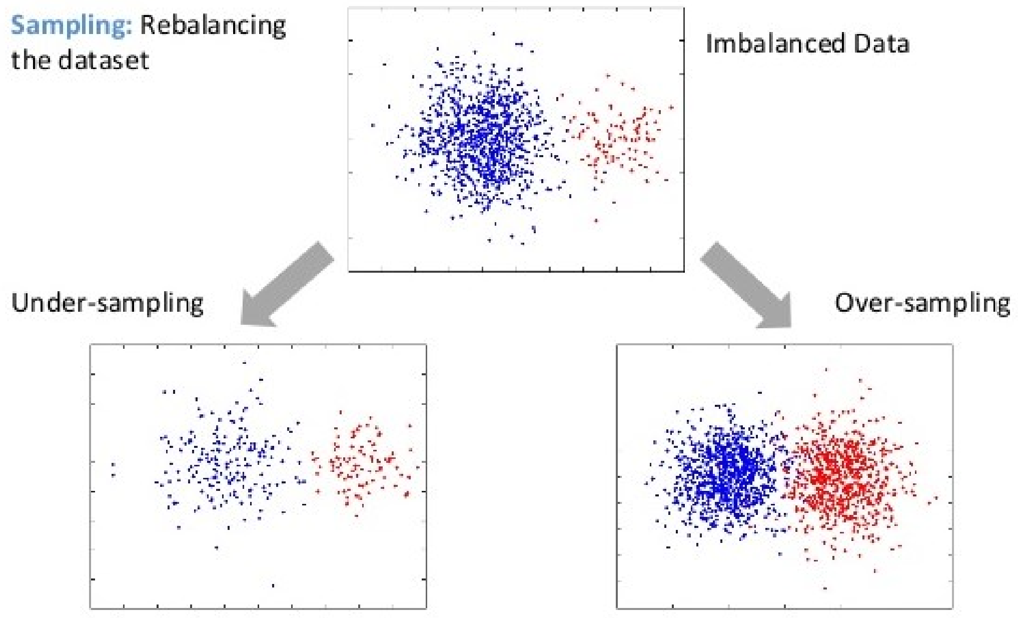


Figure 8. Sampling (Under-sampling and Oversampling) in Two Class Imbalance Data set [69]**.**

3.4.1.3 Systematic / Informed Under-sampling

Random sampling is done without considering within-class imbalance, it only concentrates on between-class imbalance [64]. Therefore, unlike random under-sampling where instances are blindly removed, “systemic under-sampling” uses techniques that preserve all the data region in the classes [41]. Informed under-sampling is used to solve the problem of small disjuncts and overlapping classes. Informed under-sampling approach may be used to identify class boundaries or sub-concepts. The identification of class boundaries using rules such as nearest neighbor aids in the discovery of overlapping data and noisy data. Data cleaning tools such as Tomek link, Nearest Neighbour Rule (NNR), Condensed Nearest Neighbour Rule (CNNR) may be used for under-sampling [47].

Sub-concepts may be discovered through the use of clustering algorithms [59], [70]. Its use as a systematic under-sampling technique is increasing in literature [59]. Under-sampling is done based on the number of clusters discovered. This reduces the probability of eliminating a sub-cluster thereby preventing the loss of information about a sub-concept [8], [41], [70]. The decision after clustering may be to select representatives from each of the discovered groups. Cluster-based under-sampling was used in [15], [8], [59], [64], [70].

Different varieties of cluster-based under-sampling have been proposed in literature. The authors in [64], developed an approach, Principal Direction Divisive Partitioning (PDDP) for clustering to discover sub-concepts within a class. The resampled method took into consideration examples from these clusters. Nickerson et al, found that there was no difference between when the data was not resampled and when it was blindly resampled. Their approach was found to be effective [64].

In [70], cluster-based under-sampling was applied on the data set as a whole. The clusters were generalized based on the idea that the most represented instance naturally owns the data space. Thus a data space is under-sampled by selecting instances from the majority in order to balance with the minority in the cluster [70].

The idea of cluster-based under-sampling used in SCUT [8] differs from the two above, each of the classes in the data set are clustered individually unlike the approach in [70]. In order to arrive at the final resampled data set for the majority labels, examples where chosen from each of these clusters up to the average of instances in the data set. The reasoning behind this was to prevent loss of information about any sub-concept, and also ensuring that all concepts are learned. SCUT also prevented the drawback involved with too much oversampling in highly imbalanced data sets. The idea of cluster-based under-sampling maybe visualized with Figure 9.

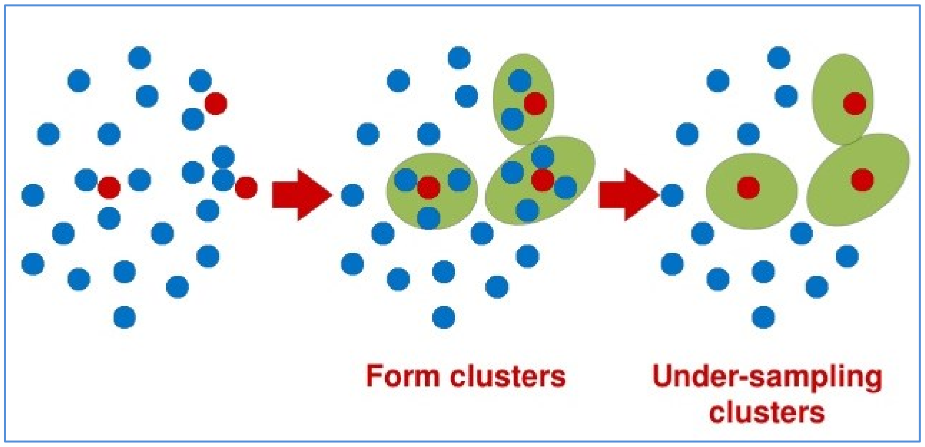


Figure 9. Cluster-based Under-sampling [71].

3.4.1.4 Systematic Oversampling

Informed oversampling increases the presence of the minority instances similar to random oversampling. Unlike random oversampling, minority instances are not replicated, synthetic instances may be generated or past examples may be used to balance the presence of the minority class with that of the majority class [6], [52]. SMOTE for example, may be used to generate synthetic instances using the nearest neighborhood of an existing minority instances [8]. This increases the number of examples in the minority-class' data space. The basis used in generating the synthetic instances is that, instances that are in the same neighborhood will occupy the same data region, hence they belong to the same class [8]. Systematic oversampling may ameliorate the issues with small disjuncts.

One particular interesting systematic oversampling approach that is pertinent to this thesis is SMOTE [18]. Chawla et al, in [18], produced synthetic instances instead of replicating the original data. The generated synthetic examples are nearest neighbor of randomly selected existing data, the number of instances generated depends on the over-sampling rate [18]. Feature space represents the region covered by the attributes, while data space is the region covered by the data involved. SMOTE generates synthetic instances based on "feature space" rather than the "data space" [18]. Synthetic examples are generated based on how similar they are to the attributes of existing instances being considered and its nearest neighbors [18]. Figure 10 below uses an example to explains how synthetic examples are generated using SMOTE. Because the data space has been extended based on the feature space, the classifier tends to be able to generalize better [18] and "hidden" minority data space that are very rare to find may be created [18], [72].

Informed oversampling may overgeneralize and generate instances that may aggravate the problem of overlapping and indistinct boundary [47], [70]. Data cleaning tools may be used to remove such examples [47]. As an illustration, SMOTE was combined with Tomek Link as the data cleaner in [47], because SMOTE may generate instances that may cause the classes to overlap. It was found that the combination of oversampling with data cleaning tools is effective [47]. Other examples of non-random sampling that uses synthetic examples are Adaptive Synthetic Sampling (ADASYN) [48], Border-line-SMOTE [41]. Similar to random oversampling, systematic oversampling may increase computational time if too much examples are introduced [6], [47].

Consider a sample (6,4) and let (4,3) be its nearest neighbor.
(6,4) is the sample for which k-nearest neighbors are being identified.
(4,3) is one of its k-nearest neighbors.
Let:
f1_1 = 6 f2_1 = 4 f2_1 - f1_1 = -2
f1_2 = 4 f2_2 =3 f2_2 - f1_2 = -1
The new samples will be generated as
(f1',f2') = (6,4) + rand(0-1)* (-2,-1)
rand(0-1) generates a random number between 0 and 1.

Figure 10. How Examples are Generated using SMOTE [18].

In [18], it was discovered that the combination of SMOTE-based oversampling with under-sampling is more effective in addressing imbalanced data sets than using only under-sampling [18]. The non-improvement in under-sampling and random oversampling was attributed to the fact that both methods do not extend the decision space of the classes in the data set [18]. Recall that, SMOTE expands the decision boundary of the minority examples [18]. Although one shortcoming of SMOTE as noted by Lopez et al, is the generation of instances without considering overlapping with other classes and crossing boundary [67]. The SMOTE algorithm is presented in Algorithm 1.

```
Algorithm SMOTE(T,N,k)
Input: Number of minority class samples T: Amount of SMOTE N%: Number of nearest
neighbors k
Output: (N/100) * T synthetic minority class samples
1. (* If N is less than 100%, randomize the minority class samples as only a random percent of
them will be SMOTEd. *)
2. if N < 100
3.        then Randomize the T minority class samples
4.                T = (N/100) * T
5.                N = 100
6. end if
7. N = (int)(N/100) (* The amount of SMOTE is assumed to be in integral multiples of 100. *)
8. k = Number of nearest neighbors
9. numattrs = Number of atrributes
10. Sample[][]: array for originsl minority class samples
11. newindex: keeps a count of number of synthetic samples generated, initialized to 0
12. Synthetic[][]: array for synthetic samples
        (* Compute k nearest neighbors for each minority class sample only. *)
13. for i - 1 to T
14.       Compute k nearest neighbors for i, and save the indices in the nnarray
15.       Populate(N, i, nnarray)
16.       end for
          Populate(N, i, nnarray) (* Function to generate the synthetic samples. *)
17. while N <> 0
18                        Choose a random number between 1 and k, call it nn. This step chooses
                  one of the k nearest neighbors of i.
19.               for attr - 1 to numattrs
20.                       Compute: dif = Sample[nnarray[nn]][attr] - Sample[i][attr]
21.                       Compute: gap = random number between 0 and 1
22.                       Synthetic[newindex][attr] = Sample[i][attr] + gap * dif
23.               end for
24.               newindex++
25.               N = N -1
26. endwhile
27. return (* End of Populate. *)
          End of Pseudo-code.
```

Algorithm 1. SMOTE Algorithm [18]

Most sampling techniques used in imbalanced data set classification combine two or more of the approaches discussed above. The essence of this is to alleviate the effect of the disadvantages of the underlying methods while benefitting from their advantages [8], [18]. SCUT prevents the excessive use of one method over the other thereby reducing the effect of the disadvantages of both methods. [8]. The goal in sampling is to obtain a balanced representative training set [66]. In determining the best proportion for resampling, it was discovered, by Estabrooks et al, that the optimal resampling proportion varies, an optimal result may not be obtained by oversampling only using the number of instances of the majority as the resampling proportion [66]. Although different suggestions are made about the best sampling rate [65], deciding on the sampling rate is non-trivial because the optimal resampling proportion cannot be predicted and the number of classes are many in multi-class imbalanced data sets. SCUT, the work that is being followed in this thesis is presented next.

3.4.1.4.1 Overview of SCUT

SCUT [8] brilliantly combines informed under-sampling with informed oversampling in order to address within-class and between-class imbalance in multi-class data set without decomposing the data set. The data set is split into sub-classes using their labels, SMOTE is applied afterwards to each of the minority sub-class to generate synthetic instances which may result in the discovery of a new data space [18]. The application of cluster-based under-sampling on the majority classes removes within-class imbalance and ensures that all sub-concepts are learned, because similar instances are grouped into the same clusters by clustering algorithms. The clustering algorithm used is EM, (EM was mentioned in Section 2.1 as one of the developed clustering algorithms). The reason for selecting EM is that the algorithm may discover the natural clusters on its own thereby removing the need to specify the number of clusters to group the data into [8]. The combination of the two techniques addresses the between-class imbalance and within-class imbalance issue. The algorithm for SCUT is presented in Algorithm 2.

```
Algorithm SCUT
Input: Data set D with n classes
Output: Data set D' with all classes having m instances, where m is the mean number of
instances of all classes.
1. Split D into $D_1, D_2, D_3, \ldots, D_n$ where $D_i$ is a single class
2. Calculate m
Undersampling:
3. For each $D_i$,  i = 1,2, …, n where number of instances > m
4.      Cluster $D_1$ using EM algorithm
5.      For each cluster $C_1$,  i = 1,2, …., k
6.          Randomly select instances from $C_i$
7.          Add selected instances to $C_i$'
8.      End for
9.      C = Ø
10.     For i = 1,2, …, k
11.         C = C U $C_i$'
12.     End for
13.     $D_i$' = C
14. End for
Oversampling:
15. For each $D_{i,}$  i = 1,2, …, n where number of instances < m
16.     Apply SMOTE on $D_i$  to get $D_i$'
17. End for
18. For each $D_i$, i = 1,2, …, n where number of instances = m
19. $D_i$' = $D_i$
20. For i = 1,2, …, n
21.     D' = D' U $D_i$'
22. End for
23. Return D'
```

Algorithm 2. SCUT Algorithm [8].

Multi-class imbalanced data set may suffer from highly imbalanced ratio; hence the use of under-sampling only may result in the loss of a considerable amount of information. This will increase the misclassifying rates of the majority class. Deciding on the minority class to use for balancing in multi-class imbalanced data set is non-trivial [73]. Therefore, SCUT reduces the amount of information loss and it simplifies the decision about the resampling distribution by using the average of all instances in the data set.

SCUT is effective in increasing the correct prediction rates of the minority labels through the introduction of synthetic instances. It avoids high increase in computational time and resources by combing under-sampling with oversampling in order to prevent excessive oversampling [8]. Moreover, SCUT is a flexible algorithm because it uses sampling.

3.4.2 Algorithmic Techniques

Algorithmic approaches to solving imbalanced data set classification involve the creation or modification of an algorithm [73] that is biased towards the minority and thus increases its effectiveness in correctly predicting the minority classes [67]. For instance, the condition for splitting or pruning may be modified in order to favor the minority labels in decision trees [73], [74]. Also, in SVM, the kernel or activation may be adjusted so that the class boundary is shifted to favor the minority class [65], [73]. The algorithmic methods are not easy to implement compared to sampling.

3.4.3 Hybrid Techniques

Another group of solution for solving imbalanced data classification in literature is the combination of the data and the algorithmic approaches. These techniques improved on existing ones by introducing varied cost-sensitive method or ensembles.

The first approach, cost-sensitive learning assumes that misclassification cost is not the same for the classes in a data set [11]. Instances of the minority classes are considered to be more sensitive compared to the others because they are the classes of interest [11], [41]. Cost-sensitive classification assumes that misclassifying instances should be penalized based on how costly the misclassification error is. For example, in a situation where a “poor” network service is predicted as being “good”, no action is carried out because the network is assumed to be good. This is a serious error because it may lead to loss of customers. Therefore, such error should be highly penalized compared to when a “poor” network service is misclassified as being “fair”. Because when the network is classified as fair, efforts are still applied in order to improve the network performance. Cost matrices are used to specify the cost that is attached to each type of classification error. Thus, cost sensitive learning requires domain knowledge [63], [65] but various heuristics are been used in literature to penalize classifier for wrong prediction [11], [67].

Cost-sensitive learning may be applied in three ways [41]. In the first method, weight may be assigned to instances based on the sensitivity of the instances [11]. Secondly, cost matrix may be combined with ensembles to minimize misclassification cost. The first and the second technique may be combined too. Lastly, the penalties for the misclassification errors may be incorporated into existing classification algorithms such as decision trees and neural networks.

The goal of the cost-sensitive method is to build models that reduces the total cost of misclassification [11], [41]. The first disadvantage of cost-sensitive learning is that they are not flexible when compared with the sampling techniques, because domain knowledge is required in order to build a cost matrix [63], The other downside is that the cost-matrix may be biased and may causing model over-fitting [53], [70]. Cost-sensitive learning may be used in fields such as medical diagnosis, fraud detection and intrusion detection [53]. Some studies reported cost-sensitive approach to be more effective than sampling and suggested extending the cost-matrix to multi-class imbalanced data sets, it may not be as flexible as stated in [41]. The cost-matrix may be biased thereby making the model to generalize poorly on incoming streams.

The second approach under the hybrid category is ensemble techniques, as introduced in Section 2.2.2. It is may be used as cost-sensitive ensemble or sampling ensemble [52], [65]. The use of more than one classifiers in ensembles is believed to improve the classification algorithms' effectiveness [16], [52], [53], [65]. The combination of cost-sensitive learning with ensembles [11] results in an algorithmic approach [67] while the combination of sampling with ensembles [67] is a sampling approach to imbalanced data set classification. In [15], using an ensemble on its own is found to be ineffective in addressing imbalance data set classification, because its performance depends on the base-classifiers. Thus, it was suggested to resample the training set in order to improve accuracy [15]. The "ensemble and sample" approach is not domain specific, unlike its algorithm counterpart [52].

Ensemble imbalanced learning in literature, although varied, is based on either Bagging or Boosting as introduced in Section 2.2.2. Also, the majority of these studies both in static and stream mining environment, focus on the binary classification [52]. Examples of ensemble approaches to solving imbalanced data set classification include SMOTEBoost [17], DataBoost-IM [10], BEV [75], SMOTEBagging [76], ESOS-ELM [65], [77] amongst others. The flexibility and range of applicability of the hybrid methods depend on whether domain knowledge is required. All the methods above have been found to be effective for imbalanced data sets. Another techniques used in literature to solve imbalance problem is active learning or so-called user-in-the-loop [41], [53], [78].

Sampling, although it is flexible, faces the challenges of choosing optimal resampling distribution, information loss in under-sampling and over-fitting with oversampling [73]. In the next section, we discuss some sampling approaches in literature to imbalanced stream classification.

3.5 Sampling Approaches to Imbalanced Stream Learning

In the section above, we discussed methods that are used to address class imbalance. In this section, we focus on some sampling approaches to imbalanced data stream classification. A number of methods suggested using all the minority instances seen so far, while in others, minority examples from previous chunks were propagated into the training set using either weights or similarity between minority instances in the current chunk. Discarding past minority examples may be unwise because their occurrence is infrequent [79]. The majority instances may be under-sampled or left untouched.

According to Ditzler et al in [80], SMOTE was incorporated into an existing algorithm, Learn++.NSE which is designed for evolving data stream. This was done in order to make the algorithm, Learn++.SMOTE to be sensitive to imbalanced data streams. SMOTE was used for generating the synthetic examples that were used to balance the training set. The methodology was evaluated with ensembles algorithms and it was discovered that the introduction of SMOTE improved the algorithm's effectiveness in the classification of highly imbalanced data set [80]. The data set used for the experiment are binary in nature [80].

In Selectively Recursive Approach towards imbalanced stream data mining (SERA) [81], Chen and He developed a technique for solving imbalanced data stream classification. The method involves the use of some minority instances from previously encountered chunks and those in the recent chunk in the training data. The minority instances in the past chunks were selected based on the Mahalanobis distance between these instances and those in the recent chunk [81]. The calculated distances were ordered. Hence, the minority training sets composed of minority instances from previous chunks that have high similarity measure with the recent chunk and the minority instances in the recent chunk. The justification for this approach is that the use of synthetic instances that are generated using only the recent chunk may cause a disconnection between previously learned concepts and may lead to “catastrophic forgetting” [81]. In addition it may prevent the model from being able to generalize on an incoming stream.

The approach in SERA assumes that the accumulation of the entire past minority instances will not overwhelm the system. The storing of all the encountered minority instances in the stream may not be feasible because data streams are assumed to be continuous. Hence, the storage and computational requirement for processing and selecting these instances may overburden the system.

The technique used in Multiple Selectively Recursive Approach towards imbalanced stream data mining (MuSERA) [82] is similar to SERA [81]. The difference between SERA and MuSERA, is that MuSERA uses weighted ensembles of classifiers for prediction. In SERA, the algorithm was evaluated using binary imbalanced data set, while the type of data set used in MuSERA was not disclosed.

The methodology in [13] addressed multi-class imbalanced data stream without class decomposition. One of the techniques used in [13] combines OzaBag with over-sampling while the other technique combines OzaBag with under-sampling. The approaches used in over-sampling and under-sampling were never defined. Recall and G-mean, which are evaluation metrics for imbalanced data set, were used in measuring accuracy. The designed methodologies in [13] were reported to be effective. In [13] the sampling techniques were not combined. The approach with under-sampling may suffer from a considerable loss of information when the data set is highly imbalanced [13]. Likewise, the method with oversampling may result in high increase in computational time and resources if the data set is highly imbalanced.

In [16] and [43], all the positive examples seen so far in the stream were accumulated and the majority examples were under-sampled. The accumulated minority instances were used to boost the numbers of instances belonging to the minority classes in the training set. Similar to the assumption in SERA, the positive examples were assumed not to be too many, so accumulating them will not overwhelm the system. This algorithm was evaluated with binary imbalanced data stream.

In Dynamic Feature Group Weighting with Importance Sampling (DFGW-IS) [9], the minority instances were accumulated using a specified window size and selected to the training set using a weighing factor that is based on the similarity between the instance and those in the recent chunk. The majority instances in the current chunk are under-sampled. The experiment was performed as binary-class learning [9].

Unlike the approaches above, which are based on the fulfillment of a specified window size, Weighted Online Sequential Extreme Learning Machine (WOS-ELM) may automatically update a model using imbalance ratio [12]. This enables the models to be updated if the imbalance ratio falls below a certain threshold, instead of waiting for the window to be of a particular chunk size. The algorithm used in this study was said to be computationally effective and to have high recognition rate for the positive instances [12]. The method is a cost-sensitive approach to imbalance learning [12], hence the need to assign weight. The algorithm was evaluated using binary imbalanced data stream [12].

Meta-cognitive Extreme Learning Machine (McELM) [83] uses neural networks for selecting and keeping relevant instances for learning in imbalanced data stream. Neural networks are the categories of classification algorithms that build classifiers using network of hidden layers of neurons. McELM was tested on both binary and multi-class data sets [83]. The algorithm was evaluated using the average of the ratio of correctly predicted instances in each class to the total number of instances in each class. The second evaluation metrics used is the overall classification accuracy. This is defined as the total number of correctly classified instances across all classes in the dataset to the overall total number of instances in the data set. The shortcomings of these evaluation metrics are that the performance of the majority classes may over-shadow that of the minority classes because of their proportion in the dataset. The other drawback is that McELM was reported not to be good for highly imbalanced data set in [84].

