M.Sc. Engg. Thesis

An Algorithmic Framework Based on the Binarization Approach for Supervised and Semi-supervised Multiclass Problems

by

Ayon Sen

Submitted to

Department of Computer Science and Engineering

in partial fulfilment of the requirements for the degree of Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology (BUET)

Dhaka 1000

May 2014

Dedicated to my loving parents

Author's Contact

Ayon Sen

Lecturer

Department of Computer Science & Engineering

Bangladesh University of Engineering & Technology (BUET).

Email: ayonsen@cse.buet.ac.bd

The thesis titled "An Algorithmic Framework Based on the Binarization Approach for Supervised and Semi-supervised Multiclass Problems", submitted by Ayon Sen, Roll No. **0412052063P**, Session April 2012, to the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering and approved as to its style and contents. Examination held on May 20, 2014.

Board of Examiners

Candidate's Declaration

This is hereby declared that the work titled "An Algorithmic Framework Based on the Binarization Approach for Supervised and Semi-supervised Multiclass Problems" is the outcome of research carried out by me under the supervision of Dr. Md. Monirul Islam, in the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka 1000. It is also declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Ayon Sen

Candidate

Acknowledgment

I express my heart-felt gratitude to my supervisor, Dr. Md. Monirul Islam for his constant supervision of this work. He helped me a lot in every aspect of this work and guided me with proper directions whenever I sought one. His patient hearing of my ideas, critical analysis of my observations and detecting flaws (and amending thereby) in my thinking and writing have made this thesis a success.

I would also want to thank the members of my thesis committee for their valuable suggestions. I thank Dr. Mohammad Mahfuzul Islam, Dr. Yusuf Sarwar Uddin, Dr. Md Shohrab Hossain and specially the external member Dr. Mohammad Nurul Huda.

In this regard, I remain ever grateful to my beloved parents, who always exists as sources of inspiration behind every success of mine I have ever made.

Abstract

Using a set of binary classifiers to solve the multiclass classification problem has been a popular approach over the years. This technique is known as binarization. The decision boundary that these binary classifiers (also called base classifiers) have to learn is much simpler than the decision boundary of a multiclass classifier. But binarization gives rise to a new problem called the class imbalance problem. Class imbalance problem occurs when the data set used for training has relatively less data items for one class than for another class. This problem becomes more severe if the original data set itself was imbalanced. Furthermore, binarization has only been implemented in the domain of supervised classification.

In this thesis, we propose a framework called Binarization with Boosting and Oversampling (BBO). Our framework can handle the class imbalance problem arising from binarization. As the name of the framework suggests, this is achieved through a combination of boosting and oversampling. BBO framework can be used with any supervised classification algorithm. Moreover, unlike any other binarization approaches used earlier, we apply our framework with semi-supervised classification as well. BBO framework has been rigorously tested with a number of benchmark data sets from UCI machine learning repository. The experimental results show that using the BBO framework achieves a higher accuracy than the traditional binarization approach.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Introduction

Multiclass classification is a problem of building a system that accurately maps an input feature space to an output space of more than two classes. It has a wide range of applications such as handwritten digit recognition, object classification, speech tagging and recognition, bioinformatics, text categorization and information retrieval. While the binary (two class) classification problem is well understood, multiclass classification is relatively less investigated. Many classification systems are developed for binary classification problems and theoretical studies of learning have focused almost entirely on learning binary functions including the well-known support vector machines (SVMs), artificial neural network algorithms such as the perceptron and the backpropagation (BP) algorithm. For most of these algorithms, the extension from the binary to the multiclass classification problem is non-trivial [1, 2], and often leads to unexpected complexity or weaker performances.

Multiclass classification problems can be addressed in different ways [1]. One of the most popular techniques consists in dividing a given multiclass classification problem into several two-class classification problems, building a different binary model, called base classifier, for each binary problem. This technique is known as binarization, which not only helps in producing simple classifiers, but also helps in reducing time to train classifiers. This is because there is a chance to train the binary classifiers in parallel. Over the years, many algorithms have been devised based on binarization [3]. There are mainly two popular approaches regarding binarization: one-vs-one (OVO) and one-vs-all (OVA).

The OVO approach divides a given problem into as many binary problems as possible. If there are m classes then the number of base classifiers is ${}^{m}\mathrm{C}_2$. This approach trains one classifier to discriminate between a pair of classes. OVA consists in creating a base classifier to learn each class, where the class is distinguished from all other classes. OVA has the benefit of using less resources than OVO. If there are m classes, then the number of base classifiers for OVA will also be m, which is significantly less than ${}^{m}\mathrm{C}_2$ for a large m. In both cases, the output of the base classifiers need to be combined to predict the output class. This is known as the aggregation strategy. OVA has mainly two aggregation strategies: the max confidence strategy (MAX) and the dynamically ordered OVA (DOO) strategy [4]. OVO, on the other hand, has multiple aggregation strategies with no clear winner. Even though many algorithms have been developed over the years based on binarization, most of them have been unable to address the class imbalance problem [5]. It occurs when the number of data items in the training set is significantly higher for one class than the other. The lack of data items for a class in the training set arising from binarization can adversely effect the performance of classifiers. Moreover, to the best of our knowledge, binarization has only been implemented in the domain of supervised classification.