Voting-based Weighted Online Sequential Extreme Learning Machine (VWOS-ELM) algorithm [84] was designed for multi-class imbalanced data stream. It a cost-based ensemble classifier that extends the cost-matrix in WOS-ELM [12] and uses the ensembles of WOS-ELM classifiers [13] for multi-class imbalanced data stream learning. Its performance was compared to some binary imbalanced data stream algorithms such as WOS-ELM [84]. The disadvantage of this method is the need to compute a cost-matrix which as discussed earlier makes the algorithm not to be flexible because domain knowledge may be required and it may reduce its range of applicability.

3.6 Discussion

In imbalanced settings, accumulation of past minority instances in the data streams using some heuristics may be necessary because the minority instances occur less frequently. In addition, since the data stream in consideration is a non-evolving stream, previously encountered minority examples may be relevant in the prediction of incoming streams. The accumulation of the minority instances was explored in [16] and [43] and it was concluded that these approaches aided in the correct prediction of the minority instances in the imbalanced data stream. The disadvantage of the methods that accumulated the minority instance was that keeping of all the minority instances seen so far, may negatively affect system performance, as mentioned in Section 3.5. Hence, there is a need to retain minority examples with manageable size that will be relevant in predicting incoming streams. Earlier in this thesis in Chapter 2, we discussed reducing the size of representative data in data stream mining in order to reduce computational time and memory requirements. The concept of forgetting mechanism and cluster-based under-sampling may help to reduce the size and to select important examples.

Updating models using resampled training sets that are balanced across all classes present in the stream, may be effective in multi-class imbalanced classification. Recall that one of the reasons why the recognition rates of standard classifiers are poor with the minority instances is because it was not presented with enough examples of the minority classes. Moreover, retraining the models aids to improve recognition rate since some new concept may be arriving in the recent chunk for the first time [43]. All concepts cannot necessarily be learned at once in data stream mining, simply because the data may exceed storage capabilities. In the next section we will discuss some areas where multi-class imbalanced learning may be applicable.

3.7 Application of Classification of Imbalanced Data in Data Stream Mining

The possible areas of application of multi-class imbalanced data stream classification in real life are discussed in this section. Stream applications are generally used for monitoring. Imbalanced data classification in streaming mining may be applied to monitoring events that do not occur regularly such as fraud detection, intrusion detection, where prompt action may be required. The sectors where the classification of imbalanced stream data may be used include financial institution, retail industry, security, telecommunication, industrial production, transportation and health. In multi-class data sets the labels are specific.

Generally, multi-class imbalance classification may be applicable to scenarios where a finer grain of minority labels are required, where a particular label rather than being classified as a label on its own, is broken down into sub-labels. For example in weather forecast, rather than label liquid precipitation as rain, we may subdivide this category according to the rate of precipitation into light-rain, moderate-rain and heavy-rain. Thus, in cases where heavy-rain is predicted, attention is directed immediately to the flood monitoring and controlling unit.

3.8 Summary

This chapter discussed the class imbalance learning issue and related topics. The modeling of the class imbalance classification task based on the availability of data and application was discussed. Solutions and relevant studies were detailed in this chapter and the chapter was concluded by discussing the application of multi-class imbalance stream classification to real life.

In the next chapter, the methodologies developed in this thesis are discussed. The chapter also elaborates on the framework of the proposed method and the experimental and statistical evaluation.

Chapter 4

SCUT-DS Methodologies

Recall that sampling approaches to solving multi-class imbalanced classification problem is an efficient and flexible technique that does not require domain knowledge. The chances of an algorithm being applicable to many imbalanced domains are greatly increased when using sampling, as noted in the earlier chapters. Thus, as stated earlier in the introductory chapter, we design sampling techniques for multi-class imbalanced classification by extending SCUT [8] for the streaming setting. Recall that, specifically, the goal in this thesis is to improve the average recognition rate for the minority classes in multi-class imbalanced data streams by resampling the training set. Resampling the training set ensures that each training set contains sufficient examples of the minority classes.

In Chapter 2, we provided general information about stream mining, while in Chapter 3, we introduced the concept of class imbalance learning and laid the background for the proposed method in this thesis. Recall that multi-class imbalanced learning is not well-researched compared to two-class imbalanced classification. Furthermore, the few algorithms for multi-class imbalanced learning in literature for the streaming environment have some drawbacks [13], [83], [84]. They include not being flexible because of the need to either assign weight or cost, not combining sampling approaches and not being applicable to imbalanced data stream with high imbalance ratio. And lastly, some of the methods accumulated too many positive instances which may overwhelm the system. The methodologies designed in this thesis seek to address these issues.

In this chapter, we will describe the framework for the two multi-class stream mining methodologies, namely SCUT-DS and SCUT-DS++ extending SCUT from the static to data stream environments. The chapter further elaborates the appropriate evaluation metrics for use in multi-class imbalanced data that will be used for evaluating the performance of the designed techniques. The chapter is concluded by discussing the approaches that will be used for testing the statistical significance of the experimental results.

4.1 Methodologies for SCUT-DS

In this thesis, we propose two effective strategies for addressing multi-class imbalanced learning in data stream, SCUT-DS and SCUT-DS++. Recall from the discussion of sliding windows in Chapter 2, instances in data stream may be grouped into chunks D_0, D_1 …, D_n using the idea of sliding window as they arrive. Recall that in the static setting, it is possible to store all data and set aside a sample as the training set and another as the test. In data stream classification, on the other hand, using interleaved-test-then-train evaluation as discussed in Chapter 2, the first chunk D_0, is used as the training set to build the very first model and the second chunk, D_1 is used for testing the learner built. Subsequently, as windows get filled up, they are first tested, used to update the model and then used to predict incoming streams.

Similar to SCUT, the proposed algorithms resample and balance the training set across all classes before using it to construct, or update, a model. The reason for balancing before updating the model is to ensure that the training set always contains sufficient and equal samples of instances of all the classes encountered in the data stream. Recall that one of the reasons why classical data mining algorithms have low recognition ratio of the minority labels in imbalanced data is because the classification algorithm during training was not presented with enough minority examples like the majority instances. Recall that according to Weiss and Provost [60], balancing the training set improves prediction accuracy of minority class, hence, the reason for resampling before building models. In addition, because stream classification is an incremental process, the model needs to be updated with the new examples of labels found in the incoming stream, while systematically forgetting the past stream, so as not to overburden the system [43]. Thus forgetting is used for selecting important instances and for reducing the size of the data that is used for analysis. Forgetting in this thesis is done using pseudo-window-based forgetting mechanism. (Forgetting techniques was elaborated in Chapter 2.) Our goal in this study is to improve the recalls of the minority instances in incoming stream using updated models that are built using seen and re-balanced training data set. We focus only on the minority instances, because classical classifiers have high recognition rate for the majority instances.

Hence we designed two resampling mechanisms. These resampling techniques were incorporated into the interleaved-test-then-train evaluation algorithm to arrive at the SCUT-DS and SCUT-DS++ algorithms. The version of the evaluation algorithm that was extended is the "chunk by chunk" version, since its description imitates the steps to achieving our aim. As mentioned in Chapter 2, INTER is a chunk-by-chunk version of the interleaved test-then-train evaluation algorithm. Recall that in Section 2.2.4, we mentioned the advantage of using the interleaved-test-then-train evaluation method. The advantage is that almost every instance or chunk except the first chunk in the stream will have the opportunity to be tested with the model while all instances or chunks will be used in updating the model. Hence, we selected the interleaved-test-then-train evaluation algorithm as the extension point for incorporating the resampling algorithms. Thus, we may infer that INTER is an ordinary evaluation technique because it does not contain a resampling algorithm when compared with SCUT-DS and SCUT-DS++.

There is need to be wary of the chunk size and balancing rate used because they may result in higher resource requirement for data preprocessing, resampling, model building and testing. Therefore we followed the balancing idea in SCUT as mentioned.

The framework of SCUT-DS and SCUT-DS++ incorporate two algorithms, “Resampling” and the evaluation algorithm. The task of resampling algorithm is to balance the instances in the training data, while that of the evaluation algorithm is to accumulate instances in the chunk up to a specified size and call the resampling algorithm to resample the training set before updating the model. The evaluation algorithm then builds the model and uses the recently built model to test incoming instances until the chunk size is reached and the process is started over. The process flow for the methodologies is presented in Figure 11.

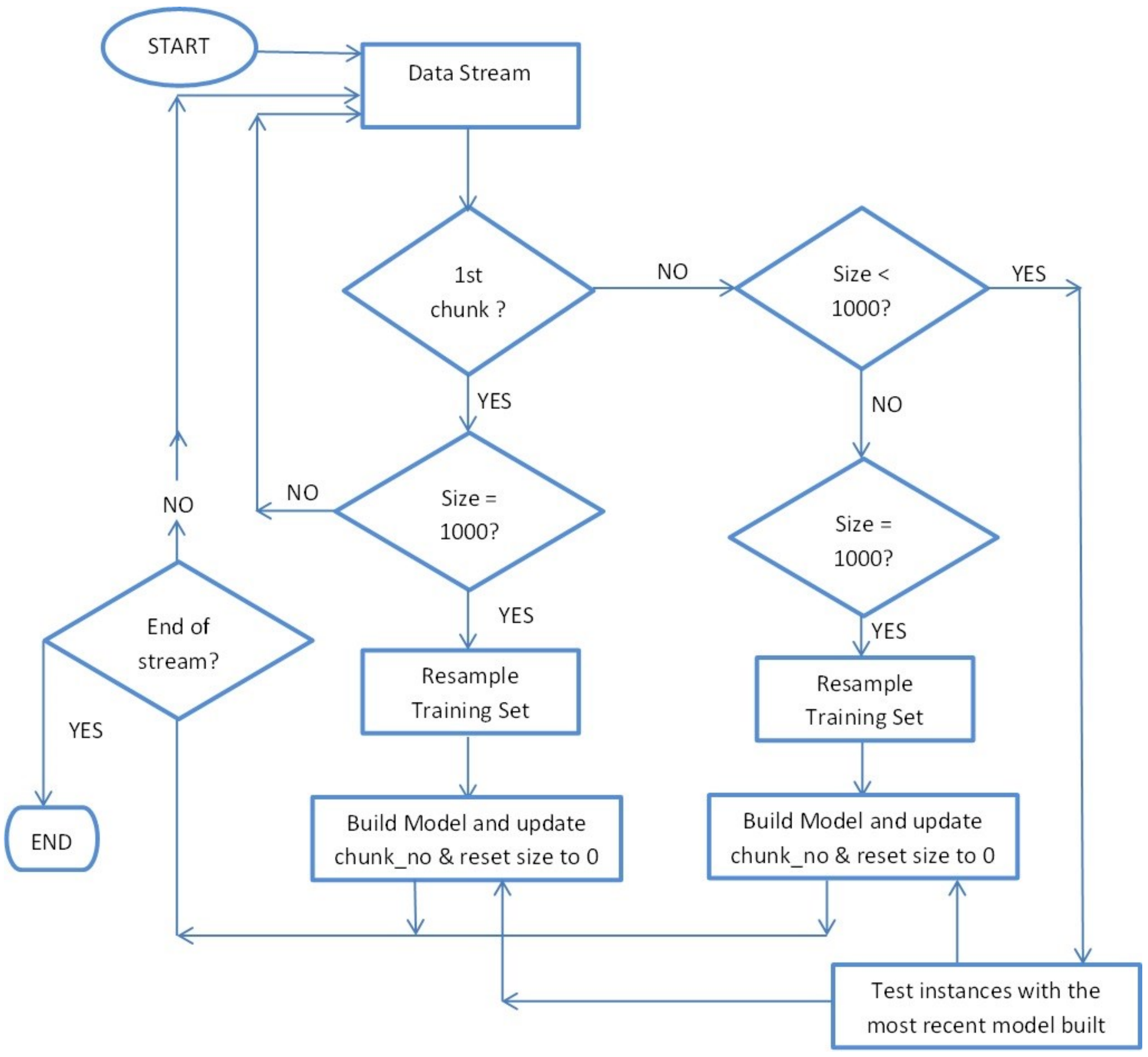

START
Data Stream
1st chunk ?
NO
Size < 1000?
YES
YES
NO
NO
NO
Size = 1000?
Size = 1000?
YES
YES
End of stream?
Resample Training Set
Resample Training Set
YES
END
Build Model and update chunk_no & reset size to 0
Build Model and update chunk_no & reset size to 0
Test instances with the most recent model built

Figure 11. Process Diagram for SCUT-DS.

As the stream flows in, instances are accumulated until a particular chunk size is reached. The accumulated examples are then split into classes. The majority instances are under-sampled per class using cluster-based under-sampling. In contrast to SCUT, the clustering algorithm of choice is K-Means. Although as pointed out in [8], EM could lead to the discovery of the natural groupings in the data set, this may result in numerous scans while trying to maximize the likelihood of clusters instances belong to. It was also said that it sometimes does not converge

[5]. Therefore using EM without specifying the number of clusters to discover might be computationally expensive given the reasoning above. Thus, K-means was chosen after extensive experimentation. While the majority instances were under-sampled, the minority instances were oversampled. The two techniques in this work defers in how minority instances are forgotten and how they are made to be balanced with the other classes in the training set.

As an illustration of resampling in this thesis, consider the scenario where the size of the window in the data stream fills up to 1000 instances. At this point, the instances in the particular window are separated into classes. Let's assume that we have 4 classes in the data stream, comprising of 2 majority classes (A and B) and 2 minority classes (C and D). These instances are split based on their classes. Assume class A contains 500 instances, class B contains 350 instances, class C contains 100 instances and class D has 50 instances in this first chunk. The average resampling rate is calculated as the total number of instances over the number of classes. Thus we have 250 in this case. Class A and class B are grouped into 5 clusters each. We select instances from each of the clusters to arrive at 250 instances for each of these classes. Afterwards, we will apply SMOTE to oversample classes C and D to 250 instances per class. Subsequently, we merge the resampled instances to arrive a new training set. The model is updated with these resampled instances and then the model is tested on incoming stream of size 1000 in Chunk 2. In subsequent streams, once we have reached the specified chunk size's threshold, we will split the previous training set and instances in the recent window based on their classes. We merge them as we described above and use it to predict incoming stream. This process continues until the end of the stream (or infinitely).

The oversampling and under-sampling methods used are explained in the next two sub-sections.

4.1.1 Cluster-based Under-sampling for the Majority Classes

After instances have been separated based on their labels, the majority instances are first clustered into groups per class. Instances are selected per cluster using the same method as in SCUT. The selected instances are merged afterwards to arrive at the specified distribution sample per class. Recall from Chapter 3, that data cleaning tools were used to improve the accuracy of classifiers by removing noise and border line [47]. We avoid the use of data cleaning tools to prevent the removal of examples that may appear as noise in current chunk which may end up being a new concept that needs to be learned in incoming streams. This may hinder the performance of the classifier. The cluster-based under-sampling algorithm is presented in Algorithm 3.

```
ALGORITHM Cluster-Based Under-sampling
Input: Merge current instances in chunk Ci with recent training instances Ti to get Di (Split Ci
to get Di, if chunk is the first)
        s = Sampling Distribution
        l = number of majority instances in each class per chunk in the data stream
        k = number of clusters
Output:  C = Selected instances for training (Resampled training set)
1. For each class i, in the merged instances, Di, i = 1, 2, ..., l where number of instances > s
2.      Cluster Di using K-means algorithm to get Ci
3.      Ci = Ø
4.      For each cluster Ci in Di, i = 1, 2, ..., k
5.          Randomly select instances from Ci making sure that at most s/k instances
             is selected from each cluster
6.          Add selected instances to Ci'
7.      End For
8.      C = Ø
9.      For each selected cluster Ci' i=1, 2, ..., k
10.         Merged all instances selected for each Ci' to C
         ie C = C U Ci'
11.     End for
12. End for
13. Return C
```

Algorithm 3. Cluster-based Under-sampling Algorithm

4.1.2 Sampling the Minority

Recall that the two algorithms we designed differ in the way we forget and generate minority examples in order to arrive at a balanced training set. In the first technique, SCUT-DS, for every chunk or window, the minority examples are oversampled per class using SMOTE to generate synthetic examples each time we update the training data. It may be said that SCUT-DS uses the idea that recent past examples are most relevant for predicting incoming minority instances in incoming stream, following the assumption used in the sliding window technique as discussed in Chapter 2. Thus, the forgetting mechanism employed in SCUT-DS is purely windowing-based. The oversampling algorithm for SCUT-DS is presented in Algorithm 4 below.

```
ALGORITHM SamplingMinorityInSCUT-DS
Input: Split current instances Ci in recent chunk based on their classes to arrive at Di
        s = Sampling Distribution
        f = number of minority instances in each class per chunk in the data stream
Output: Resampled Training Set
1. For each minority class, Di, i=1, 2, ..., f
2.            Apply SMOTE on instances of each of the minority classes
              ie      Apply SMOTE on Di to get Di'
3. End For
4. D' = Ø
5. For each SMOTE-Class Di', i = 1, 2, ..., f
6.      Merge all the SMOTE output, Di' together
          D' = D' U Di'
7. End For
8. Return D' (The rebalanced minority instances including the synthetic instances)
```

Algorithm 4. Sampling Algorithm for Minority Instances in SCUT-DS (SMOTE-based Oversampling).

In the second approach, SCUT-DS++, recent past training examples are kept and merged with the corresponding split minority instances in the recent chunk. If the number of instances in the merged file is less than the specified sampling distribution, we use SMOTE-based oversampling. However, if the number of instances in the merged file is above the specified sampling distribution, the same cluster-based under-sampling for majority instances is applied. If it is equal to the specified sampling distribution, the selected data is used as it is. The forgetting technique employed here is pseudo-windowing based, as explained in Chapter 2, because the size of the resampled training data is greater than the size of the chunk. The sampling algorithm for minority instances in SCUT-DS++ is presented in Algorithm 5 below.

```
ALGORITHM SamplingMinorityInSCUT-DS++
Input: Merge current instances Ci with recent training instances Ti to get Di
       s = Sampling Distribution
       f = number of minority instances in each class per chunk in the data stream
       k = number of clusters
Output: Selected instances for training
1. For each minority class, Di, i=1, 2, ..., f
2.     If ( Di <  s)
3.         Apply SMOTE-based Oversampling on Di to get Di'
4.     End if
5.     If ( Di >  s)
6.         Apply Cluster-based Under-sampling on Di to get Di'
7.     End if
8.     If ( Di =  s)
9.         Di' = Di
10.    End if
11. End For
12. D' = Ø
13. For each Di', i = 1, 2, ..., f
14.    Merge all the SMOTE output, Di' together
15.      D' = D’ U Di'
16. End for
17. Return D'
```

Algorithm 5. Sampling Algorithm for Minority Instances in SCUT-DS++.

4.1.3 Motivation for Accumulating Some Chunks and Using Cluster-Based Under-sampling for Minority

Recall that we discussed discriminative and generative models in Chapter 2, and according to [16], [43] it was said that generative models generalize better than discriminative models in non-evolving stream since the underlying concepts do not change. Generative models like Naïve Bayes do not build models using only the examples seen. The computation of the conditional probability of the seen examples may lead to the expansion of the data space. Whereas, in discriminative models (like Hoeffding Trees) a type of mapping is used, hence it relies heavily on the training sets, building models using trees that branches into the internal nodes and finally labels for prediction. Therefore, in order to enable classifiers of choice irrespective of their type to generalize better and have more examples because minority examples are very hard to come by [79], we decided to accumulate examples in the first *n* chunks. Also, since we restrict our study to streams without concept drift, past examples are assumed useful and not detrimental in predicting incoming streams. Therefore, we accumulated the first *n* chunks, which will result in the expansion of the size of the training set. Note that we set *n* equal to three, by inspection. After the third chunk, chunks are no longer accumulated but recent past training examples are merged with instances in the recent chunk. This is in contrast to the assumption in sliding window where it is assumed that only recent windows are relevant for analyses.

The essence for merging previous training data with the recent chunk, as noted above, is to have more examples per data space. This may reduce the amount of information loss, prevent abrupt forgetting and prevent over-fitting the model to recent chunks. This may also enable the model generalize better on the incoming stream. The reasons for the accumulation are further explained.

Firstly, there will be more examples because concepts in the older training set are merged with new concepts in the recent chunk. Secondly, information loss may be reduced because past concepts are merged with the recent ones thereby covering for the information loss caused by the assumption that only recent window is necessary to predict the labels of incoming streams. Thirdly, concepts are not forgotten abruptly, because instances from other windows are included in the training set. Fourthly, because more concepts are involved, the data region is increased, and because we use cluster-based under-sampling, all data space are represented, thus the algorithm may be able to generalize better on incoming stream.

SCUT-DS's framework is depicted with Figure 12 below. Note that there is no difference in the approach at various times T_i because we constantly generate synthetic examples for the minority labels and under-cluster the majority classes. SCUT-DS++'s framework is presented in Figure 12 and Figure 13 below. For the SCUT-DS++, two figures are used to show how the task is performed at time T_0 and at subsequent times. Recall that the approach to sampling differs, because past minority examples are accumulated.

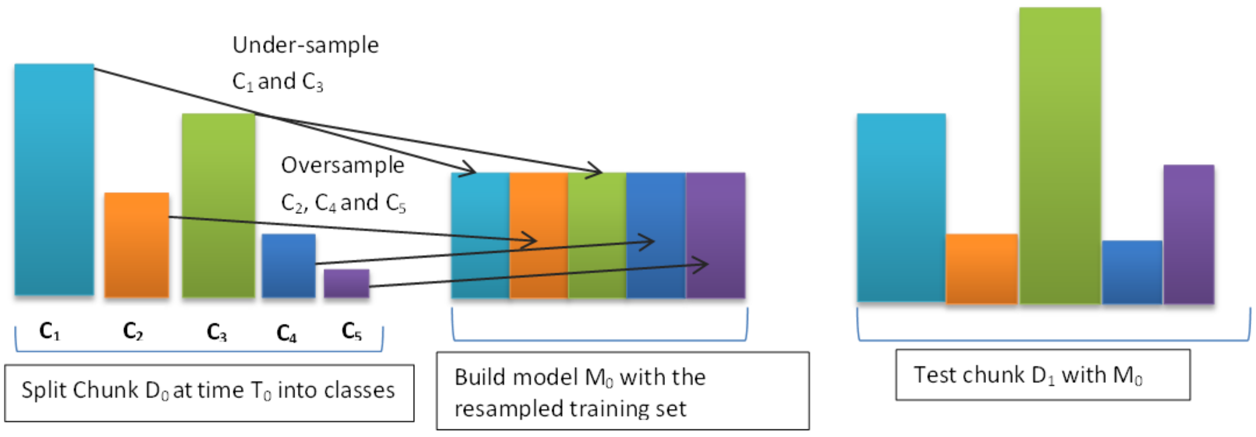
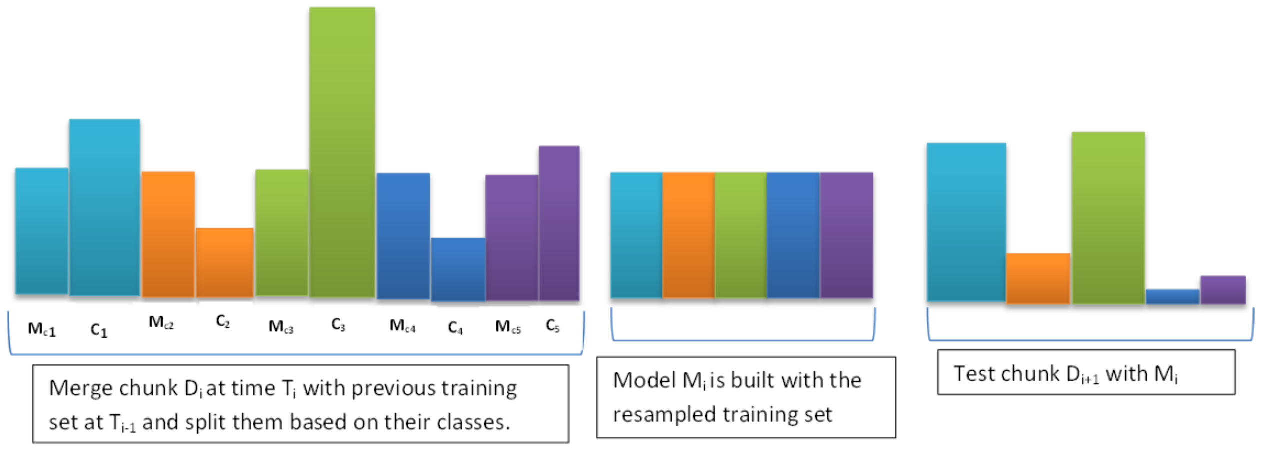

Under-sample
C_1 and C_3
Oversample
C_2, C_4 and C_5
C_1
C_2
C_3
C_4
C_5
Split Chunk D_0 at time T_0 into classes
Build model M_0 with the resampled training set
Test chunk D_1 with M_0

Figure 12. SCUT-DS++ Framework at the initial time T_0 and SCUT-DS Framework: Training set for building model M_0, is chosen by selecting negative instances of classes 1 & 3 using clustering-based under-sampling and positive instances of classes 2,4 & 5 using SMOTE-based oversampling.

Figure 12 represents time T_0 when the first chunk was accumulated. The chunk was split into classes. Cluster-based under-sampling was done on all the separated majority classes, while SMOTE-based oversampling was done on each minority label. The resampling distribution size was set as the average of instances per label in the chunk. Recall that we discussed how the sampling distribution was determined in SCUT in Chapter 3. It follows that the resampling size increases in SCUT-DS++, because of the accumulation of some earlier chunks. Thus, we need to update this average, since the total number of instances to be considered for the training set changes as chunks are accumulated. Therefore, the resampling distribution size was calculated as below at time T_0.

$$Resampling\ distribution\ size = \frac{chunk\_size}{number\ of\ classes\ in\ the\ stream} \tag{4.1}$$



M_{c1}
C_1
M_{c2}
C_2
M_{c3}
C_3
M_{c4}
C_4
M_{c5}
C_5
Merge chunk D_i at time T_i with previous training set at T_{i-1} and split them based on their classes.
Model M_i is built with the resampled training set
Test chunk D_{i+1} with M_i

Figure 13. SCUT-DS++ Framework At Time T_i (where i > 0) : Model M_0 built at Time T_0 is split into classes and merged with the respective instances per class in the recent chunk D_i. For the negative classes, training data are selected using cluster-based under-sampling. For the positive classes, if the total of instances after being merged is less than the average of the model, SMOTE-based oversampling is applied, if the total is more than the average of the model, cluster-based under-sampling is applied, if their total is equal to the average, all instances are selected as the training data.

Figure 13 above presents the scenario when the time is greater than, or equal to 1. From this time onwards, the examples in the recently used training instances are split and merged with the new chunk's split. Recall that the majority examples are under-sampled using cluster-based under-sampling. In the case of the minority labels, if after merging the two samples, the previous and the recent is below the excepted redistribution size, SMOTE-based over sampling is used. If it is above, cluster-based under-sampling is applied similar to the majority classes. The resampled instances are merged and the classifier of choice is applied to build a model. Afterwards, the model is tested on incoming stream.

For chunks 2 and 3 the resampling distribution size, s is calculated by multiplying with the chunk number, but afterwards it maintained at 3, which is the number of chunks accumulated. Thus the new resampling distribution becomes

$$s = \frac{chunk\_size}{number\ of\ classes\ in\ tthe\ stream} * number\ of\ accumulated\ chunks \qquad (4.2)$$

as based on the need a balanced training data equally distributed across the classes present in the data set in [60]. Also Chawla [85] agreed that the use of the natural distribution of a data set as its training data in imbalanced learning does not achieve good recognition for minority instances. Thus, we choose to set the number of the training instances for each class to the average of the chunk in SCUT-DS like in SCUT [8]. This was extended to SCUT-DS++ as discussed above.

4.2 Algorithms

Recall that SCUT-DS and SCUT-DS++ are both made up of two algorithms each, the resampling algorithm and the evaluation algorithm. The pseudo-codes for the resampling and evaluation algorithms for SCUT-DS++ are hereby presented in Algorithm 6 and Algorithm 7 respectively.

```
ALGORITHM Resampling
Input: Chunk of m recently seen instances in the stream C
       Recent previous training data T
       s = Sampling Distribution
       n = number of classes in the data stream
Output: Resampled ARFF File containing recently balanced training data.
1. If (Chunk = firstChunk)
2.     Split the recent chunk C into classes Ci
3.     Ci = Di
4.     For each Di, i= 1,2, …, n
5.         If (Di = minority label)
6.             Apply Minority Sampling algorithm to minority Di
7.         End if
8.         If (Di = majority label)
9.             Cluster-based Under-sampling algorithm to majority Di
10.        End if
11.    End for
12. End if
13. If (Chunk != firstChunk)
14.    Split the recent chunk C into classes Ci
15.    Split recent previous training set T into classes Ti and exclude synthetic instances
16.    Merge Ci and Ti to get Di
17.        For each Di, i= 1,2, …, n
18.            If (Di = minority label)
19.                Apply Minority Sampling algorithm to minority Di
20.            End if
21.            If (Di = majority label)
22.                Cluster-based Under-sampling algorithm to majority Di
23.            End if
24.        End for
25. End if
26. Return Resampled Training Data
```

Algorithm 6. Algorithm for Resampling the Training Set.

Note that in the Resampling algorithm for SCUT-DS, the number of chunk is not accumulated; hence the chunk is not tested if it is the first in the SCUT-DS algorithm. Also, the previous training set is not merged with the recent chunk. The extended evaluation algorithm is presented next.

```
ALGORITHM:SCUT-DS Evaluation Algorithm
Input: D is a non-evolving stream
        Resampled ARFF File containing recently balanced training set.
        n = number of classes in the data stream D
        l = number instances per majority classes
        f = number instances per minority classes
        s = sampling distribution
        W = number of instances per chunk, that is, chunk size
        Accum = number of accumulated chunks
        i= index of chunk
        Learner = Selected classification algorithm
Output: Labeled instances in a file.
1. If ( chunk_size = W)
2.      if (i = 1)
3.              s = W / n
4.              Apply Resample algorithm to Chunk_i
5.              Build model Mi with Learner using the output from Resample algorithm
6.      End if
7.      If (I > 1 & I < Accum)
8.              s = (i* W) / n
9.              Apply Resample algorithm to Chunk_i
10.             Update model Mi with Learner the output from Resample algorithm
11.     End if
12.     If (I > 1 & I >= Accum)
13.             s = (Accum * W) / n
14.             Apply Resample algorithm to Chunk_i
15.             Update model Mi with Learner using the output from Resample algorithm
16.     End if
17. End if
```

Algorithm 7. SCUT-DS++ Evaluation Algorithm.

Recall the discussion in Section 4.1.3, where we discussed the reason for accumulating past minority examples in SCUT-DS++. The reason why the computation of the resampling distribution does not change in SCUT-DS is because we do not accumulate past minority instances.

We discuss the measures used in evaluating multi-class imbalanced data set classification in the following section.

4.3 Evaluation Metrics for Multi-class Imbalanced Learning

This section details the measures used in evaluating learning in multi-class imbalanced data set. We start by a brief discussion of general performance measures used in data mining for classification. We then discuss the metrics for measuring performance in multi-class imbalanced classification. The section is concluded by specifying the evaluation metric that will be adopted in this study and the challenges in evaluating algorithms.

Classification algorithm's performance are generally evaluated based on the ability of the model to correctly classify the test data [5], [53], [86]. Evaluation gives the opportunity for methods that were developed independently of each other to be compared using some benchmarks. Evaluation metrics such as accuracy and error rate are used in static and stream learning. These metrics, although they are widely used, may not be sensitive to the needs of some specific classification problems like class-imbalance and multi-class imbalance classification [86]. These well-known metrics are discussed below.

Overall accuracy which is usually referred to accuracy is not good for imbalance data because it is does not take into consideration the proportion of instances belonging to each class in the data set [60], [41]. Misclassifying minority instances hide under the generalization of accuracy. Accuracy is calculated as the

$$Accuracy = \frac{Total\ number\ of\ instances\ correctly\ classified}{Total\ number\ of\ instances\ in\ the\ dataset} \tag{4.3}$$

Error rate or misclassifying rate is the overall rate of misclassifying instances [5], [53] and it is calculated as

$$Error\ rate = 1 - Accuracy \tag{4.4}$$

or as

$$Error\ Rate\ = \frac{Total\ number\ of\ instances\ misclassified}{Total\ number\ of\ instances\ in\ the\ dataset} \tag{4.5}$$

Similar to overall accuracy, error rate assumes that the class instances have equal distribution [60], therefore it is not a good performance measure for imbalanced data set. The algorithm's poor predictive performance on the minority classes is covered up by the majority classes' predictions because of their proportion in the data set [86].

In order to observe the accuracy and misclassifying rates of each of the classes in the data set, confusion matrix is used. A confusion matrix gives the opportunity to see the number of instances that were correctly predicted and misclassified per class in the data set. Confusion matrix and derivative measures are discussed next.

A confusion matrix is also known as contingency table, as it is an *n x n* matrix table [53] that gives a tabular representation of recognition rate of classes present in a model. This is good for binary-class imbalanced data sets and it may be extended to the multi-class setting [41]. The complexity of the confusion matrix increases as the number of classes in the data set increases [41]. Multi-class confusion matrix was used in [87]. Table 4 shows a 2 X 2 confusion matrix for binary imbalanced data set.

Table 4. 2X2 Confusion Matrix

	True Labels		
Prediction Class		**Positive**	**Negative**
	Positive	True Positives (TP)	False Positives (FP)
	Negative	False Negatives (FN)	True Negatives (TN)

The terms in the confusion matrix are defined below.

True Positives (TP): These are the positive instances that are correctly predicted by the classifier as being positive [5].

True Negatives (TN): They are instances that belong to the negative labels, which are recognized by the classifier as belonging to them [5].

False Positives (FP): These are examples of the negative labels that are incorrectly classified by the classifier as being positive instances [5].

False Negative (FN): They are positive instances that are wrongly classified as negative instances [5].

The terms in the confusion matrix have been combined in different ways to give metrics that have been extended to multi-class imbalanced classification [86]. We discuss the derived measures next.

Sensitivity or True Positive Recognition (TPR) Rate measures the proportion of positive instances in its class that were correctly identified by the model [88]. It gives us an idea of how accurately our model identifies the positive instances. It is calculated as

$$Sensitivity = \frac{TP}{Total\ Number\ of\ Positive\ Instances\ per\ class} \tag{4.6}$$

Specificity or True Negative Recognition (TNR) Rate, unlike sensitivity, it measures the proportion of the negative instances that were correctly identified as negative instances by the learner [88]. From specificity we have the notion of how accurately our model recognizes negative instances. The calculation is given below.

$$Specificity = \frac{TN}{Total\ Number\ of\ Negative\ Instances\ per\ class} \tag{4.7}$$

The Precision of a classifier is the proportion of positive instances that were correctly identified to the total number of predictions of a particular class [41]. It is a measure of the proportion of the classifier's prediction of a particular class that actually belongs to the class. It is calculated as

$$Precision = \frac{TP}{TP+FP} \tag{4.8}$$

Recall measures the ratio of instances of a particular class that were accurately classified. It measures completeness [7], [41]. It is the same as sensitivity. Recall is calculated as

$$Recall = \frac{TP}{TP+FN} \tag{4.9}$$

Sensitivity, specificity, precision and recall evaluates per class [88]. Thus they are single-class metrics, and have been extended and combined in different ways in literature so that the performance of the classifiers on the minority classes is taken into consideration [7], [41], [53]. Examples of these extended metrics include F-Measure, Geometric Mean (G-Mean), ROC, AUC and Precision-Recall (PR) Curve. ROC, AUC and PR-Curve are used to visualize the performance's evaluation.

The single-class measures above were combined for performance evaluation in binary-class imbalanced data set [41]. These extended metrics are discussed below.

F-Measure is the harmonic mean of the two metrics, recall and precision [64]. F-Measure is sensitive to data distribution [41]. The measure is based on the positive prediction rate and negative prediction rates of instances in a class. F-Measure takes into consideration the accuracy and error rate of instances per class, not as a whole where some good performance may over-shadow other deficient performance. It is a better performance measure because accuracy of methods is low if any measure of the two measures is sacrificed for the each other.

$$F\text{-}Measure = \frac{(1+\alpha)*Precision*Recall}{(\alpha *Precision) + Recall} \tag{4.10}$$

The F-measure may be weighted depending on the value assigned to α. We have a balanced F-score which is referred to as F_1 score if $\alpha = 1$, that is precision and recall are assumed to carry equal weights in the metric. Other possible values for α include 2 and 0.5 named as F_2 and $F_{0.5}$ score respectively [88]. In F_2, recall weighs twice as much as precision while in $F_{0.5}$, precision is weighed twice as much as recall [88].

The geometric mean (G-mean) developed by Kubat et al in [89], may be used for evaluating the performance of classifiers on imbalanced data sets [53]. It combines sensitivity and specificity [5], [53], thus like F-Measure, it ensures that one class' performance is not more important than the others [51]. G-mean is calculated for binary class as given below

$$G\text{-}mean = \sqrt{sensitivity * specificity} \tag{4.11}$$

Metrics such as F-Measure and G-mean which combines both are said to be better measures [7], [64]. There are no agreed standardized methods for extending these binary metrics to the multi-class setting [86], although many extensions have been developed in literature [7]. It should be noted however that, under-sampling results in loss of information and thus increases the misclassifying rates of the negative instances which is regarded as FP in the confusion matrix. Recall, that negative classes are the classes with the frequently occurring instances, thus their proportion reflects and overshadows that of the minority in the computation of Precision. Hence, extended multi-class imbalance evaluation metrics such as G-mean and F-measures which incorporates Precision into their computation is subjected to this drawback. Furthermore, recall that our focus is to improve the recognition rates of the minority instances, since classical classification algorithms have higher recalls on the majority classes. Thus, there is need for an evaluation measure that is totally devoid of the influence of the proportion of the majority instances.

Bifet and Frank in [90] proposed prequential evaluation using Cohen's Kappa statistic as the evaluation measure. It was said to be sensitive to imbalanced stream and may measure accuracy overtime [90]. The Kappa statistic which, are referred to as agreement statistics, is said to be able to remove the effect of predicting a label by chance when calculating classifier's accuracy [90], [88].

Although the Kappa statistic was said to be sensitive to imbalance stream and may be extended to multi-class imbalance [90], it does not give details about the accuracy of the minority classes. As stated previously our main goal in this thesis is improve the recognition rate of the instances of the minority classes, since standard classifiers are known to have high accuracy on instances of the majority classes. Hence, we resort to using the recalls of the minority classes as the metric for evaluation in this study. Recalls are single-class metrics, thus in order to arrive at an evaluation metric for all the minority classes, we measure the average of the recalls by adding all the minority classes' recalls and then divided this total by the number of minority classes in the data set. The formula for the Average Recall is given below.

$$Average\ Recall = \frac{\sum_{i=1}^{n} Recall_i}{n} \tag{4.12}$$

Where *n* is the total number of minority classes in the data set. We discuss the challenges in data stream algorithms' evaluation next.

4.3.1 Challenges in Evaluating Data Stream Algorithms

We discussed evaluation of data stream algorithms earlier in Chapter 2. The techniques used have significant effect on the development of effective and efficient algorithms. There should be standardized error bounds, application specific evaluation metrics and publicly available standardized experimental data set that will be used to evaluate the performance of data stream algorithms.

Algorithms in data stream mining usually trade efficiency for accuracy with specified error-bound [24]. Hence, there is the need to specify the acceptable error-bound for the developed algorithms [28]. Currently, the error bounds of stream mining techniques are unknown which makes it difficult to know if the result is within the acceptable accuracy range. Thus, we must be able to test how the developed methodologies' performance compares with existing techniques given a specific error bound. Different applications require different methods for evaluating the task. For example, there is no agreed problem-specific metrics for evaluating algorithms used for concept drift and imbalanced data [20], [91].

Moreover, using small amount of data does not correctly mimic the boundless characteristics of a stream [39], [42]. Thus, it may not test the resilience and effectiveness of a technique [42]. Coupled with size, synthetic data sets may not portray the behavior of real life data. Real life data may behave differently with the algorithm compared with the way the synthetic data did [42]. Hence the need for more publicly available data sets.

Next we discuss the criteria used for evaluating the performance of the algorithms in this thesis.

4.4 Evaluation Criteria

4.4.1 Criteria for Evaluating the Methodologies' Performance

As noted in Chapter 2 and in Section 4.1, the chunk-by-chunk variant of the interleaved-then-train evaluation method is chosen as the evaluation technique in this thesis. In addition as indicated above, the average recall measure (equation 4.12) will be used as the evaluation metric for evaluating the accuracy of developed methodologies in this study. Evaluation criteria give the basis on which the measured evaluation accuracy is analyzed. Thus, in this thesis we will be evaluating our algorithms from four different perspectives.

First, the average recalls amongst the three methodologies, INTER, SCUT-DS and SCUT-DS++ will be compared. The performance of the algorithms will be analyzed in terms of their ability to predict incoming streams. Secondly, the analysis of the accuracy of SCUT-DS and SCUT-DS++ will be used to determine if keeping of previous minority instances generalizes better than synthetic instances on incoming stream. Thirdly, the accuracy of the six classifiers used in this experiment will be compared. Lastly, the resilience of the three algorithms to different levels of noise will be analyzed.

Statistical Significance testing of the experimental results using the four perspectives discussed above will be used to determine if one methodology is significantly better than the other. Statistical significance is the topic in the next section.

4.4.2 Metrics for Testing Statistical Significance

Statistical significant tests are carried out to determine whether the observed results from the evaluation of the performance of algorithms are not attributed to chance [88]. There are basically two types of statistical significance test, parametric and non-parametric. A parametric test makes a strong assumption about the data distribution of the observation in making the null hypothesis [88]. Non-parametric tests rank the algorithms based on the observed data, they do not assume that the data have a normal distribution [88]. ANOVA is an example of parametric test while Wilcoxon signed-rank test and Friedman's test are examples of non-parametric test [88], [92].

In statistical significance testing we propose a null hypothesis, H_0 that assumes that there is no significant difference amongst the algorithms based on our observations given a certain p-value. The p-value gives the significance level and the commonly used values are 1% and 5% and is used to determine whether we will accept or reject the null hypothesis [88], [93]. If we obtain a value lower than the p-value, this will lead to the acceptance of the alternate hypothesis H_1, which states that there is significant difference amongst the observations. Otherwise we accept the null hypothesis. Wilcoxon's signed-rank test, Friedman's test and Conover post-hoc test are further explained below.

Wilcoxon's Signed-Rank Test: The Wilcoxon's signed-rank test is a non-parametric test that checks for the existence of significant statistical differences among the observed data for two algorithms on single or multiple data sets [88], [94]. Wilcoxon signed-rank tests are not really affected by extremely good or bad performances from few observed data because the data are ranked [94].

Friedman's test: Friedman's test is another type of non-parametric test that is used for verifying if the accuracies of at least a pair of algorithm among multiple algorithms on multiple data sets differs significantly [88], [94]. The null hypothesis is rejected if the p-value signifies that such statistical difference exists [88]. This will lead to the acceptance of the alternate hypothesis and the need to conduct further pairwise comparison test to determine the classifiers with the difference(s) [88], [94].

Conover Post-hoc Test: Conover post-hoc test is an example of the additional pairwise comparison test that is carried out when the alternate hypothesis is accepted after a Friedman's test. It is used to conduct a pairwise comparison amongst algorithms [95], [96]. Every algorithm is compared with every other ones to determine the pairs with the statistical difference.

According to Japkowicz and Shah, different types of statistical significance tests apply to different studies depending on the number of domains and the number of algorithms [88]. Because we do not assume a normal data distribution in this thesis, we will be using non-parametric tests. In addition, we will be using the Friedman's test for comparing three algorithms, SCUT-DS, SCUT-DS++ and INTER on multiple domain sets, and post-hoc tests using Conover Post-hoc test will be conducted if the alternate hypotheses are accepted. Also, the Wilcoxon test will be used in comparing two algorithms on different domains, that is, in testing the algorithm that generalizes better between SCUT-DS, and SCUT-DS++.

4.4.3 Summary

Our aim in this thesis is to design a flexible and effective algorithm that will improve the recognition rate of instances of the minority classes in multi-class imbalanced data stream. The frameworks and algorithms for the designs were discussed in this chapter. We talked about the metrics and the basis for evaluating the performance of the algorithms. Furthermore the method to be employed in testing statistical significance of the algorithm's performance was discussed. The next chapter presents the experimental design and the results obtained from the experimental evaluation are reported and analyzed.

Chapter 5

Experimental Design and Evaluation

In Chapter 4, we presented the algorithms, the evaluation measure, evaluation criteria and the technique to be used for testing for statistical significance. In this chapter, we discuss the experimental design and the results of the SCUT-DS and SCUT-DS++ approaches introduced in Chapter 4.

This chapter is organized as follows. Section 5.1 discusses the experimental design. Section 5.2 analyses the results of the experimental evaluation, the statistical validation is presented in Section 5.3 and finally in Section 5.4, the lessons learned.

5.1 Experimental Design

The experiment was performed on a machine with an Intel(R) Core(TM)i5-6200U processor, CPU @ 2.30GHz - 2.40GHz processor, 8.0 GB RAM on the 64-bit Windows 10 Operating System (OS). The SCUT-DS and SCUT-DS++ algorithms were implemented using the WEKA [97] and MOA [42] frameworks. The WEKA and MOA Application Program Interfaces (API) were extended using Java, with Eclipse as the Integrated Development Environment (IDE). A resampling algorithm was implemented and the evaluation algorithm was extended by incorporating the resampling algorithm into it.

The algorithms were tested with 14 real data sets and 31 synthetic data sets with varying levels of noise. The software used for the study is described in the next section.

5.1.1 Software Used

The software for experimentation in this thesis includes WEKA (release 3-7-12), MOA (MOA-release-2016.04) and the Eclipse IDE (Eclipse Mars.2 Release (4.5.2)), the platform used for utilizing the APIs in WEKA and MOA.

The classification algorithms used in this study are presented in the following section.

5.1.2 Classification Algorithms Used for Experimentation

The six classifiers that are to be experimented in this study, namely the Hoeffding tree (HT), Naïve Bayes (NB), OzaBag-HT, OzaBag-NB OzaBoost-HT, and OzaBoost-NB methods have been implemented in MOA [25]. They are briefly discussed below.

Hoeffding tree (HT): HTs are incremental decision trees for data stream classification [25], [37]. This category of decision trees use the Hoeffding bound to determine the smallest sample size that is needed to determine the splitting attribute. The assumption is that the tree produced using the sample data is synonymous to the tree that would have been derived if all data were stored [25], [37]. The Hoeffding bound Ɛ is given by the formula below

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{5.1}$$

The Hoeffding bound states that given the probability 1- Ɛ, the true mean of a random variable of range *R* will not be different from the estimated mean after *n* independent observation by more than the Hoeffding bound [42].

The advantage of this tree is that multiple scan is not needed and it is an incremental learner. The creators also noted that it guarantees performance which is asymptotically close to that of classical learner with infinite examples [39], [42]. The research issue with HT is that it spends a considerable time trying to split when there is a tie, and memory usage and nodes are not being updated once they are created [28], [37].

Naive Bayes (NB): This is an efficient algorithm that assumes conditional independence amongst attributes similar to the traditional NB algorithm [5], [25]. The training sets are used to build models that are based on the Bayes' conditional probability, these models are used in predicting the labels of the test instances [5], [25]. The models built are incremental, they are similar to the model built in the static Naive Bayes [19], [25]. The forgetting techniques used with the incremental NB determine how older and irrelevant concepts are forgotten [19]. The advantage of this learning mechanism is that predictions are readily available [25], while its disadvantage is the assumption of independency which may not be true for some data set [5].

OzaBag and OzaBoost: OzaBag is an online variant of the Bagging ensemble while OzaBoost is an online version of the Boosting ensemble introduced in Chapter 2 [25], [38], [39]. Recall that we mentioned in Chapter 2 that ensembles make predictions using many base classifiers. Ensembles use these multiple models to predict the labels of the incoming test instances. Each of the base model is built from various samples of the training set [39]. Bootstrapping is used to create a number of samples from the training data, where each of the training instances has a certain probability of being replaced in each of the training samples [5], [39], [88]. In both variants of online ensembles, the probability of replacement is computed using the Poisson distribution [39].

In OzaBag [39], the instances do not have weights. In order to predict the label of a test instance, each of the base models predicts the class of the test instances. The class with the majority vote is agreed to be the final prediction for the label of that instance.

In contrast to OzaBag, weights are assigned to the instances in OzaBoost. The Poisson distribution is increased for an instance in the next training samples for building the next classifiers each time it is wrongly predicted [39]. The weights of the wrongly classified instances are increased, hence the error rates of the base models are updated too [39]. The final prediction in OzaBoost is the weighted prediction of all the base classifiers.

Generally in these online ensembles, the base models are updated with each training set. In addition, any classification algorithms such as NB, HT can be used with these ensembles. Recall that we mentioned in the introductory chapter that will be using OzaBag and OzaBoost with HT and NB as their classification algorithms in this thesis.

5.1.3 Clustering Algorithm Used for the Experiment

Recall that the EM clustering approach was used in the original SCUT algorithm. The EM technique is iterated several times in order to maximize the membership likelihood of instances [5]. Instances may belong to more than one cluster with different membership likelihood. EM may either be allowed to discover clusters on its own or the required number of partitions maybe specified [8]. EM has not been used in this thesis, because it would be computationally expensive to maximize the membership likelihood of instances in an online data stream setting. In our work, the K-means algorithm was selected. K-means is a well-known clustering algorithm for partitioning data into groups [21]. K-means uses distance as the similarity measure, thus it aims to minimize the squared error of the partitions based on the distance between the cluster center and points in the cluster. The K-means technique reduces the squared error that partitions instances based on the number of clusters. A restriction of K-means is that the number of partitions must be specified [21]. Also, this algorithm is best suited for scenarios where the clusters are of similar shape and size; as is the case in our use cases.

5.1.4 Data sets

Most of the data sets found in public data repositories are not suitable for evaluating multi-class imbalanced data stream classification. They either contain too few examples or are not multi-minority or multi-majority data set. In order to conduct experiments in this research, synthetic data sets comprising varying multi-class ratio and distribution ratio were generated, using the Waveform and Light Emitting Diode (LED) data generators in MOA [25]. The real data set used are historical Weather data we obtained from the Open Data Canada [98] repository and publicly available New York City (NYC) Taxi Fare Data [99]. The data sets are further discussed in the section below.

5.1.5 Data Generation

The following two sub-sections give a description of how different data sets were derived from the data sets above.

5.1.5.1 Synthetic Data sets

The synthetic data sets were generated from the Waveform Generator and LED Generator [25]. . It should be noted that the data were generated without concept drift, because concept drift is not being considered in this study. The data set are generated with different levels of noise. The two generators in [25] are discussed below.

LED Generator: The LED generator generates streams that may be used for predicting the digits that is displayed in a seven-segment LED display [25]. It is based on the work in [100]. The data set contains seven Boolean attributes and ten labels. The labels are the set of decimal digits 0 - 9, while the features are the seven light-emitting diodes, which may be either 0 or 1. The probability that each of the attributes in this data set will be inverted is 10%. The LED Generator in MOA comes in two variants, depending on if it generates the stream with concept drift or without concept drift. The LED generator in MOA may also generate data set with different levels of noise.

The type of LED Generator that was used in this work is the one without drift but with three different levels of noise, 10%, 20% and 30%. The 0% noise level of LED data set is not challenging because the data space of the labels is not wide. In total, seven synthetic data sets were generated using the LED data set. These comprise of four classes and five classes per noise levels. Table 5 shows the class distribution of the instances selected.

Table 5. Distribution of the LED Data set for each Noise Level.

No of Class	Majority	Minority	Distribution Ratio	Class 1	Class 2	Class 3	Class 4	Class 5	Total
4	1	3	75/15/5/5	37,500	7500	2500	2500		50,000
4	1	3	80/14/5/1	40000	7000	2500	500		50000
4	2	2	55/30/10/5	27500	15000	5000	2500		50000
4	2	2	50/44/5/1	25000	22000	2500	500		50000
5	1	4	69/15/10/5/1	34500	7500	5000	2500	500	50000
5	2	3	50/34/10/5/1	25000	17000	5000	2500	500	50000
5	3	2	32/30/23/10/5	16000	15000	11500	5000	2500	50000

Waveform Generator: The Waveform Generator was used for generating stream that may be grouped into three classes of waveforms [25]. These waveform classes are made by combining two or three waves. A Waveform Generator can either be with or without concept drift. It may be grouped into Wave21 and Wave40. Wave21 has 21 attributes while Wave40 has 40 attributes, both of them contains noise. 19 extra irrelevant attributes are included in the Wave40 [25]. Table 6 below gives the distribution of the waveform data set that is used in this research. Five data sets each were created for the experiment from the two types of Waveform using the Waveform Generator.

Table 6. Distribution of the Waveform Data set Used for the Experiment.

No_of_Classes	Majority	Minority	Distribution Ratio	Class 1	Class 2	Class 3	Total
3	1	2	75/15/10	37500	7500	5000	50000
3	1	2	80/5/15	40000	2500	7500	50000
3	1	2	94/5/1	47000	2500	500	50000
3	2	1	45/45/10	22500	22500	5000	50000
3	2	1	54/40/6	27000	20000	3000	50000

5.1.5.2 Real Data set

The Weather data set comprises of historical weather data from 1997 to 2017 for a total of ten provinces in Canada. These data were obtained from [98]. Our assumption in this study is that the stream does not contain missing value, thus rows and columns with missing values were deleted to arrive at the final data set. In total, nine different variations of the weather data set were used in this thesis. Note that the “Ice Crystals” label was renamed to “Ice” and “Snow_Blowing_Snow” to “Snow” for the purpose of simplicity and clarity. The distributions of the various weather data sets used are presented in Table 7. The name convention used for the data set is the hyphenation of the classes in the data set.

Table 7. Weather Data set Distribution

Data set / Combination_Used	No_of_Classes	Majority	Minority	1	2	3	4	5	Total
Cloudy-Fog-Snow	3	1	2	21146	1557	4091			26794
Clear-Fog-Ice	3	1	2	27209	1557	2581			31347
Cloudy-Snow-Ice	3	1	2	21146	4091	2581			27818
Cloudy-Clear-Fog	3	2	1	21146	27209	1557			49912
Cloudy-Clear-Snow	3	2	1	21146	27209	4091			52446
Cloudy-Clear-Ice	3	2	1	21146	27209	2581			50936
Cloudy-Clear-Snow-Ice	4	2	2	21146	27209	4091	2581		55027
Cloudy-Snow-Ice-Fog	4	1	3	21146	4091	2581	1557		29375
Cloudy-Clear-Snow-Ice-Fog	5	2	3	21146	27209	4091	2581	1557	56584

The second real data set is the NYC Taxi Fare Data [99]. The NYC Taxi Fare Data is publicly available and it gives information about taxi charges and tips collected during certain period in New York based on the driver, pick-up and drop-off location and date. It contains two files, *trip_data* which contains information about the trip and *trip_fare* which stores information about the charges of the trip. They are joined based on their common attribute and the data were grouped into five bins based on the amount of tip. The groupings served as the labels. Table 8 presents the grouping criteria used and the number of instances that fall into each category.

Table 8. Grouping Criteria for the NYC Taxi Fare Data set.

Key		
Class	Tip Amount	No of Instances
1	>0 - 0.99	31157
2	1 - 1.99	26301
3	2 - 2.99	9440
4	3 - 5.99	3512
5	> 6	1876

Thus, with these grouping we derived five different combinations of data set using varying proportion of data with different number of classes. The data sets, their combination and distribution are presented in Table 9 below. The naming nomenclature for the data set is the combination of the classes in the data set. Overall, 45 data sets were used for the experimental evaluation.

Table 9. NYC Taxi Fare Data set Distribution.

Data set	No_of_ Classes	Majority	Minority	Combination_Used	1	2	3	4	5	Total
NYC_123a	3	1	2	1_2_3	75000	15000	10000			100000
NYC_123b	3	2	1	1_2_3	50000	45000	5000			100000
NYC_1234a	4	2	2	1_2_3_4	50000	44000	5000	1000		100000
NYC_1234b	4	1	3	1_2_3_4	75000	15000	5000	5000		100000
NYC_12345	5	2	3	1_2_3_4_5	50000	34000	10000	5000	1000	100000

5.2 Experimental Results

The results of the experimentation conducted against the data sets described in Section 5.1.4 are presented in this section. Recall that the metric chosen for evaluating performance is the average recall of all the minority classes in the data set. Thus, the average recalls of the minority classes present in the data set is presented against each classifier and each algorithm in tabular form. (The tables with the individual recalls of the minority classes are presented in Appendix A). The hyphenated suffix behind the name of the data set represents the proportion of instances of each class in the data set. For illustration, the data set, Wave21_75-15-10 implies that this particular Wave21 data set has three classes and the classes are in proportion 75:15:10.

We hereby present the results of the experiments, starting with the synthetic data set and then followed by the real data set. The synthetic data sets are presented according to the data set and noise levels of the data set in the following sub-sections.

5.2.1 Results against Synthetic Data

5.2.1.1 Waveform Data set

The results of the two variety of waveform data set are presented in the following subsections, starting with the Wave21 data sets and followed by the Wave40 data sets.

5.2.1.1.1 Results of Wave21 Data Sets

The results of the experiment performed with the Wave21 data sets are displayed in Table 10. The results, as shown in Table 10, when comparing all the three methodologies, indicate that both SCUT-DS and SCUT-DS++ algorithms outperformed INTER on all the data set. In all cases, the accuracy of SCUT-DS and SCUT-DS++ were better than that of INTER. SCUT-DS and SCUT-DS++ yielded good results in terms of the average recalls of all the minority classes.

Table 10. Experimental Results of Wave21 Data Sets.

		Classifier Used	**HT**	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**	**OzaBoost-NB**
Data set	**No of Attributes**	**Algorithm**	**Average Recalls of Minority Classes in the Data set**					
Wave21_75-15-10	**21**	**INTER**	0.6098	0.9393	0.6149	0.9382	0.7116	0.9334
		SCUT-DS	0.8325	0.9466	0.8469	0.9463	0.8525	0.9424
		SCUT-DS++	0.8450	0.9489	0.8716	0.9490	0.8555	0.9465
Wave21_94-5-1	21	**INTER**	0.4367	0.8826	0.3913	0.8682	0.5399	0.8752
		SCUT-DS	0.7629	0.9241	0.7949	0.9251	0.7607	0.9223
		SCUT-DS++	0.7396	0.9331	0.7540	0.9331	0.7331	0.9294
Wave21_80-5-15	21	**INTER**	0.5540	0.9297	0.5220	0.9297	0.6790	0.9265
		SCUT-DS	0.8340	0.9415	0.8584	0.9409	0.8362	0.9375
		SCUT-DS++	0.8383	0.9429	0.8572	0.9427	0.8309	0.9398

		Classifier Used	HT	NB	OzaBag-HT	OzaBag-NB	OzaBoost-HT	OzaBoost-NB
Data set	**No of Attributes**	**Algorithm**	**Average Recalls of Minority Classes in the Data set**					
Wave21_54-40-6	21	**INTER**	0.3733	0.9158	0.3722	0.9125	0.6736	0.9091
		SCUT-DS	0.7889	0.9247	0.7866	0.9247	0.7832	0.9216
		SCUT-DS++	0.7886	0.9291	0.8422	0.9294	0.8293	0.9179
Wave21_45-45-10	21	**INTER**	0.5831	0.9273	0.5896	0.9262	0.7555	0.9181
		SCUT-DS	0.8030	0.9368	0.8166	0.9358	0.8244	0.9301
		SCUT-DS++	0.8486	0.9423	0.8600	0.9425	0.8439	0.9405

The greatest improvement, with around 200% increase in the accuracies of the SCUT-DS and SCUT++, when compared with INTER was obtained in the Wave21_54-40-6 with OzaBag-HT classifier and Wave21_94-5-1 with HT and OzaBag-HT classifiers. The overall best accuracies of SCUT-DS and SCUT-DS++ were observed in the Wave21_75-15-10 data set. The highest accuracy obtained from SCUT-DS was 0.9466 against Wave21_75-15-10 with NB as the classifier. While the highest in SCUT-DS++ was 0.9490 against Wave21_75-15-10 using OzaBag-NB. The lowest accuracy obtained from SCUT-DS and SCUT-DS++ is 0.7607 and 0.7331 respectively is against Wave21_94-5-1 with OzaBoost-HT classifier. The accuracy of INTER at this point was 0.5399. Thus SCUT-DS and SCUT-DS++ may be said to improve accuracies of all classifiers used. These improvements may be attributed to the resampling techniques employed in SCUT-DS and SCUT-DS++. The cluster-based under-sampling, generation of synthetic instances and forgetting technique employed by SCUT-DS and SCUT-DS++ may be said to be responsible for the high accuracies. The rebalancing of the training sets ensures that instances of the minority classes are adequately represented. Cluster-based under-sampling ensures that relevant instances which represent all identified data space of the classes are selected into the training data. The forgetting mechanism employed prevents already learned concepts from being forgotten.

Overall, when comparing the accuracies of SCUT-DS and SCUT-DS++, in most cases the accuracies of SCUT-DS++ were higher than that of SCUT-DS. Recall the discussion in Chapter 4 about past and synthetic instances. Thus, this improvements may be attributed to the past accumulated minority instances being more relevant in predicting the incoming streams than the synthetic instances generated from the most recent window.

Looking at the results from the perspectives of the six classifiers, all the classifiers were effective against the SCUT-DS and SCUT-DS++ algorithms. There were many cases with INTER where the classifiers had poor accuracies. Generally, NB and NB-based ensemble classifiers were observed to have higher average recalls than their HT counterparts against all the Wave21 data set. The accuracies of the ordinary NB against all the data sets were slightly lower, or slightly higher or comparable to those of the NB-based ensembles. While, the accuracies of the ordinary HT classifiers were observed to be slightly lower, slightly higher or comparable to those of the HT-based ensembles. Recall that in Chapter 2, HT and HT-based classifiers were said to be highly dependent on the number of examples of the classes seen. Also, recall from Chapter 3, that instances of the minority labels have lower prediction rates, because they are not well represented. Thus based on the observations from the six classifiers, it may be said that INTER when compared to the other two algorithms, had lower average recalls from the results in Table 10 because of the reasons given above.

We evaluate the results from the experimentation of the Wave40 data sets in the next section.

5.2.1.1.2 Results of Wave40 Data Sets

Table 11 displays the experimental results with the Wave40 data sets. Recall that the Wave40 data sets have extra 19 irrelevant attributes. From the accuracy of all the three algorithms in Table 11, similar observations were made to those for Wave21 in Table 10 above, INTER had the lowest average minority recalls of the three algorithms, while SCUT-DS++ had the highest. The accuracies of SCUT-DS and SCUT-DS++ may be said to be generally good.

Table 11. Experimental Results of Wave40 Data Sets.

		Classifier Used	**HT**	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**	**OzaBoost-NB**
Data set	**No of Attributes**	**Algorithm**	**Average Recalls of Minority Classes in the Data set**					
Wave40_75-15-10	**40**	**INTER**	0.5227	0.9388	0.5981	0.9378	0.7315	0.9330
		SCUT-DS	0.8240	0.9399	0.8382	0.9401	0.8275	0.9321
		SCUT-DS++	0.8591	0.9477	0.8729	0.9470	0.8501	0.9394
Wave40_94-5-1	**40**	**INTER**	0.4086	0.8808	0.3887	0.8688	0.4393	0.8647
		SCUT-DS	0.6905	0.8802	0.6986	0.8785	0.6654	0.8775
		SCUT-DS++	0.6369	0.8539	0.6721	0.8537	0.6785	0.8694
Wave40_80-5-15	40	**INTER**	0.5080	0.9286	0.4910	0.9272	0.6727	0.9191
		SCUT-DS	0.8166	0.9311	0.8318	0.9310	0.8007	0.9193
		SCUT-DS++	0.8436	0.9436	0.8608	0.9430	0.8308	0.9346
Wave40_54-40-6	40	**INTER**	0.3763	0.9084	0.5501	0.9046	0.6576	0.9040
		SCUT-DS	0.7214	0.9026	0.7397	0.9019	0.7516	0.8958
		SCUT-DS++	0.8219	0.9460	0.8622	0.9457	0.8222	0.9447
Wave40_45-45-10	40	**INTER**	0.5010	0.9211	0.5833	0.9193	0.7186	0.9116
		SCUT-DS	0.7738	0.9207	0.7910	0.9214	0.7930	0.9169
		SCUT-DS++	0.8570	0.9366	0.8767	0.9370	0.8401	0.9358

The best percentage improvement of around 200% similar to the Wave21 data set, was observed with Wave40_54-40-6 with HT classifier. Likewise, the overall best average minority recalls of SCUT-DS and SCUT-DS++ were observed in the Wave21_75-15-10 data set. The highest average minority recalls of SCUT-DS was 0.9401 against Wave21_75-15-10 using OzaBag-NB, while that of SCUT-DS++ was 0.9477 against Wave21_75-15-10 using NB. The lowest average recall was observed in data set Wave40_94-5-1. It was 0.6654 for SCUT-DS against OzaBoost-HT and 0.6369 for SCUT-DS++ against HT. The overall lowest improvements were observed with this data set too. This may be said to be caused by the proportion of the minority classes to the majority class in this data set when compared with the other Wave40 data sets. The sampling rate although it improved the accuracy of the minority instances, it may be said that it is not the optimal sampling rate for the minority instances.

Overall SCUT-DS++ have higher accuracy than SCUT-DS. The same reason given for the Wave21 data set may be attributed to this behavior.

From the perspective of the six classifiers, more classifiers have poor accuracies when used with INTER. The accuracy of all the classifiers may be said to improve with SCUT-DS and SCUT-DS++. Similar to the observations with the results of Wave21 in Table 10, NB and NB-based ensemble classifiers have comparable accuracies. The HT and HT-based ensembles have close accuracies too. In addition, the accuracies of the HT and HT-based classifiers are lower than their NB counterparts. Thus, the same reason given in the analysis of the Wave21 may be attributed to this observed behavior.

Recall that the Wave40 data sets have additional irrelevant attributes. Comparing the results in the two tables, Table 10 and Table 11 based on this, the accuracies obtained from the experimentations may be said to be comparable since slight differences in accuracies which do not follow specific pattern were observed. Thus, it can be concluded that the introduction of the extra irrelevant attributes slightly affected the accuracy of the three methodologies and was worst with INTER.

5.2.1.2 LED Data set

In this section we present the results of the 21 LED data sets described in Section 5.1 according to their noise level in the next sub-sections.

5.2.1.2.1 LED Data Set With 10% Noise

The results of the LED data set with 10% noise are shown in Table 12. The LED data set with 0% percent noise is not interesting for this study because the three algorithms had 100% recalls. This observation is based on experimentation and it may be concluded that the feature and data space of the classes of the 0% LED data set do not spread. Also, the class boundaries do not overlap. Recall the discussion in Chapter 3, about the reasons why some data set are difficult to learn.

Table 12. Result of LED Data sets with 10% Noise.

		Classifier Used	**HT**	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**	**OzaBoost-NB**
Data set	**Noise Level**	**Algorithm**	**Average Recalls of Minority Classes in the Data Set**					
LED_50-44-5-1	10%	INTER	0.6486	0.6798	0.6494	0.6753	0.6462	0.6932
		SCUT-DS	0.6751	0.7339	0.6978	0.7339	0.7275	0.7422
		SCUT-DS++	0.8021	0.8003	0.8121	0.8001	0.8031	0.8069
LED_55-30-10-5	10%	INTER	0.7813	0.8127	0.7631	0.8095	0.7414	0.7948
		SCUT-DS	0.8291	0.8566	0.8449	0.8568	0.8330	0.8486
		SCUT-DS++	0.8572	0.8637	0.8592	0.8625	0.8523	0.8628
LED_75-15-5-5	10%	INTER	0.7724	0.7688	0.7703	0.7662	0.7879	0.8000
		SCUT-DS	0.8661	0.8779	0.8720	0.8779	0.8621	0.8739
		SCUT-DS++	0.8840	0.8870	0.8860	0.8879	0.8820	0.8884
LED_80-14-5-1	10%	INTER	0.7369	0.7394	0.7331	0.7392	0.7522	0.7581
		SCUT-DS	0.7864	0.8094	0.8012	0.8081	0.8057	0.8184

		Classifier Used	HT	NB	OzaBag-HT	OzaBag-NB	OzaBoost-HT	OzaBoost-NB
Data set	Noise Level	Algorithm	Average Recalls of Minority Classes in the Data Set					
		SCUT-DS++	0.8543	0.8561	0.8641	0.8571	0.8404	0.8722
LED_32-30-23-10-5	10%	INTER	0.7551	0.7768	0.7733	0.7760	0.7716	0.7758
		SCUT-DS	0.8335	0.8330	0.8278	0.8325	0.8200	0.8348
		SCUT-DS++	0.8576	0.8620	0.8604	0.8612	0.8496	0.8600
LED_50-34-10-5-1	10%	INTER	0.7047	0.7632	0.6821	0.7552	0.6597	0.7464
		SCUT-DS	0.7492	0.7782	0.7567	0.7783	0.7345	0.7815
		SCUT-DS++	0.8155	0.8439	0.8237	0.8437	0.8212	0.8434
LED_69-15-10-5-1	10%	INTER	0.7698	0.7974	0.7544	0.7902	0.7467	0.7894
		SCUT-DS	0.8010	0.8144	0.8068	0.8151	0.7999	0.8202
		SCUT-DS++	0.8415	0.8456	0.8421	0.8452	0.8388	0.8448

From Table 12, SCUT-DS and SCUT-DS++ have higher average minority recalls on all the data sets than INTER. This is similar to the observations in the results of Wave21 and Wave40 above. Thus, it may be inferred that the resampling approach used in SCUT-DS and SCUT-DS++ yielded better results than INTER and the reason is similar to the reasons given for Wave21 and Wave40 above.

When comparing the two algorithms, SCUT-DS++ has higher average minority recalls than SCUT-DS on all classifiers and all data sets. Hence, the accumulation of past minority instances may be said to aid the boosting and prediction of the minority instances in incoming streams. Clustering-based under-sampling of the past minority instances may be said to aid the selection of relevant instances from all data space. This ensures better generalization on incoming stream than the use of synthetic instances from recent chunk. A large difference in the accuracy of SCUT-DS and SCUT-DS++ was observed from the data sets, LED_50-44-5-1, LED_80-14-5-1 and LED_50-34-10-5-1. The highest average minority recalls in both algorithms were obtained from the LED_75-15-5-5 data set.

From the analyses of the results from the perspective of the six classifiers, similar to Wave21 and Wave40 results, SCUT-DS and SCUT-DS++ have better accuracy than INTER. As observed in the Wave21 and Wave40, the accuracies of the NB and NB-based ensembles are similar, while those of the HT and HT-based ensemble classifiers are comparable. The NB and NB-based ensembles have higher accuracies than the HT and HT-based classifiers. The same reason given for the Wave21 and Wave40 may be used to explain this.

However, the difference in the average recalls between the NB and NB-based ensembles and HT and HT-based ensembles is not much compared to the Wave21 and Wave40 datasets. This may be because the data spaces defined by the training data are not wide-spread. Hence the examples seen by the HT related classifiers' generalization is similar to the conditional probability of the NB related classifiers. Overall, the best performing classifiers are NB and OzaBoost-NB. The two algorithms, SCUT-DS and SCUT-DS++ at 10% noise level may be said to be tolerant to noise.

5.2.1.2.2 LED Data Set With 20% Noise

Table 13 depicts the results of the experiment with the LED data set with 20% noise. The same pattern in Table 12 above was observed in Table 13, except that the best performing classifiers now include OzaBoost-HT. Thus, it may be said that the Boosting classifiers were able to generate more diverse training set for building diverse models used for prediction. The accuracies of all the classifiers have dropped considerably. The clustering-based under-sampling and SMOTE approaches to resampling used may be said to be highly dependent on the data they obtained from the chunk.

Table 13. Result of LED Data sets with 20% Noise.

		Classifier Used	**HT**	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**	**OzaBoost-NB**
Data set	**Noise Level**	**Algorithm**	**Average Recalls of Minority Classes in the Data Set**					
LED_50-44-5-1	**20%**	**INTER**	0.2857	0.2979	0.2865	0.2883	0.2961	0.3158
		SCUT-DS	0.4944	0.5459	0.5170	0.5453	0.4825	0.5457
		SCUT-DS++	0.5798	0.5848	0.5923	0.5838	0.5957	0.5969
LED_55-30-10-5	20%	**INTER**	0.4208	0.4188	0.4177	0.4170	0.4329	0.4437
		SCUT-DS	0.6365	0.6505	0.6380	0.6491	0.6325	0.6471
		SCUT-DS++	0.6878	0.6941	0.6934	0.6945	0.6824	0.6965
LED_75-15-5-5	20%	**INTER**	0.5205	0.5460	0.5304	0.5470	0.5273	0.5508
		SCUT-DS	0.6787	0.6906	0.6886	0.6905	0.6814	0.6894
		SCUT-DS++	0.7177	0.7312	0.7218	0.7331	0.7187	0.7346
LED_80-14-5-1	20%	**INTER**	0.4332	0.4511	0.4288	0.4422	0.4393	0.4512
		SCUT-DS	0.6148	0.6461	0.6327	0.6464	0.6147	0.6457
		SCUT-DS++	0.6811	0.6694	0.6808	0.6698	0.6678	0.6760
LED_32-30-23-10-5	20%	**INTER**	0.4903	0.5025	0.4743	0.5000	0.4642	0.5008
		SCUT-DS	0.6238	0.6338	0.6265	0.6358	0.6183	0.6339
		SCUT-DS++	0.6726	0.6907	0.6840	0.6910	0.6731	0.6939
LED_50-34-10-5-1	20%	**INTER**	0.2759	0.2686	0.2636	0.2668	0.2874	0.2931
		SCUT-DS	0.5551	0.5935	0.5597	0.5918	0.5618	0.5961
		SCUT-DS++	0.6183	0.6241	0.6221	0.6239	0.6266	0.6253
LED_69-15-10-5-1	20%	**INTER**	0.4541	0.4725	0.4390	0.4635	0.4514	0.4741
		SCUT-DS	0.6146	0.6366	0.6200	0.6361	0.6093	0.6343
		SCUT-DS++	0.6542	0.6655	0.6632	0.6655	0.6631	0.6695

It may be inferred that the three algorithms against the six classifiers have low accuracy at 20% noise level in the data set, although their level of resiliency varies. The best result in the chart was obtained with the LED-75-15-5-5 data set. Recall from our discussion in Chapter 3 about not using cleaning tools because new concepts may be regarded as noise in a chunk. It may be said that the non-removal of noise was responsible for the drop in average recalls of the algorithms.

5.2.1.2.3 LED Data Set With 30% Noise

The result of the accuracy of the classifiers against the three algorithms may be seen to plunge further with increase in noise level to 30% in Table 14. The three algorithms are intolerant of noise, although SCUT-DS++ had the best accuracy amongst the three and INTER the worst accuracy. The best accuracy was observed in this result is with LED-75-15-5-5, similar to the observation in the tables for the 10% and 20% noise level above. Recall the discussion about the effect of the non-removal of noisy data in Section 5.2.1.2.2 above. Hence, the same reasons given in the analyses above are applicable here.

Table 14. Result of LED Data sets with 30% Noise.

		Classifier Used	**HT**	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**	**OzaBoost-NB**
Data set	**Noise Level**	**Algorithm**	**Average Recalls of Minority Classes in the Data set**					
LED_50-44-5-1	30%	INTER	0.0670	0.0842	0.0518	0.0820	0.0525	0.0977
		SCUT-DS	0.3080	0.3483	0.3037	0.3448	0.3012	0.3499
		SCUT-DS++	0.3392	0.3976	0.3435	0.3967	0.3476	0.3996
LED_55-30-10-5	30%	INTER	0.1475	0.1563	0.1461	0.1548	0.1547	0.1608
		SCUT-DS	0.4145	0.4423	0.4201	0.4424	0.4188	0.4393
		SCUT-DS++	0.4785	0.4943	0.4878	0.4951	0.4685	0.4928
LED_75-15-5-5	30%	INTER	0.1697	0.1946	0.1508	0.1926	0.1823	0.2201
		SCUT-DS	0.4678	0.4944	0.4785	0.4941	0.4721	0.4960
		SCUT-DS++	0.5136	0.5308	0.5203	0.5299	0.5143	0.5304
LED_80-14-5-1	30%	INTER	0.1253	0.1615	0.1168	0.1611	0.1568	0.1955
		SCUT-DS	0.4337	0.4637	0.4195	0.4611	0.4150	0.4591
		SCUT-DS++	0.4497	0.4770	0.4531	0.4747	0.4403	0.4782
LED_32-30-23-10-5	30%	INTER	0.1252	0.1189	0.1076	0.1194	0.1174	0.1204

		Classifier Used	**HT**	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**	**OzaBoost-NB**
Data set	**Noise Level**	**Algorithm**	**Average Recalls of Minority Classes in the Data set**					
		SCUT-DS	0.3870	0.4076	0.3889	0.4076	0.3878	0.4030
		SCUT-DS++	0.4546	0.4782	0.4542	0.4785	0.1891	0.1158
LED_50-34-10-5-1	30%	INTER	0.1074	0.1057	0.0976	0.1046	0.1058	0.1061
		SCUT-DS	0.3562	0.3905	0.3527	0.3935	0.3457	0.3944
		SCUT-DS++	0.4042	0.4205	0.4017	0.4196	0.4005	0.3997
LED_69-15-10-5-1	30%	INTER	0.1947	0.2091	0.1754	0.2070	0.1730	0.2125
		SCUT-DS	0.4039	0.4389	0.4131	0.4367	0.4180	0.4428
		SCUT-DS++	0.4311	0.4482	0.4308	0.4468	0.4247	0.4476

It may be concluded that SCUT-DS++ shows better ability to generalize on incoming streams than INTER. The accuracy of SCUT-DS may be said to be comparable to that of SCUT-DS++. Thus the relationship between recent chunk, past accumulated chunk and incoming chunk may be said to be a major determinant to the generalization ability of SCUT-DS and SCUT-DS++. Also, it may be said that NB and ensembles with NB as base learner have higher recognition rate of the minority instances than their HT counterparts. Recall our discussion about discriminative models and generative models. We discuss the results of the experiments on the real data sets next.

5.2.2 Results against Real Data

In this section, we present the results of the real data sets. The weather data set results are presented first, followed by that of the NYC Taxi Fare data set.

5.2.2.1 Weather Data set

Recall that the naming convention for the data set is the concatenation of the classes in the data sets. The proportion of the various classes in the data sets is written below the data set's name, in order to reflect the imbalance proportion. From Table 15, it may be noted that the SCUT-DS and SCUT-DS++ algorithms have high accuracies on this data set generally. The average minority recalls of SCUT-DS and SCUT-DS++, as expected, were higher than that of INTER. INTER had higher accuracies on some data sets, although it was not the overall best. In this particular experiment, there where situations where SCUT-DS had higher accuracy than SCUT-DS++ and INTER had slightly close or slightly higher accuracy. But overall, SCUT-DS and SCUT-DS++ performed better than INTER on the Weather data sets. Thus, it may be inferred that our resampling approach improved the recognition rates of the minority classes in the Weather data sets.

Table 15. Results of Experiments on Weather Data sets

	Classifier Used	**HT**	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**	**OzaBoost-NB**
Data set	**Algorithm**	**Average Recalls of Minority Classes in the Data set**					
Cloudy-Fog-Snow	**INTER**	0.9016	0.8638	0.9084	0.8678	0.9155	0.9101
	SCUT-DS	0.9213	0.8974	0.9267	0.8972	0.9344	0.9220
	SCUT-DS++	0.9354	0.8753	0.9359	0.8754	0.9447	0.9179
Clear-Fog-Ice	**INTER**	0.5464	0.5711	0.5162	0.5706	0.5637	0.6039
	SCUT-DS	0.8261	0.7797	0.8593	0.7774	0.7959	0.7928
	SCUT-DS++	0.8114	0.7189	0.8236	0.7181	0.8142	0.7473
Cloudy-Snow-Ice	**INTER**	0.7662	0.8307	0.7554	0.8312	0.7846	0.8226
	SCUT-DS	0.8729	0.8652	0.8772	0.8646	0.8848	0.8749
	SCUT-DS++	0.8695	0.8265	0.8641	0.8262	0.8651	0.8345
Cloudy-Clear-Fog	**INTER**	0.9642	0.9798	0.9837	0.9805	0.9863	0.9844
	SCUT-DS	0.9883	0.9772	0.9876	0.9779	0.9883	0.9824
	SCUT-DS++	0.9883	0.9759	0.9883	0.9759	0.9889	0.9824
Cloudy-Clear-Snow	**INTER**	0.9058	0.9238	0.9090	0.9238	0.9211	0.9273
	SCUT-DS	0.9162	0.9233	0.9201	0.9233	0.9300	0.9277
	SCUT-DS++	0.9253	0.9211	0.9292	0.9211	0.9361	0.9280
Cloudy-Clear-Ice	**INTER**	0.2227	0.2565	0.2086	0.2619	0.2194	0.3060
	SCUT-DS	0.6599	0.6474	0.7090	0.6515	0.7065	0.6632

	Classifier Used	HT	NB	OzaBag-HT	OzaBag-NB	OzaBoost-HT	OzaBoost-NB
Data set	Algorithm	Average Recalls of Minority Classes in the Data set					
	SCUT-DS++	0.6490	0.5712	0.7052	0.5670	0.6803	0.5733
Cloudy-Clear-Snow-Ice	**INTER**	0.5559	0.5668	0.5808	0.5653	0.5565	0.5964
	SCUT-DS	0.7572	0.7606	0.7668	0.7610	0.7732	0.7664
	SCUT-DS++	0.7844	0.7320	0.7898	0.7325	0.7823	0.7408
Cloudy-Snow-Ice-Fog	**INTER**	0.8019	0.7991	0.8101	0.7993	0.8112	0.8183
	SCUT-DS	0.8831	0.8497	0.8858	0.8489	0.8730	0.8686
	SCUT-DS++	0.8645	0.8294	0.8641	0.8296	0.8650	0.8561
Cloudy-Clear-Snow-Ice-Fog	**INTER**	0.6368	0.6225	0.6432	0.6232	0.6434	0.6506
	SCUT-DS	0.8182	0.7814	0.8285	0.7784	0.8131	0.7807
	SCUT-DS++	0.8495	0.7523	0.8517	0.7518	0.8307	0.7759

When comparing accuracy between SCUT-DS and SCUT-DS++, SCUT-DS outperformed SCUT-DS++. Although there were times when the accuracy of SCUT-DS++ were higher, such as in the Cloudy-Fog-Snow data set against HT, OzaBag-HT and OzaBoost-HT. There were some other data sets in the table that exhibited similar trend. It may be noticed that the classifiers at this points were mostly HT and HT-based classifiers. We may infer that the examples in the training set covered more of the classes' data space than the conditional probability of NB and NB-based ensembles. In the data sets where SCUT-DS performed better, it may be concluded that synthetic instances generalized well than the accumulated past examples. In these cases the recent chunk has more semblances with the incoming streams.

When comparing the accuracy against the six classifiers, HT and OzaBag-HT have comparable accuracies. While the accuracies of OzaBoost-HT improved, it is comparable to and sometimes better than those of the NB and NB-based ensembles. Recall the discussion about discriminatory and generative models in Chapter 2. It may be said that HTs being a discriminatory model in this case have examples that were able to establish a better boundary than the conditional probability of NB. Recall, the characteristics of NB classifiers discussed in Chapter 2 and Section 5.1.2 about the assumption of conditional independence. Thus, it may be inferred from the accuracies of the NB and NB-based classifiers that the assumption of conditional independence amongst the attributes of the Weather data sets may not hold.

In general, accuracy was seen to reduce whenever the data set contains instances from the Ice label. This means that of all the minority classes present, instances of the Ice labels are the most difficult to classify. Recall from our discussion in Chapter 3, about factors that make some minority classes difficult to classify. In the case of the Ice class, this may be inferred to be the result of its class overlapping with other classes. Moreover, it has a smaller representation compared with the other labels. Cluster-based under-sampling of the majority in SCUT-DS and SCUT-DS++ may have been said to aid the reduction in overlapping.

5.2.2.2 NYC Taxi Fare Data sets

The result performed on this data set is discussed in Table 16. As expected SCUT-DS and SCUT-DS++ performed better than INTER in most cases. INTER's accuracy was better than that of SCUT-DS++ with the NYC_1234b data set with the NB and OzaBag-NB classifiers. This may be said to be caused by the instances selected by the clustering algorithm as the training data. The choice of instances in the training set is highly dependent on the clustering algorithm.

Table 16. Results of Experiments on NYC Taxi-Fare Data set.

NYC TRIP FARE	**Classifier Used**	**HT**	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**	**OzaBoost-NB**
Data set	**Algorithm**	**Ave-Min Recall**	**Ave-Min Recall**	**Ave-Min Recall**	**Ave-Min Recall**	**Ave-Min Recall**	**Ave-Min Recall**
NYC_123a	**INTER**	0.9834	0.8496	0.9846	0.8755	0.9941	0.9505
(75-15-10)	**SCUT-DS**	0.9891	0.8967	0.9895	0.8969	0.9993	0.9567
	SCUT-DS++	0.9913	0.8292	0.9901	0.8238	0.9996	0.9460

NYC TRIP FARE	Classifier Used	HT	NB	OzaBag-HT	OzaBag-NB	OzaBoost-HT	OzaBoost-NB
Data set	**Algorithm**	**Ave-Min Recall**	**Ave-Min Recall**	**Ave-Min Recall**	**Ave-Min Recall**	**Ave-Min Recall**	**Ave-Min Recall**
NYC_123b	**INTER**	0.9986	0.7465	0.9988	0.7529	0.9960	0.8630
(50-45-5)	**SCUT-DS**	0.9966	0.9483	0.9980	0.9543	0.9996	0.9628
	SCUT-DS++	0.9964	0.9616	0.9976	0.9624	0.9978	0.9653
NYC_1234a	**INTER**	0.9063	0.6720	0.9088	0.6718	0.9251	0.8422
(50-44-5-1)	**SCUT-DS**	0.9970	0.8239	0.9969	0.8247	0.9967	0.8863
	SCUT-DS++	0.9937	0.8196	0.9963	0.8220	0.9969	0.8649
NYC_1234b	**INTER**	0.9666	0.8063	0.9723	0.8224	0.9893	0.9280
(75-15-5-5)	**SCUT-DS**	0.9790	0.8648	0.9797	0.8742	0.9996	0.9319
	SCUT-DS++	0.9860	0.7981	0.9869	0.7928	0.9999	0.9297
NYC_12345	**INTER**	0.9153	0.7500	0.8738	0.7234	0.9787	0.8821
	SCUT-DS	0.9967	0.8426	0.9956	0.8455	0.9962	0.8911
	SCUT-DS++	0.9879	0.7715	0.9925	0.7754	0.9985	0.8370

The accuracies of the three algorithms were observed to be close with the NYC_123a and NYC_123b data set against all the three algorithms and the six classifiers. The best percentage improvement was observed with the NYC_1234a data set against NB and NB-based ensembles.

Almost the same trend similar to the Weather data set was observed when comparing the accuracy between SCUT-DS and SCUT-DS++.

Considering the accuracy of the six classifiers, HT and HT-based ensemble classifiers outperformed the NB and NB-based classifiers. The same trends observed between the HT-based classifiers and the NB-based classifier in the Weather data set were observed in the NYC Taxi Fare data set. Thus, the same reason given for the Weather data set may be inferred. In most cases OzaBoost-HT had the best accuracies, while the accuracies of HT and OzaBag-HT were comparable. It may be said that the training sample used in OzaBoost-HT were more diverse, thus the reason for the higher accuracy when compared with OzaBag-HT. Sometimes, the accuracies of HT in SCUT-DS and SCUT-DS++ were comparable and sometimes lower than that of OzaBoost-HT. It may be inferred that the training samples are not so diverse. Of the three NB-based classifiers used, OzaBoost-NB had the best accuracies and this may be due to the ability to build more diverse models for prediction.

In conclusion, the results, as shown from Tables 10, indicate that the results obtained by the SCUT-DS and SCUT-DS++ methodologies are better than INTER. Most importantly when the proportion of the minority classes in the data set are very low, SCUT-DS and SCUT-DS++ have around 200% improvement in accuracy when compared with INTER. The results, therefore, indicates that our resampling approaches improved the recognition rates of the minority classes in the data sets. The clustering-based under-sampling approaches and the SMOTE-based oversampling methods aided in selecting relevant training data for resampling. Cluster-based under-sampling of the majority instances reduces overlapping between instances. This improved accuracies of SCUT-DS and SCUT-DS++ across all the six classifiers used.

Hence, our methodologies, SCUT-DS and SCUT-DS++ may be said to have improved the average recognition rates of the minority classes in multi-class imbalanced data stream classification. The algorithms addressed the factors that make the classification of imbalanced data set difficult. These factors were discussed in Chapter 3. Generally, between-class imbalance was addressed by combining resampling approaches. Flexibility of the algorithm was improved by sampling and using the average of the total instances in the classes as the resampling distribution rate. The issue of disjuncts or within-class imbalanced was solved by using cluster-based under-sampling and SMOTE-based oversampling. Cluster-based under-sampling was also used to address partly the problem of overlapping between the classes, because it reduces the examples across the boundaries. Cluster-based under-sampling of the minority classes in SCUT-DS++ was used to address the problem of overwhelming the system by accumulating too many positive instance. The issue of noise was not addressed because of the reason given earlier in Chapter 3. Recall from our discussion that regarding an example as noise may be difficult because it may be a new concept that needs to be learn.

Unfortunately, a definite ordering of the six classifiers could not be arrived at because, given the accuracies in the experimental results, these accuracies do not follow a definite pattern that may make ordering possible.

To conclude this section, we present and interpret four interesting rules obtained from the minority classes in the Weather data sets.

1. (Temperature <= -16.40) and (Temperature <= -22.50) and (Visibility >= 9.70) and (RelativeHumidity >= 71.00) and (WindSpeed <= 5.98) and (WindSpeed <= 8.98) and (WindDirection >= 21) => Class=Ice

The rule above states that Ice is likely to occur when the temperature is lower than -16.40 and -22.50 Celsius (C), the Visibility is farther than 9.70 kilometer (km), the Relative Humidity is quite high, above 71%, the Wind Speed is below 5.98 and 8.98 km/h, and the Wind Direction is greater than 21 degrees.

2. (Temperature <= -16.50) and (Temperature <= -24.80) and (Visibility >= 9.70) and (WindDirection >= 9.01) and (RelativeHumidity >= 68.02) and (StationPressure <= 100.42) and (StationPressure <= 99.97) and (WindSpeed >= 9.00) and (WindSpeed >= 15.03) => Class=Ice

This rule, is similar to the first rule, it considers the Temperature, Visibility, Relative Humidity and Wind Speed. In contrast to the first rule, it also takes into consideration the Station Pressure. Ice is predicted to happen when the Temperature falls below 24.80 C, the Wind Direction is above 9.01 degrees, the Wind Speed is faster than 9.00 and 15.03 km/h, and the Station Pressure is below 100.42 and 99.97 kilo Pascal (kPa).

3. (WindSpeed >= 22.00) and (Visibility <= 11.30) and (Visibility <= 3.2) and (WindChill <= -9.97) and (DewPoint >= -25.55) => Class=Snow

This rule indicates that higher Wind Speed, lower Visibility, with a Wind Chill factor of less than -10C, and a Dew Point greater than or equal to -25.55 C, may be indicative of Snow.

4. => Class=Fog

This rule implies that all instances that do not fall into any of the categories in the listed rules should be classified as belonging to the Class “Fog”.

Therefore it may be concluded from these rules, these attributes namely, temperature, Visibility, Relative Humidity, Wind Speed, Wind Direction and Station Pressure are important in order to predict instances of Class “Ice”. The relevant attributes for predicting instances of the Class "Snow" include Wind Speed, Visibility, Wind Chill and Dew Point. The Class “Fog” instances are quite different from the other classes, because according to its rule, if it does not belong to any of the class then it means that it belongs to Class “Fog”. We discuss the statistical significant testing of the result of the experiments in the tables above in the next section.

5.3 Statistical Evaluation

We now analyze the statistical significance testing of the results of the experiments in the tables above. The results are analyzed from three perspectives. The first statistical significance testing involves contrasting the results obtained amongst the six classifiers used. Secondly, we contrast INTER, SCUT-DS and SCUT-DS++. While the last test is the testing for statistical significance between the results obtained from SCUT-DS and SCUT-DS++ to determine the technique that provide superior results, in terms of generalization.

The statistical significant testing is done so as to ascertain that the result is not attributed to chance. The first two statistical significant testing will be done using Friedman's test while the last test will be conducted with Wilcoxon's test. Recall that the Friedman's test is used to indicate if significance difference is observed amongst all elements under consideration. A post-hoc test is conducted if there is a significance difference to determine where the significance difference exists amongst pairs. The results of the statistical significant testing are presented in Table 17 below.

Table 17. Results of the Statistical Significance Evaluation

Statistical Testing Category	Wave21	Wave40	LED_10% _Noise	LED_20% _Noise	LED_30% _Noise	Weather	NYC Taxi Fare
Classifiers	1.05E-12	9.60E-13	1.02E-10	4.47E-10	6.74E-12	9.15E-09	4.47E-13
3-Algorithms	2.00E-11	4.58E-08	2.20E-16	2.20E-16	2.20E-16	1.90E-12	8.87E-06
SCUT-DS vs SCUT-DS++	0.3555	0.08503	2.37E-05	4.69E-05	0.006804	0.5845	0.5058

The p-values obtained from the statistical testing of significance of the experimental results are presented against the data set and the statistical testing category. We now draw conclusion on this table based on the statistical evaluation criteria discussed in Section 4.4.2. First, from the classifiers' perspective, and from the results in the table, it may be agreed that the alternate hypothesis should be accepted because the p-values in those rows are all smaller than 0.05. Therefore it may be concluded that, based on the results, there exist significant differences between the accuracy of the classifiers used.

Secondly, according to the p-values in the table, it may be concluded from the category of the 3-algorithms, INTER, SCUT-DS and SCUT-DS++ that there exist significant differences amongst these 3-algorithms for each data set. Hence, the alternate hypothesis is accepted.

Thirdly, the statistical significance testing between SCUT and SCUT++ reveals that for the three LED data sets, the null hypothesis should be rejected, while in the Waveform and real data sets, the null should be accepted.

Readers are directed to Appendix B for the results of the Conover pair-wise test. The analysis of the Conover pair-wise test is reported below. The evaluations are given as a pair between a methodology and the others.

The Conover pair-wise test on the experimental results of the Wave21 data sets revealed that there are significant differences between all the pairs of the classifiers except between NB and OzaBag-NB. Also no significant difference was observed between OzaBag-HT and OzaBoost-HT.

According to the post-hoc test on the result of experimentation of Wave40, pair-wise comparison between OzaBag-HT and OzaBoost-HT shows no significant difference. While there were significant differences between other classifier pairs.

The Conover post-hoc test on the experimental results of the LED_10% data sets provided the evidence of the existence of significant difference (at $p < 0.05$) between the mean ranks of the pair-wise comparison of all the classification algorithms. The exception to this was noticed between NB and OzaBag-NB, NB and OzaBoost-NB, HT and OzaBoost-HT.

The post-hoc test on LED_20% revealed that there exist significant differences between the pair-wise comparison of all classifiers with the exclusion of HT and OzaBag-HT, HT and OzaBoost-HT, NB and OzaBag-NB.

In the post-hoc test with the LED_30%, it was observed that there was a significant difference between all pairs with the exception of HT and OzaBag-HT, HT and OzaBoost-HT, NB and OzaBag-NB, NB and OzaBoost-NB.

Also, the analysis of the result of the Conover post-hoc test on the Weather data set results shows that there is a significant difference when all the six classification algorithms are paired. The exceptions to this include the following pairs, NB and OzaBag-NB, OzaBag-HT and OzaBoost-HT, OzaBag-HT and OzaBoost-NB.

The post-hoc test on the results of the NYC-Taxi-Fare data sets revealed that all the pairs of classification algorithms exhibit significant differences except HT and OzaBag-HT,

We now discuss the result of the post-hoc test on all the data sets used for experimenting from the perspective of the three methodologies used in this study. The results revealed that significance difference was observed in all the data sets and in all the pairs based on the results of the pair-wise Conover test in Appendix B.

5.4 Discussion and Lessons Learned

Our goal in this thesis was to improve the recognition rate of the minority classes in multi-class imbalanced data stream. The performance of imbalance data classification is generally improved by increasing the presence of the minority classes in the training set. In sampling, this may be done by reducing the presence of the majority class or increasing the presence of the positive instances to achieve a balanced training set. There are various ways through which this may be achieved as discussed in Chapter 2.

We saw the need to accumulate some past minority example rather than generating synthetic examples to augment the minority labels which may result in forgetting some concept learned earlier. This may cause the model built not to be able to generalize on incoming stream. We also observed that the criterion for selecting from these minority instances need to be determined with care, because accumulating too much data may have a negative impact on system performance. Thus, we introduced clustering based under-sampling for the positive classes for selecting relevant training instances across almost all the classes' data spaces. This observation led to the development of the second methodology SCUT-DS++, which had higher average recalls on some data sets when compared to SCUT-DS. The reason for this is that clustering is highly dependent on the semblance between the examples in the recent windows and accumulated windows.

In testing the algorithms designed, we learned that the two algorithms may tolerate noise to a certain extent. With the Waveform data sets, SCUT-DS and SCUT-DS++ may be said to have comparable accuracies, thus the introduction of the irrelevant attributes in Wave40 data set did not really affect the average recalls. However, the increase in the level of noise in LED data set, caused the accuracy of all the three algorithms to drop. The tolerance of SCUT-DS and SCUT-DS++ at 10% may be said to be within acceptance range. But the resilience at 20% or 30% may be said to be unacceptable. Thus the lesson learned is that the methodologies are heavily affected by increase in noise level, rather than by the introduction of irrelevant attributes.

We also studied the effect of the choice of classifier used for the classification task in multi-class imbalance stream learning. NB and NB-based meta-learners showed a higher recall than their HT counterparts with the synthetic data set, while it is the other way with the real data sets. Recall our discussion about generative and discriminative models. NB is a generative model that is based on conditional probability and was able to generalize better compared to HT, a discriminative model that relies solely on the instances in its training set. In the cases where HT related classification algorithms outperformed NB, it may be said that such data set do not have wide data space. Thus, with a few seen examples the HT classifiers were able to generalize well over the incoming streams. We were able to confirm, as expected, that the performances of classification algorithms are dependent on the characteristics of the data sets. Further the condition of independence among attributes in NB, may also affect the performance.

5.5 Summary

In this chapter, we presented and discussed the results of our experimental evaluations. Our research aim was to develop methodologies that improve the average recalls of the minority labels of the multi-class imbalance stream. Our results indicate that we did indeed, obtain higher average recalls with our methodologies compared with INTER. We also observed that the three algorithms were not tolerant to increasing noise levels, their accuracies plummeted with increase in the percentage of noise. We were able to show that in non-evolving stream, past examples may generalize better than synthetic instances. The generalization ability of SCUT-DS and SCUT-DS++ may thus be said to be highly dependent on the incoming stream. In addition we showed that the accuracy of classifiers and ensembles with similar base classifiers are comparable.

The conclusion that may be drawn from these results is that SCUT-DS and SCUT-DS++ consistently improves the recall of the minority classes. In addition, in cases where the accuracy of INTER is very close or slightly higher than SCUT-DS++, this is may be caused by loss of information due to cluster-based under-sampling.

Our results suggest some reasons for the improvement in SCUT-DS and SCUT-DS++. The first reason is that the resultant training set always contains balanced training data across all classes present. The second is the use of synthetic or accumulated past minority instances to augment the presence of the minority instances in the training set. Recall that we assumed the stream to be non-evolving, with no drift in data distribution, thus past instances may not have detrimental effect on predicting incoming streams. The third is the use of cluster-based under-sampling, which reduces overlapping between classes and prevents within-class imbalance.

The next chapter concludes the study. It provides a general conclusion to the thesis by discussing the contribution in this thesis and future work.

Chapter 6

Conclusion and Future Work

In the previous chapter, we presented the experimental design of the methodologies in this study and analyzed the results of our extensive experimental evaluations.

In this thesis, the focus was to develop a methodology for learning from multi-class imbalanced data stream. To this end, we extended the existing SCUT algorithm from a static to the streaming environment, with the aim of yielding high recognition rates for minority instances. Experiments were performed in order to compare the performance of the two algorithms in this thesis against a benchmarking algorithm, in terms of the average recall of all minority classes in the data stream. The impacts of the choice of base learners, or classifiers, on the designed methodologies were monitored. In addition, the resilience of the algorithms to noise was investigated. Furthermore, comparisons between the ability of the two approaches, SCUT-DS and SCUT-DS++ to generalize on incoming streams were made.

In this chapter we draw a general conclusion to the thesis by discussing the contribution of the thesis and making suggestions for future work.

6.1 Thesis Contribution

Existing studies do not directly address multi-class imbalanced data stream classification. The developed methodologies in this study were intended to address the research gap in imbalanced data set classification, more specifically in multi-class imbalanced classification in the non-stationary environment. SCUT was designed for multi-class imbalanced data set in static environment, thus the need for extending to the streaming environment as researched in this thesis.

The two approaches developed in this study, which actualized our main contribution, led to the improvement in the recognition rates of the instances of the minority classes by standard classifiers through the use of our resampling technique. This enabled the direct prediction of the minority instances in multi-class imbalanced data stream, without class reduction. We focused on improving the average recalls of all minority classes present in the data set because traditional classifiers have high recall on the majority labels.

Thus, the first contribution, which is the main contribution, is to extend SCUT from static environments to streaming environments. Secondly, most of the methods in literature used to address imbalanced data set decompose the data set; these approaches may introduce bias to the result. The sampling approaches used in SCUT-DS and SCUT-DS++ directly addressed multi-class imbalanced data stream classification without resorting to class decomposition. This eliminated the bias introduced when computing the results.

Furthermore, the direct multi-class imbalanced classification algorithms for data streams in literature accumulated too much minority instances that may over overburden the system. The third contribution was the use of cluster-based under-sampling of the minority instances and the generation of synthetic minority examples to address this issue. The synthetic and accumulated past minority instances were used to increase the data space of the minority classes. The accumulation of past minority instances prevented some earlier encountered instances from being forgotten and the reliance on only the recent window. Thus aiding the generation and selection of relevant training sets that will generalize on incoming data stream via cluster-based under-sampling.

Moreover, the use of the average of instances of the classes in the window as the sampling rate prevented excessive oversampling or under-sampling, it also reduced the computational time and resources when sampling methods are used in highly imbalanced dataset. Recall the discussion from Chapter 3 about the drawback of some methodologies that directly addressed multi-class imbalanced learning in data stream which were said to be unsuitable for highly imbalanced data sets. The adopted resampling rate removes the need to compute imbalance ratio.

Finally, the flexibility of the multi-class imbalanced algorithm is addressed because the need to assign weight, cost or the requirement for domain knowledge are eliminated.

The results of the experimentation indicates that SCUT-DS and SCUT-DS++ successfully increased the recognition rates of the minority instances in the data stream. Typically, we have higher average recalls when compared with INTER, although this was achieved in longer time. However, as evident in the individual recalls of the minority classes in Appendix B, the individual minority recalls were higher.

6.2 Future Work

The approach in this study may be extended in order to improve further on our methodologies and to address multi-class imbalanced data stream learning in other domains. Thus, the next few points explain the areas that may be considered for improvement.

We have experimented with three groups of learning algorithms, namely, HT, NB and meta-learners; however, extending the study to other classification algorithms will help to observe the behavior of the other classification algorithms on the SCUT-DS methodologies. We also believe this work may be extended by using other clustering algorithms apart from K-means. Experimenting with more data streams shows the flexibility and robustness of a methodology, thus the need to experiment with more data stream from other domains.

It was observed in the results of the experiments that the individual recall of some classes of minority instances were low compared to others. Hence, in a future study, the optimal sampling ratio that will be beneficial to all minority classes present in the data stream based on the distribution of instances per class in the data stream may be investigated.

Another possible extension is in terms of the evaluation method used. The evaluation technique used in this project was on chunk-by-chunk bases, which could be further studied by extending the resampling and model updating approach to the instance-by-instance scenario. Thus, this would provide the option of chunk-by-chunk and instance-by-instance evaluation.

Considering that this thesis focused on only the recall of the minority instances, we will like to extend performance evaluation to the majority instances in future studies. In our implementation, we assumed that we always have a preponderance of the majority instances, thus, we under-sampled the majority instances. However, in future works, it would be worth investigating how to resolve situations where majority classes may become minority classes in some windows.

As an extension to this study, which has focused on non-evolving streams, it would be worth conducting a similar research on evolving data streams. Evolving data streams are data streams with concept drift where the underlying concept may changes. Hence, instances in a recent window may not be applicable to predicting incoming streams. Relying on models built from the recent chunks only, as done in SCUT-DS, or accumulating minority instances from earlier chunks in SCUT-DS++ may be detrimental to learning. Therefore, in future studies the goal might be to experiment on the optimal technique to use in resampling the training set in evolving data streams.

References

[1] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[2] S. Jang *et al.*, “Structural Health Monitoring of A Cable-stayed Bridge Using Smart Sensor Technology: Deployment and Evaluation,” *Smart Structures and Systems*, vol. 6, no. 5–6, pp. 439–459, 2010.

[3] E. Jovanov, A. Milenkovic, C. Otto, and P. C. de Groen, “A Wireless Body Area Network of Intelligent Motion Sensors for Computer Assisted Physical Rehabilitation,” *Journal of NeuroEngineering and Rehabilitation*, vol. 2, no. 1, pp. 1–10, 2005.

[4] S. Patel *et al.*, “Monitoring Motor Fluctuations in Patients With Parkinson’s Disease Using Wearable Sensors,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 6, pp. 864–873, 2009.

[5] J. Han and M. Kamber, *Data Mining Concepts and Techniques*, 2nd Editio. Elsevier Inc., 2006.

[6] N. V Chawla, N. Japkowicz, and A. Kotcz, “Editorial : Special Issue on Learning from Imbalanced Data Sets,” *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 1–6, 2004.

[7] S. Wang and X. Yao, “Multiclass Imbalance Problems: Analysis and Potential Solutions.,” *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, vol. 42, no. 4, pp. 1–13, 2012.

[8] A. Agrawal, H. L. Viktor, and E. Paquet, “SCUT : Multi-Class Imbalanced Data Classification using SMOTE and Cluster-based Undersampling,” *Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K), 2015 7th International Joint Conference. IEEE*, vol. 1, no. November, pp. 226–234, 2015.

[9] K. Wu, A. Edwards, W. Fan, J. Gao, and K. Zhang, “Classifying Imbalanced Data Streams via Dynamic Feature Group Weighting with Importance Sampling.,” *Proceedings of the 2014 SIAM International Conference on Data Mining*, no. April, pp. 722–730, 2014.

[10] H. Guo and H. L. Viktor, “Learning from Imbalanced Data Sets with Boosting and Data Generation : The DataBoost-IM Approach,” *ACM SIGKD Explorations Newsletter - Special Issue on Learning from Imbalanced Datasets*, vol. 6, no. 1, pp. 30–39, 2004.

[11] N. Thai-Nghe, Z. Gantner, and L. Schmidt-Thieme, “Cost-sensitive Learning Methods for Imbalanced Data,” *Proceedings of the International Joint Conference on Neural Networks (IJCNN). IEEE*, no. July, pp. 1–8, 2010.

[12] B. Mirza, Z. Lin, and K. A. Toh, "Weighted Online Sequential Extreme Learning Machine for Class Imbalance Learning," *Neural Processing Letters*, vol. 38, no. 3, pp. 465–486, 2013.

[13] S. Wang, L. L. Minku, and X. Yao, "Dealing With Multiple Classes in Online Class Imbalance Learning," *IJCAI International Joint Conference on Artificial Intelligence*, no. July, pp. 2118–2124, 2016.

[14] G. Ou and Y. L. Murphey, "Multi-class Pattern Classification Using Neural Networks," *Pattern Recognition*, vol. 40, no. 1, pp. 4–18, 2007.

[15] P. Sobhani, H. Viktor, and S. Matwin, "Learning from Imbalanced Data Using Ensemble Methods and Cluster-based Undersampling," *International Workshop on New Frontiers in Mining Complex Patterns. Springer, Cham*, no. September, pp. 69–83, 2014.

[16] J. Gao, B. Ding, W. Fan, J. Han, and P. S. Yu, "Classifying Data Streams With Skewed Class Distributions And Concept Drifts," *IEEE Internet Computing*, vol. 12, no. 6, pp. 37–49, 2008.

[17] N. V Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "SMOTEBoost: Improving Prediction of The Minority Class in Boosting," *Knowledge Discovery in Databases: PKDD*, vol. 2838, pp. 107–119, 2003.

[18] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[19] Dariusz Brzeziński, "MINING DATA STREAMS WITH CONCEPT DRIFT," 2010.

[20] J. Gama, R. Sebastião, and P. P. Rodrigues, "Issues in Evaluation of Stream Learning Algorithms," *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '09*, pp. 329–337, 2009.

[21] A. K. Jain, "Data Clustering: 50 Years Beyond K-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010.

[22] B. S. Everitt, S. Landau, M. Leese, and D. Stahl, *Cluster Analysis*, 5th Editio. West Sussex, PO19 8SQ, United Kingdom: John Wiley & Sons Limited, 2011.

[23] R. J. Kuo, L. M. Ho, and C. M. Hu, "Integration of Self-Organizing Feature Map and K-Means Algorithm for Market Segmentation," *Computers & Operations Research*, vol. 29, no. 11, pp. 1475–1493, 2002.

[24] C. Aggarwal, *Data streams: Models And Algorithms*. 2006.

[25] A. Bifet, R. Kirkby, P. Kranen, and P. Reutemann, *Massive Online Analysis Manual*. Centre for Open Software Innovation, The University of Waikato, 2012.

[26] I. Zliobaite, "Learning Under Concept Drift: An Overview," *Training*, vol. abs/1010.4, pp. 1–36, 2009.

[27] A. Pesaranghader and H. L. Viktor, "Fast Hoeffding Drift Detection Method for Evolving Data Streams," *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, vol. Springer I, no. September, pp. 96–111, 2016.

[28] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining Data Streams: A Review," *SIGMOD Record*, vol. 34, no. 2, pp. 18–26, 2005.

[29] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 1–16, 2002.

[30] J. S. Vitter, "Random Sampling With A Reservoir," *ACM Transactions on Mathematical Software*, vol. 11, no. 1, pp. 37–57, 1985.

[31] B. Babcock, M. Datar, and R. Motwani, "Sampling From a Moving Window Over Streaming Data," *Proceedings of 13th Annual ACMSIAM Symposium on Discrete Algorithms*, vol. 102, no. 2001–33, pp. 633–634, 2002.

[32] S. Chaudhuri, R. Motwani, and V. Narasayya, "On Random Sampling Over Joins," *ACM SIGMOD Record*, vol. 28, no. 2, pp. 263–274, 1999.

[33] B. Babcock, M. Datar, and R. Motwani, "Load Shedding Techniques for Data Stream Systems," *Proceedings of the 2003 Workshop on Management and Processing of Data Streams (MPDS)*, pp. 1–3, 2003.

[34] N. Tatbul and S. Zdonik, "Window-aware Load Shedding for Aggregation Queries Over Data streams," *Proceedings of the 32nd international conference on Very large data bases ()*, vol. 6, no. VLDB Endowment, pp. 799–810.

[35] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1794–1813, 2002.

[36] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning From Streaming Data With Concept Drift And Imbalance: An Overview," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.

[37] P. Domingos and G. Hulten, "Mining High-Speed Data Streams," *Proceedings of The Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 71–80, 2000.

[38] N. C. Oza, "Online Bagging and Boosting," *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 105–112, 2005.

[39] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New Ensemble Methods for Evolving Data Streams," *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '09*, p. 139, 2009.

[40] J. Gama, R. Sebastião, and P. P. Rodrigues, "Issues in Evaluation of Stream Learning Algorithms," *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '09*, no. January, pp. 329–337, 2009.

[41] H. He and E. A. Garcia, “Learning From Imbalanced Data,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[42] A. Bifet *et al.*, “MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering,” *HaCDAIS 2010*, vol. 11, no. September 2015, p. 3, 2010.

[43] J. Gao, W. Fan, J. Han, and P. S. Yu, “A General Framework For Mining Concept-drifting Data Streams With Skewed Distributions,” *Proceedings of the 2007 SIAM International Conference on Data Mining*, vol. April, no. Society for Industrial and Applied Mathematics, pp. 3–14, 2007.

[44] L. I. Kuncheva, “Classifier Ensembles for Changing Environments,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3077, no. June 2004, pp. 1–15, 2004.

[45] M. Mermillod, A. Bugaiska, and P. Bonin, “The stability-plasticity Dilemma: Investigating The Continuum From Catastrophic Forgetting To Age-limited Learning Effects.,” *Frontiers in Psychology*, vol. 4, p. 504, 2013.

[46] S. Wang, L. L. Minku, and X. Yao, “A Learning Framework for Online Class Imbalance Learning,” *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 36–45, 2013.

[47] G. E. a. P. a. Batista, R. C. Prati, and M. C. Monard, “A Study of The Behavior of Several Methods For Balancing Machine Learning Training Data,” *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, p. 20, 2004.

[48] H. He, Y. Bai, E. A. Garcia, and S. Li, “Adaptive Synthetic Sampling Approach for Imbalanced Learning,” *In Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference*, no. 3, pp. 1322–1328.

[49] “UCI Machine Learning Repository: Yeast Data Set.” [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Yeast?ref=datanews.io. [Accessed: 05-Nov-2017].

[50] C. X. Ling and C. Li, “Data Mining for Direct Marketing: Problems and Solutions,” *Fourth International Conference on Knowledge Discovery*, vol. 98, pp. 1–7, 1998.

[51] M. Kubat, R. C. Holte, and S. Matwin, “Machine Learning for The Detection of Oil Spills in Satellite Radar Images,” *Machine Learning*, vol. 30, no. 2–3, pp. 195–215, 1998.

[52] M. Galar, A. Fern, E. Barrenechea, and H. Bustince, “A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches,” *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS,* vol. 42, no. 4, pp. 463–484, 2012.

[53] S. L. Phung, A. Bouzerdoum, and G. H. Nguyen, “Learning Pattern Classification Tasks With Imbalanced Data Sets,” *Pattern Recognition*, pp. 193–208, 2009.

[54] S. S. Khan and M. G. Madden, “A Survey of Recent Trends In One Class Classification,” *Irish Conference on Artificial Intelligence and Cognitive Science.*, vol. August, no.

Springer Berlin Heidelberg., pp. 188–197, 2009.

[55] S. S. Khan and M. G. Madden, “One-Class Classification: Taxonomy of Study and Review of Techniques,” *The Knowledge Engineering Review*, vol. 29(3), no. January, pp. 345–374, 2014.

[56] N. Japkowicz, “Supervised Versus uUsupervised Binary-learning by Feedforward Neural Networks,” *Machine Learning*, vol. 42, no. 1–2, pp. 97–122, 2001.

[57] Y. Sun, M. S. Kamel, and Y. Wang, “Boosting for Learning Multiple Classes with Imbalances Class Distribution,” *Proceedings - IEEE International Conference on Data Mining, ICDM*, no. Sixth, pp. 592–602, 2006.

[58] B. Liu, Y. Xiao, L. Cao, and P. S. Yu, “Orientation Distance-based Discriminative Feature Extraction for Multi-class Classification,” *Proceedings of the 19th ACM international conference on Information and knowledge management - CIKM ’10*, p. 909, 2010.

[59] W. Y. Yang, S. X. Liu, T. S. Jin, and X. M. Xu, “An Optimization Criterion for Generalized Marginal Fisher Analysis on Undersampled Problems,” *International Journal of Automation and Computing*, vol. 8, no. 2, pp. 193–200, 2011.

[60] G. M. Weiss and F. Provost, “Learning When Training Data Are Costly: The Effect of Class Distribution on Tree Induction,” *Journal of Artificial Intelligence Research*, vol. 19, pp. 315–354, 2003.

[61] E. Lughofer, E. Weigl, W. Heidl, C. Eitzinger, and T. Radauer, “Fast and Economic Integration of New Classes On the Fly in Evolving Fuzzy Classifiers using Class Decomposition,” *Fuzzy Systems (FUZZ-IEEE), 2015 IEEE International Conference on*, no. IEEE, pp. 1–8, 2015.

[62] E. R. Faria, A. C. de Leon Ferreira Carvalho, and J. Gama, “MINAS: Multiclass Learning Algorithm for Novelty Detection in Data Streams,” *Data Mining and Knowledge Discovery*, vol. 30, no. 3, pp. 640–680, 2015.

[63] T. Jo and N. Japkowicz, “Class Imbalances versus Small Disjuncts,” *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 40–49, 2004.

[64] A. S. Nickerson, N. Japkowicz, and E. Milios, “Using Unsupervised Learning to Guide Resampling in Imbalanced Data Sets,” *In Proceedings of the Eighth International Workshop on AI and Statitsics*, p. 5, 2001.

[65] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, “Learning From Class-Imbalanced Data: Review of Methods and Applications,” *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017.

[66] A. Estabrooks, T. Jo, and N. Japkowicz, “A Multiple Resampling Method for Learning from Imbalanced Data Sets,” *Computational Intelligence*, vol. 20, no. November 1, pp. 18–36, 2004.

[67] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, “An Insight into

Classification with Imbalanced Data: Empirical Results and Current Trends on Using Data Intrinsic Characteristics," *Information Sciences*, vol. 250, pp. 113–141, 2013.

[68] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "RUSBoost: A Hybrid Approach To Alleviating Class Imbalance," *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, vol. 40, no. 1, pp. 185–197, 2010.

[69] "An Automated Prompting System for Smart Environments." [Online]. Available: https://www.slideshare.net/barnandas/barnan-das-icost2011talk. [Accessed: 16-Aug-2017].

[70] S. J. Yen and Y. S. Lee, "Cluster-based Under-sampling Approaches for Imbalanced Data Distributions," *Expert Systems with Applications*, vol. 36, no. 3 PART 1, pp. 5718–5727, 2009.

[71] "Handling Class Overlap and Imbalance to Detect Prompt Situations in S…." [Online]. Available: https://www.slideshare.net/barnandas/handling-class-overlap-and-imbalance-to-detect-prompt-situations-in-smart-homes. [Accessed: 16-Aug-2017].

[72] B. X. Wang and N. Japkowicz, "Boosting Support Vector Machines For Imbalanced Data Sets," *Knowledge and Information Systems*, vol. 25, no. 1, pp. 1–20, 2010.

[73] Y. SUN, A. K. . WONG, and M. S. KAMEL, "Classification of Imbalanced Data: a Review," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 4, pp. 687–719, 2009.

[74] C. X. Ling, Q. Yang, J. Wang, and S. Zhang, "Decision Trees with Minimal Costs," *Twenty-first International Conference on Machine Learning (ICML2004). ACM*, no. July, p. 69, 2004.

[75] C. Li, "Classifying Imbalanced Data Using a Bagging Ensemble Variation (BEV)," *Proceedings of the 45th annual southeast regional conference. ACM*, pp. 203–208, 2007.

[76] S. W. S. Wang and X. Y. X. Yao, "Diversity Analysis on Imbalanced Data Sets by Using Ensemble Models," *IEEE Symposium on Computational Intelligence and Data Mining*, vol. CIDM'09, no. March, pp. 324–331, 2009.

[77] S. Wang, L. L. Minku, and X. Yao, "Resampling-based Ensemble Methods for Online Class Imbalance Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 5, pp. 1356–1368, 2015.

[78] S. Ertekin, J. Huang, and C. L. Giles, "Active learning for Class Imbalance Problem," *Proceedings of the International ACM SIGIR conference on Research and Development in Information Retrieval*, pp. 823–824, 2007.

[79] T. Hoens and N. Chawla, "Learning In Non-stationary Environments With Class Imbalance," *Proceedings of the 18th ACM SIGKDD …*, pp. 168–176, 2012.

[80] G. Ditzler, R. Polikar, and N. Chawla, "An Incremental Learning Algorithm for Non-stationary Environments and Class Imbalance," *Proceedings - International Conference*

on Pattern Recognition, pp. 2997–3000, 2010.

[81] S. Chen and H. Haibo, “SERA: Selectively Recursive Approach towards Nonstationary Imbalanced Stream Data Mining,” *Neural Networks, 2009. IJCNN 2009. International Joint Conference. IEEE*, vol. 1, no. June 14-19, pp. 522–529, 2009.

[82] S. Chen, H. He, K. Li, and S. Desai, “MuSeRA : Multiple Selectively Recursive Approach towards Imbalanced Stream Data Mining,” *IEEE, International Joint Conference on Neural Networks (IJCNN)*, no. July, pp. 1–8, 2010.

[83] R. Savitha, S. Suresh, and H. J. Kim, “A Meta-Cognitive Learning Algorithm for an Extreme Learning Machine Classifier,” *Cognitive Computation*, vol. 6, no. 2, pp. 253–263, 2014.

[84] B. Mirza, Z. Lin, J. Cao, and X. Lai, “Voting Based Weighted Online Sequential Extreme Learning Machine for Imbalance Multi-class Classification,” *Proceedings - IEEE International Symposium on Circuits and Systems*, vol. 2015–July, pp. 565–568, 2015.

[85] N. V Chawla, “DATA MINING FOR IMBALANCED DATASETS : AN OVERVIEW,” *In Data Mining and Knowledge Discovery Handbook*, no. Springer US, pp. 875–886, 2009.

[86] R. P. Espíndola and N. F. F. Ebecken, “On Extending F-measure and G-mean Metrics to Multi-class Problems,” *WIT Transactions on Information and Communication Technologies*, vol. 35, 2005.

[87] J. Shotton, J. Winn, C. Rother, and A. Criminisi, “TextonBoost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3951 LNCS, pp. 1–15, 2006.

[88] N. Japkowicz and M. Shah, *Evaluating Learning Algorithms: A Classification Perspective.* Cambridge: Cambridge University Press, 2011.

[89] M. Kubat, R. Holte, and S. Matwin, “Learning When Negative Examples Abound,” *Proceedings of the 9th European Conference on Machine Learning*, pp. 146–153, 1997.

[90] A. Bifet and E. Frank, “Sentiment Knowledge Discovery in Twitter Streaming Data,” *In International conference on discovery science*, no. Springer, Berlin, Heidelberg., pp. 1–15, 2010.

[91] G. Krempl *et al.*, “Open Challenges For Data Stream Mining Research,” *ACM SIGKDD Explorations Newsletter*, vol. 16, no. 1, pp. 1–10, 2014.

[92] D. W. Zimmerman and B. D. Zumbo, “Relative power of the wilcoxon test, the friedman test, and repeated-measures ANOVA on ranks,” *Journal of Experimental Education*, vol. 62, no. 1, pp. 75–86, 1993.

[93] J. D. Perezgonzalez, “Fisher, Neyman-Pearson or NHST? A Tutorial For Teaching Data Testing.,” *Frontiers in Psychology*, vol. 6, p. 223, 2015.

[94] J. Demšar, “Statistical Comparisons of Classifiers over Multiple Data Sets,” *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[95] W. J. Conover, *Practical Nonparametric Statistics*, 3rd editio. New York: Wiley, 1999.

[96] T. Pohlert, “The Pairwise Multiple Comparison of Mean Ranks Package (PMCMR),” *R package*, p. 27, 2014.

[97] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA Data Mining Software,” *SIGKDD Explorations Newsletter*, vol. 11, no. 1, p. 10, 2009.

[98] “Historical Data - Climate - Environment and Climate Change Canada.” [Online]. Available: http://climate.weather.gc.ca/historical_data/search_historic_data_e.html. [Accessed: 26-Oct-2017].

[99] “NYC Taxi Trips Data.” [Online]. Available: http://www.andresmh.com/nyctaxitrips/. [Accessed: 26-Oct-2017].

[100] L. Breiman, J. . Friedman, R. . Olshen, and C. . Stone, *Classification and Regression Trees. Wadsworth International Group.* CRC Press., 1984.

Appendices

Appendix A

Experimental Results with Individual Minority Recalls

Table 18. Results of the Weather Data Sets with Individual Recalls of the Minority Classes

	Classifier Used	**HT**			**NB**			**OzaBag-HT**			**OzaBag-NB**			**OzaBoost-HT**			**OzaBoost-NB**		
Dataset	**Algorithm**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**
Cloudy-Fog-Snow	**INTER**	0.9064	0.8967		0.8429	0.8848		0.9178	0.8990		0.8523	0.8833		0.9338	0.8972		0.9291	0.8910	
	SCUT	0.9519	0.8907		0.9225	0.8723		0.9432	0.9102		0.9218	0.8725		0.9525	0.9162		0.9472	0.8967	
	SCUT++	0.9559	0.9149		0.8770	0.8735		0.9693	0.9025		0.8770	0.8738		0.9672	0.9222		0.9385	0.8972	
Clear-Fog-Ice	**INTER**	0.9171	0.1757		0.9151	0.2271		0.8851	0.1472		0.9158	0.2254		0.9158	0.2116		0.9589	0.2488	
	SCUT	0.9700	0.6821		0.9295	0.6299		0.9772	0.7415		0.9295	0.6253		0.9510	0.6407		0.9817	0.6039	
	SCUT++	0.9628	0.6600		0.9138	0.5240		0.9634	0.6838		0.9138	0.5224		0.9680	0.6604		0.9608	0.5337	
Cloudy-Snow-Ice	**INTER**	0.9046	0.6277		0.8882	0.7732		0.9039	0.6069		0.8882	0.7741		0.9019	0.6672		0.8907	0.7546	
	SCUT	0.9148	0.8310		0.8907	0.8397		0.9069	0.8475		0.8905	0.8388		0.9203	0.8493		0.9001	0.8497	
	SCUT++	0.9076	0.8315		0.8840	0.7689		0.8967	0.8315		0.8835	0.7689		0.9235	0.8067		0.8927	0.7763	

	Classifier Used	**HT**			**NB**			**OzaBag-HT**			**OzaBag-NB**			**OzaBoost-HT**			**OzaBoost-NB**		
Dataset	**Algorithm**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**
Cloudy-Clear-Fog	**INTER**	0.9642			0.9798			0.9837			0.9805			0.9863			0.9844		
	SCUT	0.9883			0.9772			0.9876			0.9779			0.9883			0.9824		
	SCUT++	0.9883			0.9759			0.9883			0.9759			0.9889			0.9824		
Cloudy-Clear-Snow	**INTER**	0.9058			0.9238			0.9090			0.9238			0.9211			0.9273		
	SCUT	0.9162			0.9233			0.9201			0.9233			0.9300			0.9277		
	SCUT++	0.9253			0.9211			0.9292			0.9211			0.9361			0.9280		
Cloudy-Clear-Ice	**INTER**	0.2227			0.2565			0.2086			0.2619			0.2194			0.3060		
	SCUT	0.6599			0.6474			0.7090			0.6515			0.7065			0.6632		
	SCUT++	0.6490			0.5712			0.7052			0.5670			0.6803			0.5733		
Cloudy-Clear-Snow-Ice	**INTER**	0.8761	0.2356		0.8884	0.2452		0.8938	0.2677		0.8887	0.2419		0.8983	0.2148		0.8889	0.3039	0.5964
	SCUT	0.9025	0.6120		0.8884	0.6328		0.9049	0.6286		0.8897	0.6324		0.9081	0.6382		0.8929	0.6399	0.7664
	SCUT++	0.9106	0.6582		0.8845	0.5795		0.9185	0.6611		0.8847	0.5803		0.9202	0.6445		0.8874	0.5941	0.7408
Cloudy-Snow-Ice-Fog	**INTER**	0.8972	0.6843	0.8242	0.8751	0.7554	0.7667	0.8935	0.6722	0.8647	0.8766	0.7546	0.7667	0.8860	0.6873	0.8601	0.8748	0.7441	0.8359
	SCUT	0.8811	0.8348	0.9333	0.8642	0.8235	0.8614	0.8853	0.8270	0.9451	0.8644	0.8209	0.8614	0.8892	0.8023	0.9275	0.8674	0.8365	0.9020
	SCUT++	0.8617	0.7827	0.9490	0.8696	0.7819	0.8366	0.8691	0.7749	0.9484	0.8711	0.7810	0.8366	0.8960	0.7415	0.9575	0.8709	0.7914	0.9059
Cloudy-Clear-Snow-Ice-Fog	**INTER**	0.8897	0.1986	0.8221	0.8769	0.2277	0.7629	0.8823	0.1873	0.8599	0.8744	0.2302	0.7648	0.8911	0.1765	0.8625	0.8783	0.2735	0.8000

	Classifier Used	HT			NB			OzaBag-HT			OzaBag-NB			OzaBoost-HT			OzaBoost-NB		
Dataset	**Algorithm**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**
	SCUT	0.8756	0.6511	0.9277	0.8594	0.6157	0.8691	0.8931	0.6515	0.9407	0.8616	0.6057	0.8678	0.8857	0.6049	0.9485	0.8580	0.6087	0.8756
	SCUT++	0.9115	0.7048	0.9322	0.8786	0.5749	0.8033	0.9078	0.7077	0.9394	0.8783	0.5733	0.8039	0.8956	0.6532	0.9433	0.8796	0.5953	0.8528

Table 19. Results of the NYC-Taxi-Fare Data Sets with Individual Recalls of the Minority Classes

NYC TRIP FARE	**Classifier Used**	**HT**			**NB**			**OzaBag-HT**			**OzaBag-NB**			**OzaBoost-HT**			**OzaBoost-NB**		
Dataset	**Algorithm**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**
NYC_123a	**INTER**	0.9950	0.9718		0.9617	0.7375		0.9954	0.9738		0.9582	0.7928		0.9978	0.9903		0.9801	0.9209	0.9505
	SCUT	0.9946	0.9836		0.9607	0.8327		0.9947	0.9843		0.9596	0.8342		0.9989	0.9997		0.9660	0.9473	0.9567
	SCUT++	0.9988	0.9838		0.9711	0.6874		0.9987	0.9816		0.9719	0.6757		0.9991	1.0000		0.9487	0.9433	0.9460
NYC_123b	**INTER**	0.9986			0.7465			0.9988			0.7529			0.9960			0.8630		
	SCUT	0.9966			0.9483			0.9980			0.9543			0.9996			0.9628		
	SCUT++	0.9964			0.9616			0.9976			0.9624			0.9978			0.9653		
NYC_1234a	**INTER**	0.8778	0.9347		0.6274	0.7166		0.8869	0.9307		0.6371	0.7065		0.8863	0.9638		0.7688	0.9156	0.8422
	SCUT	1.0000	0.9940		0.8398	0.8080		0.9998	0.9940		0.8503	0.7990		0.9994	0.9940		0.8319	0.9407	0.8863
	SCUT++	0.9984	0.9889		0.7559	0.8834		0.9986	0.9940		0.7605	0.8834		0.9998	0.9940		0.8133	0.9166	0.8649
NYC_1234b	**INTER**	0.9922	0.9284	0.9792	0.9623	0.6314	0.8252	0.9908	0.9409	0.9852	0.9580	0.6691	0.8400	0.9971	0.9784	0.9925	0.9483	0.8781	0.9576
	SCUT	0.9904	0.9546	0.9919	0.9536	0.7807	0.8602	0.9868	0.9596	0.9927	0.9291	0.8430	0.8505	0.9997	0.9994	0.9998	0.9690	0.8553	0.9713
	SCUT++	0.9966	0.9631	0.9984	0.9617	0.6045	0.8280	0.9969	0.9653	0.9984	0.9633	0.5793	0.8357	0.9998	1.0000	1.0000	0.9675	0.8763	0.9452
NYC_12345	**INTER**	0.9317	0.9125	0.9018	0.6922	0.7419	0.8158	0.8528	0.9327	0.8360	0.6004	0.7682	0.8016	0.9839	0.9806	0.9717	0.8427	0.8745	0.9291
	SCUT	0.9993	0.9929	0.9980	0.8449	0.7852	0.8978	0.9968	0.9909	0.9990	0.8604	0.7825	0.8937	0.9991	0.9935	0.9960	0.9001	0.8422	0.9312
	SCUT++	0.9996	0.9753	0.9889	0.8470	0.6831	0.7844	0.9995	0.9903	0.9879	0.8539	0.6888	0.7834	0.9993	0.9962	1.0000	0.8453	0.7589	0.9069

Table 20. Results of the Wave40 Data Sets with Individual Recalls of the Minority Classes

		Classifier Used	**HT**		**NB**		**OzaBag-HT**		**OzaBag-NB**		**OzaBoost-HT**		**OzaBoost-NB**	
Dataset	**No of Attributed**	**Algorithm**	**RECALL**											
			Min 1	**Min 2**	**Min 1**	**Min 2**	**Min 1**	**Min 2**	**Min 1**	**Min 2**	**Min 1**	**Min 2**	**Min 1**	**Min 2**
Wave40_75-15-10	40	INTER	0.5782	0.4672	0.9506	0.9271	0.6396	0.5566	0.9484	0.9273	0.7498	0.7131	0.9446	0.9214
		SCUT	0.8670	0.7810	0.9532	0.9267	0.8729	0.8036	0.9536	0.9267	0.8629	0.7920	0.9458	0.9183
		SCUT++	0.8590	0.8592	0.9477	0.9476	0.8855	0.8602	0.9472	0.9468	0.8595	0.8407	0.9402	0.9385
Wave40_94-5-1	40	INTER	0.5116	0.3057	0.9376	0.8239	0.4737	0.3036	0.9401	0.7976	0.5850	0.2935	0.9217	0.8077
		SCUT	0.8182	0.5628	0.9405	0.8198	0.8386	0.5587	0.9393	0.8178	0.7782	0.5526	0.9230	0.8320
		SCUT++	0.8447	0.4291	0.9527	0.7551	0.8907	0.4534	0.9523	0.7551	0.8125	0.5445	0.9372	0.8016
Wave40_80-5-15	40	INTER	0.4186	0.5975	0.9076	0.9496	0.3840	0.5981	0.9072	0.9473	0.5912	0.7541	0.9019	0.9363
		SCUT	0.7537	0.8796	0.9104	0.9517	0.7700	0.8936	0.9100	0.9520	0.7451	0.8562	0.8974	0.9411
		SCUT++	0.8111	0.8760	0.9438	0.9435	0.8302	0.8914	0.9426	0.9433	0.8070	0.8547	0.9316	0.9376
Wave40_54-40-6	40	INTER		0.3763		0.9084		0.5501		0.9046		0.6576		0.9040
		SCUT		0.7214		0.9026		0.7397		0.9019		0.7516		0.8958
		SCUT++		0.8219		0.9460		0.8622		0.9457		0.8222		0.9447
Wave40_45-45-10	40	INTER		0.5010		0.9211		0.5833		0.9193		0.7186		0.9116
		SCUT		0.7738		0.9207		0.7910		0.9214		0.7930		0.9169
		SCUT++		0.8570		0.9366		0.8767		0.9370		0.8401		0.9358

Table 21. Results of the Wave21 Data Sets with Individual Recalls of the Minority Classes

		Classifier Used	**HT**		**NB**		**OzaBag-HT**		**OzaBag-NB**		**OzaBoost-HT**		**OzaBoost-NB**	
Dataset	**No of Attributes**	**Algorithm**	**RECALL**											
			Min 1	**Min 2**	**Min 1**	**Min 2**	**Min 1**	**Min 2**	**Min 1**	**Min 2**	**Min 1**	**Min 2**	**Min 1**	**Min 2**
Wave21_75-15-10	21	INTER	0.6555	0.5642	0.9453	0.9334	0.6424	0.5874	0.9431	0.9334	0.7297	0.6936	0.9370	0.9299
		SCUT	0.8067	0.8582	0.9500	0.9432	0.8273	0.8665	0.9490	0.9436	0.8496	0.8553	0.9447	0.9401
		SCUT++	0.8485	0.8415	0.9434	0.9544	0.8765	0.8667	0.9437	0.9544	0.8609	0.8500	0.9412	0.9517
Wave21_94-5-1	21	INTER	0.4949	0.3785	0.9331	0.8320	0.4647	0.3178	0.9348	0.8016	0.6466	0.4332	0.9164	0.8340
		SCUT	0.8557	0.6700	0.9515	0.8968	0.8671	0.7227	0.9515	0.8988	0.8210	0.7004	0.9458	0.8988
		SCUT++	0.8525	0.6267	0.9633	0.9028	0.8826	0.6255	0.9633	0.9028	0.8447	0.6215	0.9580	0.9008
Wave21_80-5-15	21	INTER	0.4780	0.6300	0.9002	0.9592	0.4344	0.6095	0.9002	0.9592	0.5985	0.7595	0.8986	0.9545
		SCUT	0.8115	0.8566	0.9190	0.9640	0.8131	0.9036	0.9186	0.9633	0.7952	0.8773	0.9169	0.9580
		SCUT++	0.8286	0.8480	0.9157	0.9701	0.8322	0.8822	0.9153	0.9701	0.8017	0.8600	0.9190	0.9607
Wave21_54-40-6	21	INTER	0.3733		0.9158		0.3722		0.9125		0.6736		0.9091	
		SCUT	0.7889		0.9247		0.7866		0.9247		0.7832		0.9216	
		SCUT++	0.7886		0.9291		0.8422		0.9294		0.8293		0.9179	
Wave21_45-45-10	21	INTER	0.5831		0.9273		0.5896		0.9262		0.7555		0.9181	
		SCUT	0.8030		0.9368		0.8166		0.9358		0.8244		0.9301	
		SCUT++	0.8486		0.9423		0.8600		0.9425		0.8439		0.9405	

Table 22. Results of the LED_10% Data Sets with Individual Recalls of the Minority Classes

		Classifier Used	HT			NB			OzaBag-HT			OzaBag-NB			OzaBoost-HT			OzaBoost-NB								
Dataset	**Noise Level**	**Algorithm**	**RECALL**																							
			Min 1	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**						
LED_50-44-5-1	10%	INTER	0.7770	0.5202		0.8231	0.5364		0.7786	0.5202		0.8161	0.5344		0.7864	0.5061		0.8602	0.5263							
		SCUT	0.8806	0.4696		0.9132	0.5547		0.8936	0.5020		0.9132	0.5547		0.8740	0.5810		0.9054	0.5789							
		SCUT++	0.9160	0.6883		0.9083	0.6923		0.9197	0.7045		0.9079	0.6923		0.9119	0.6943		0.9013	0.7126							
LED_55-30-10-5	10%	INTER	0.8916	0.6710		0.9524	0.6731		0.8886	0.6376		0.9509	0.6682		0.8256	0.6572		0.8978	0.6918							
		SCUT	0.8841	0.7742		0.9100	0.8031		0.8925	0.7974		0.9084	0.8051		0.8841	0.7819		0.8961	0.8011							
		SCUT++	0.8628	0.8516		0.8661	0.8614		0.8716	0.8467		0.8665	0.8585		0.8674	0.8372		0.8708	0.8549							
LED_75-15-5-5	10%	INTER	0.9303	0.6840	0.7028	0.9343	0.6682	0.7040	0.9340	0.6804	0.6966	0.9343	0.6633	0.7011	0.9283	0.7313	0.7040	0.9341	0.7439	0.7221						
		SCUT	0.9330	0.8172	0.8481	0.9412	0.8404	0.8522	0.9389	0.8290	0.8481	0.9407	0.8420	0.8510	0.9418	0.8322	0.8124	0.9449	0.8502	0.8268						
		SCUT++	0.9431	0.8546	0.8543	0.9489	0.8734	0.8387	0.9446	0.8559	0.8576	0.9495	0.8742	0.8399	0.9375	0.8583	0.8502	0.9524	0.8640	0.8489						
LED_80-14-5-	10%	INTER	0.9372	0.72	0.54	0.94	0.72	0.54	0.93	0.73	0.53	0.94	0.72	0.54	0.94	0.79	0.51	0.94	0.79	0.53						

		Classifier Used	HT			NB			OzaBag-HT			OzaBag-NB			OzaBoost-HT			OzaBoost-NB								
Dataset	**Noise Level**	**Algorithm**	**RECALL**																							
			Min 1	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**						
1				70	66	63	54	66	83	07	04	61	29	86	32	73	62	51	08	85						
		SCUT	0.9358	0.9072	0.5162	0.9555	0.9281	0.5445	0.9451	0.9179	0.5405	0.9552	0.9285	0.5405	0.9498	0.8864	0.5810	0.9552	0.9191	0.5810						
		SCUT++	0.9482	0.9367	0.6781	0.9517	0.9526	0.6640	0.9531	0.9407	0.6984	0.9527	0.9526	0.6660	0.9545	0.9189	0.6479	0.9521	0.9518	0.7126						
LED_32-30-23-10-5	10%	INTER	0.7567	0.7535		0.7704	0.7832		0.7687	0.7779		0.7702	0.7819		0.7726	0.7705		0.7697	0.7819							
		SCUT	0.8330	0.8340		0.8247	0.8413		0.8289	0.8267		0.8236	0.8413		0.8226	0.8173		0.8230	0.8466							
		SCUT++	0.8443	0.8710		0.8440	0.8800		0.8518	0.8690		0.8424	0.8800		0.8355	0.8637		0.8457	0.8743							
LED_50-34-10-5-1	10%	INTER	0.9000	0.6474	0.5668	0.9531	0.6645	0.6721	0.8914	0.6327	0.5223	0.9533	0.6563	0.6559	0.8641	0.6535	0.4615	0.9194	0.6841	0.6356						
		SCUT	0.8998	0.7689	0.5789	0.9169	0.7982	0.6194	0.9037	0.7795	0.5870	0.9141	0.8015	0.6194	0.8755	0.7835	0.5445	0.9047	0.7880	0.6518						
		SCUT++	0.8665	0.8349	0.7449	0.9020	0.8259	0.8036	0.8743	0.8377	0.7591	0.9016	0.8259	0.8036	0.8741	0.8325	0.7571	0.8990	0.8255	0.8057						
		Classifier Used	**HT**				**NB**				**OzaBag-HT**				**OzaBag-NB**				**OzaBoost-HT**				OzaBoost-NB			
Dataset	**Noise Level**	**Algorithm**	**RECALL**																							
			Min 1	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**	**Min**

		Classifier Used	**HT**			**NB**			**OzaBag-HT**			**OzaBag-NB**			**OzaBoost-HT**			**OzaBoost-NB**								
Dataset	**Noise Level**	**Algorithm**	**RECALL**																							
			Min 1	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**						
				2	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**	**4**
LED_69-15-10-5-1	10%	INTER	0.9240	0.8940	0.6570	0.6041	0.9279	0.9244	0.6517	0.6857	0.9221	0.8762	0.6521	0.5673	0.9279	0.9197	0.6501	0.6633	0.9199	0.8465	0.6835	0.5367	0.9275	0.8870	0.6737	0.6694
		SCUT	0.9219	0.8925	0.7897	0.6000	0.9282	0.9187	0.8028	0.6082	0.9308	0.9044	0.7840	0.6082	0.9290	0.9209	0.8003	0.6102	0.9260	0.8829	0.7926	0.5980	0.9280	0.9123	0.8016	0.6388
		SCUT++	0.9067	0.8752	0.8207	0.7633	0.8883	0.8954	0.8293	0.7694	0.9094	0.8762	0.8277	0.7551	0.8886	0.8929	0.8318	0.7673	0.9098	0.8678	0.8163	0.7612	0.8908	0.8927	0.8203	0.7755

Table 23. Results of the LED_20% Data Sets with Individual Recalls of the Minority Classes

		Classifier Used	HT			NB			OzaBag-HT			OzaBag-NB			OzaBoost-HT			OzaBoost-NB								
Dataset	**Noise Level**	**Algorithm**	**RECALL**																							
			Min 1	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**						
LED_50-44-5-1	20%	INTER	0.4338	0.1377		0.4378	0.1579		0.4333	0.1397		0.4370	0.1397		0.4505	0.1417		0.4594	0.1721							
		SCUT	0.6731	0.3158		0.6930	0.3988		0.7000	0.3340		0.6898	0.4008		0.6816	0.2834		0.6926	0.3988							
		SCUT++	0.7709	0.3887		0.7566	0.4130		0.7636	0.4211		0.7566	0.4109		0.7501	0.4413		0.7587	0.4352							
LED_55-30-10-5	20%	INTER	0.5593	0.2823		0.5609	0.2766		0.5534	0.2819		0.5597	0.2742		0.5571	0.3088		0.5944	0.2929							
		SCUT	0.7055	0.5675		0.7383	0.5627		0.7208	0.5553		0.7375	0.5606		0.7093	0.5557		0.7320	0.5622							
		SCUT++	0.7369	0.6387		0.7434	0.6448		0.7440	0.6428		0.7434	0.6456		0.7216	0.6432		0.7375	0.6554							
LED_75-15-5-5	20%	INTER	0.7493	0.4176	0.3946	0.7781	0.4404	0.4194	0.7587	0.4269	0.4056	0.7784	0.4424	0.4203	0.7521	0.4282	0.4015	0.7754	0.4298	0.4471						
		SCUT	0.8177	0.6135	0.6050	0.8175	0.6527	0.6017	0.8118	0.6494	0.6046	0.8164	0.6531	0.6021	0.8158	0.6429	0.5854	0.8225	0.6490	0.5968						
		SCUT+	0.8021	0.70	0.65	0.81	0.72	0.65	0.80	0.70	0.65	0.81	0.73	0.65	0.80	0.69	0.65	0.81	0.73	0.65						

		Classifier Used	HT			NB			OzaBag-HT			OzaBag-NB			OzaBoost-HT			OzaBoost-NB								
Dataset	Noise Level	Algorithm	RECALL																							
			Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3						
		+		08	01	33	98	05	63	78	13	57	14	22	79	80	01	91	02	46						
LED_80-14-5-1	20%	INTER	0.7725	0.4521	0.0749	0.8024	0.4782	0.0729	0.7821	0.4537	0.0506	0.8024	0.4778	0.0466	0.7808	0.4562	0.0810	0.7964	0.4782	0.0789						
		SCUT	0.8261	0.7044	0.3138	0.8344	0.7012	0.4028	0.8360	0.7097	0.3522	0.8348	0.7016	0.4028	0.8312	0.7012	0.3117	0.8372	0.7053	0.3947						
		SCUT++	0.8348	0.7672	0.4413	0.8337	0.7493	0.4251	0.8311	0.7680	0.4433	0.8353	0.7489	0.4251	0.8313	0.7591	0.4130	0.8395	0.7513	0.4372						
LED_32-30-23-10-5	20%	INTER	0.4701	0.5106		0.4664	0.5386		0.4511	0.4976		0.4646	0.5354		0.4470	0.4813		0.4679	0.5338							
		SCUT	0.6089	0.6387		0.6056	0.6619		0.6134	0.6395		0.6089	0.6627		0.6024	0.6343		0.6054	0.6623							
		SCUT++	0.6389	0.7063		0.6597	0.7217		0.6471	0.7209		0.6595	0.7225		0.6424	0.7038		0.6632	0.7246							
	20%																									
LED_50-34-10-5-1	20%	INTER	0.5251	0.2764	0.0263	0.5359	0.2699	0.0000	0.5208	0.2699	0.0000	0.5318	0.2687	0.0000	0.5610	0.2850	0.0162	0.5908	0.2784	0.0101						
		SCUT	0.7394	0.5495	0.3765	0.7571	0.5556	0.4676	0.7347	0.5556	0.3887	0.7551	0.5548	0.4656	0.7200	0.5463	0.4190	0.7488	0.5618	0.4777						

		Classifier Used	HT			NB			OzaBag-HT			OzaBag-NB			OzaBoost-HT			OzaBoost-NB								
Dataset	Noise Level	Algorithm	RECALL																							
			Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3						
		SCUT++	0.7139	0.6229	0.5182	0.7059	0.6543	0.5121	0.7086	0.6376	0.5202	0.7082	0.6535	0.5101	0.7204	0.6168	0.5425	0.7006	0.6612	0.5142						

		Classifier Used	HT				NB				OzaBag-HT				OzaBag-NB				OzaBoost-HT				OzaBoost-NB			
Dataset	Noise Level	Algorithm	RECALL																							
			Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4
LED_69-15-10-5-1	20%	INTER	0.7493	0.5255	0.3698	0.1721	0.7646	0.5220	0.3787	0.2247	0.7505	0.5182	0.3640	0.1235	0.7664	0.5237	0.3795	0.1842	0.7509	0.5441	0.3567	0.1538	0.7645	0.5443	0.3893	0.1984
		SCUT	0.7767	0.7208	0.5479	0.4130	0.7729	0.7569	0.5834	0.4332	0.7898	0.7314	0.5581	0.4008	0.7707	0.7561	0.5842	0.4332	0.7814	0.7153	0.5479	0.3927	0.7716	0.7518	0.5785	0.4352
		SCUT++	0.7638	0.7171	0.6319	0.5040	0.7747	0.7567	0.6205	0.5101	0.7716	0.7298	0.6270	0.5243	0.7741	0.7563	0.6192	0.5121	0.7671	0.7263	0.6307	0.5283	0.7790	0.7516	0.6249	0.5223

Table 24. Results of the LED_30% Data Sets with Individual Recalls of the Minority Classes

		Classifier Used	**HT**			**NB**			**OzaBag-HT**			**OzaBag-NB**			**OzaBoost-HT**			**OzaBoost-NB**								
Dataset	**Noise Level**	**Algorithm**	**RECALL**																							
			Min 1	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**	**Min 1**	**Min 2**	**Min 3**						
LED_50-44-5-1	30%	INTER	0.1279	0.0062		0.1684	0.0000		0.1037	0.0000		0.1639	0.0000		0.1049	0.0000		0.1955	0.0000							
		SCUT	0.4061	0.2099		0.4332	0.2634		0.4180	0.1893		0.4303	0.2593		0.4234	0.1790		0.4365	0.2634							
		SCUT++	0.5508	0.1276		0.5791	0.2160		0.5697	0.1173		0.5795	0.2140		0.5574	0.1379		0.5770	0.2222							
LED_55-30-10-5	30%	INTER	0.2539	0.0411		0.2690	0.0435		0.2582	0.0341		0.2686	0.0411		0.2600	0.0493		0.2727	0.0489							
		SCUT	0.4940	0.3350		0.5044	0.3801		0.4954	0.3448		0.5021	0.3826		0.4948	0.3428		0.5030	0.3756							
		SCUT++	0.5539	0.4031		0.5633	0.4253		0.5544	0.4212		0.5650	0.4253		0.5495	0.3875		0.5611	0.4245							
LED_75-15-5-5	30%	INTER	0.4069	0.0380	0.0641	0.4614	0.0519	0.0707	0.3704	0.0180	0.0641	0.4585	0.0421	0.0772	0.4030	0.0637	0.0801	0.4744	0.0931	0.0927						
		SCUT	0.6271	0.4220	0.3542	0.6360	0.4420	0.4052	0.6336	0.4342	0.3676	0.6342	0.4440	0.4040	0.6226	0.4171	0.3766	0.6362	0.4481	0.4036						
		SCUT+	0.6206	0.47	0.44	0.63	0.48	0.46	0.62	0.47	0.46	0.63	0.48	0.46	0.61	0.48	0.44	0.63	0.48	0.46						

		Classifier Used	HT			NB			OzaBag-HT			OzaBag-NB			OzaBoost-HT			OzaBoost-NB								
Dataset	Noise Level	Algorithm	RECALL																							
			Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3						
		+		55	49	69	65	90	21	63	24	55	49	94	23	16	89	65	73	73						
LED_80-14-5-1	30%	INTER	0.3137	0.0623	0.0000	0.4057	0.0787	0.0000	0.3111	0.0393	0.0000	0.4095	0.0738	0.0000	0.3680	0.1004	0.0021	0.4405	0.1459	0.0000						
		SCUT	0.6462	0.4471	0.2078	0.6382	0.4607	0.2922	0.6708	0.4500	0.1379	0.6377	0.4574	0.2881	0.6585	0.4426	0.1440	0.6371	0.4480	0.2922						
		SCUT++	0.6567	0.5545	0.1379	0.6341	0.5725	0.2243	0.6580	0.5820	0.1193	0.6333	0.5725	0.2181	0.6607	0.5697	0.0905	0.6398	0.5684	0.2263						
LED_32-30-23-10-5	30%	INTER	0.1478	0.1025		0.1357	0.1021		0.1364	0.0789		0.1384	0.1005		0.1453	0.0895		0.1345	0.1062							
		SCUT	0.3795	0.3946		0.3880	0.4272		0.3901	0.3877		0.3897	0.4255		0.3768	0.3987		0.3801	0.4260							
		SCUT++	0.4011	0.5081		0.4446	0.5118		0.4040	0.5045		0.4440	0.5130		0.1751	0.2030		0.1051	0.1265							
LED_50-34-10-5-1	30%	INTER	0.2627	0.0594	0.0000	0.2660	0.0512	0.0000	0.2531	0.0398	0.0000	0.2650	0.0488	0.0000	0.2511	0.0643	0.0021	0.2641	0.0541	0.0000						
		SCUT	0.4925	0.3270	0.2490	0.5013	0.3676	0.3025	0.5026	0.3352	0.2202	0.5040	0.3742	0.3025	0.4911	0.3340	0.2119	0.5028	0.3676	0.3128						

		Classifier Used	HT			NB			OzaBag-HT			OzaBag-NB			OzaBoost-HT			OzaBoost-NB								
Dataset	Noise Level	Algorithm	RECALL																							
			Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3	Min 1	Min 2	Min 3						
		SCUT++	0.4807	0.4316	0.3004	0.4787	0.4516	0.3313	0.4905	0.4430	0.2716	0.4801	0.4496	0.3292	0.4864	0.4291	0.2860	0.4646	0.4320	0.3025						

		Classifier Used	HT				NB				OzaBag-HT				OzaBag-NB				OzaBoost-HT				OzaBoost-NB			
Dataset	Noise Level	Algorithm	RECALL																							
			Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4	Min 1	Min 2	Min 3	Min 4
LED_69-15-10-5-1	30%	INTER	0.4434	0.2733	0.0578	0.0041	0.4822	0.3086	0.0434	0.0021	0.4060	0.2519	0.0418	0.0021	0.4814	0.3074	0.0373	0.0021	0.3972	0.2400	0.0529	0.0021	0.4871	0.3046	0.0541	0.0041
		SCUT	0.5747	0.5036	0.3561	0.1811	0.5456	0.5183	0.3770	0.3148	0.5843	0.5166	0.3623	0.1893	0.5423	0.5189	0.3750	0.3107	0.5649	0.5052	0.3611	0.2407	0.5426	0.5252	0.3742	0.3292
		SCUT++	0.5313	0.4948	0.4184	0.2798	0.5344	0.4893	0.4377	0.3313	0.5245	0.4930	0.4156	0.2901	0.5343	0.4909	0.4328	0.3292	0.5231	0.4854	0.4082	0.2819	0.5298	0.4958	0.4275	0.3374

Appendix B

Pairwise comparisons using Conover's test

Category 1: Classifiers					
Data Set: Wave21					
	HT	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**
NB	< 2e-16				
OzaBag.HT	0.0095	< 2e-16			
OzaBag.NB	< 2e-16	0.2385	< 2e-16		
OzaBoost.HT	0.0417	< 2e-16	0.5302	< 2e-16	
OzaBoost.NB	< 2e-16	1.30E-09	1.40E-11	7.60E-07	1.10E-12
Data Set: Wave40					
	HT	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**
NB	< 2e-16				
OzaBag.HT	0.00042	< 2e-16			
OzaBag.NB	< 2e-16	0.01163	< 2e-16		
OzaBoost.HT	0.00262	< 2e-16	0.52925	< 2e-16	
OzaBoost.NB	< 2e-16	6.30E-10	1.30E-11	5.40E-05	1.00E-12
Data Set: LED_10%					
	HT	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**
NB	< 2e-16	-	-	-	-
OzaBag.HT	0.013	1.60E-11	-	-	-
OzaBag.NB	1.20E-13	0.127	2.60E-07	-	-
OzaBoost.HT	0.306	< 2e-16	1.00E-04	< 2e-16	-
OzaBoost.NB	< 2e-16	0.306	1.20E-13	0.013	< 2e-16
Data Set: LED_20%					
	HT	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**
NB	< 2e-16	-	-	-	-
OzaBag.HT	0.0713	2.90E-11	-	-	-
OzaBag.NB	1.60E-12	0.2214	1.30E-07	-	-
OzaBoost.HT	0.2228	6.50E-13	0.4238	3.40E-09	-
OzaBoost.NB	< 2e-16	0.0033	< 2e-16	3.80E-06	< 2e-16

Category 1: Classifiers					
Data Set: LED_30%					
	HT	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**
NB	< 2e-16				
OzaBag.HT	0.101	< 2e-16			
OzaBag.NB	3.80E-12	0.067	< 2e-16		
OzaBoost.HT	0.067	< 2e-16	1	< 2e-16	
OzaBoost.NB	< 2e-16	1	< 2e-16	0.067	< 2e-16
Data Set: Weather					
	HT	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**
NB	9.40E-07				
OzaBag.HT	2.60E-05	< 2e-16			
OzaBag.NB	9.40E-07	1	< 2e-16		
OzaBoost.HT	3.40E-09	< 2e-16	0.1462	< 2e-16	
OzaBoost.NB	0.004	2.00E-14	0.3545	2.00E-14	0.0043
Data Set: NYC-Taxi Fare					
	HT	**NB**	**OzaBag-HT**	**OzaBag-NB**	**OzaBoost-HT**
NB	< 2e-16				
OzaBag.HT	0.10278	< 2e-16			
OzaBag.NB	< 2e-16	0.04717	< 2e-16		
OzaBoost.HT	3.80E-07	< 2e-16	0.00016	< 2e-16	
OzaBoost.NB	9.50E-09	2.90E-12	1.00E-11	3.20E-08	< 2e-16

Category 2: The Three Algorithms		
Data Set: Wave21		
	INTER	SCUT
SCUT_DS	< 2e-16	
SCUT-DS++	< 2e-16	7.80E-14
Data Set: Wave40		
	INTER	SCUT
SCUT_DS	1.80E-15	
SCUT-DS++	< 2e-16	< 2e-16
Data Set: LED_10%		
	INTER	SCUT
SCUT_DS	< 2e-16	
SCUT-DS++	< 2e-16	< 2e-16
Data Set: LED_20%		
	INTER	SCUT
SCUT_DS	< 2e-16	
SCUT-DS++	< 2e-16	< 2e-16
Data Set: LED_30%		
	INTER	SCUT
SCUT_DS	< 2e-16	
SCUT-DS++	< 2e-16	< 2e-16
Data Set: WEATHER		
	INTER	SCUT
SCUT_DS	< 2e-16	
SCUT-DS++	< 2e-16	< 2e-16
Data Set: NYC-TAXI-FARE		
	INTER	SCUT
SCUT_DS	< 2e-16	
SCUT-DS++	3.50E-15	1.60E-08