1.2 Related Work

The use of binary classifiers to solve multiclass problems has been a popular technique over the years. In this section, we briefly discuss different OVO and OVA approaches. These approaches have been implemented to solve various problems using a multitude of binary classifiers. Our proposed approach described in Chapter 3 works in the domain of both supervised and semi-supervised classification. Since our approach works with semi-supervised classification, we also discuss here several popular semi-supervised classification algorithms.

OVO has been used by researchers to solve different multiclass problems. The main difference between the different approaches is the aggregation strategy, which combines the output of the base classifiers to get a class prediction. The different aggregation strategies are VOTE [6], PC [7], DDAG [8] and others [9, 10, 11, 12]. OVA, on the other hand, has only two aggregation strategies, MAX and DOO [4]. OVA techniques have also been used on multiple problem domains over the years. Some of these include fingerprint classification [4], handwriting detection [13], video categorization [14] etc. Binarization techniques cause the training set to become imbalance. Each base classifier is provided with a training set with only two classes. In case of of OVA, one class is learned by the base classifier called the target class. All other data items are labeled as the default class. So, data items for the default class becomes significantly larger than the target class causing the training set to become imbalance. For OVO, if the original training set is imbalanced then the training set of all base classifier also become imbalance.

Now, we look at semi-supervised classification. As labeled data is expensive to come by, the use of unlabeled data to improve the effectiveness of a classifier has become popular in recent years. Hence, a lot of theoretical and empirical studies have been done in the field of semi-supervised classification. These algorithms can be broadly categorized into co-training, manifold assumption, cluster assumption and ensemble.

In the co-training methods, independent classifiers are trained and they learn from each other. The authors of [15] divide the feature of the training set into two sets (views) assuming each set is independent of the other and individually is enough for the classification task. One classifier is assigned to each view. Each classifier is trained with the labeled data of its view and then labels the unlabeled data of its view. A confidence measure is associated with these labeling. The unlabeled data with the highest labeling confidence is used as training data for the other classifier. In this way the two classifiers learn from each other. The main problem associated with this technique is how the features can be divided. To counter this, the authors of [16] propose using three classifiers instead of two. Here if two classifiers agree on the label of an unlabeled data then that data is used in the training set of the other classifier. But this method depends on the effectiveness of the data splitting technique. The authors of [17] propose an algorithm based on [16] where the label of an unlabeled data can be changed.

In semi-supervised classification, it is assumed that the true structure of the data lies in a low-dimensional manifold embedded in the high-dimensional data space. This is known as the *manifold assumption.* Algorithms based on this assumption typically build graphs to represent all the instances [18, 19]. In order to predict labels in the graph these methods usually assume label smoothness among these instances.

The cluster assumption states that classes are often separated by a low-density region. TSVM [20], SemiBSVM [21] and Learning with Local and Global Consistency (LLGC) [22] algorithms are based on this assumption. LLGC uses the regularization framework. During each step the information of the unlabeled data set is gathered from its neighbors based on a parameter, α. Zhai *et al.* (2012) proposed a multiview version of LLGC in [23]. LLGC has been known to perform well in different domains like image and text classification. In [24], the authors proposed a technique based on regularization. Their algorithm takes the partition given by an algorithm as a regularization term in the loss function of an semi-supervised classifier.

The authors of [25, 26, 27] proposed algorithms based on ensemble. The authors of [26] also address the issue of imbalanced data set. They work with multiclass classifiers and use boosting to address the issue. Moreover they mention using binarization for solving multiclass problems but do not present any existing work.

In this section, we gave overviews of research works related to binarization and semi-supervised classification. Most of the research work discussed here related to OVO and OVA fail to address the issue of class imbalance problem [5]. Moreover, binarization has not been used with semi-supervised classification to the best of our knowledge. Most of the research work related to semi-supervised classification also deal only with binary classification.

1.3 Objective of the Thesis

There are a multitude of ways to handle the multiclass classification problem. Using binarization is one of the most popular of approaches. Many research works exist in literature that deal with using binarization to solve the problem. Similarly there are many works that address the issue of class imbalance problem. But unfortunately there is no research work that addresses class imbalance problem arising from binarization.

The objective of this thesis is to tackle the problems arising from binarization. We propose a framework based on OVA. The salient feature of our framework is that it can handle the class imbalance problem occurring due to binarization. Unlike previous research works that deal with class imbalance problem, our framework takes a different approach. Our framework uses a combination of boosting and oversampling techniques to address the class imbalance problem. To predict the output class we use the DOO aggregation strategy. Furthermore, it can be used with any supervised classification algorithm. It can also be used with any semi-supervised classification algorithm, which has not been used with binarization before.

We also perform a rigorous empirical study on the effect of our framework on various base classifiers. For both supervised and semi-supervised classification we use multiple data sets and multiple algorithms to test the BBO framework. All that experimental study is also presented with thorough analysis.

1.4 Thesis Organization

The remainder of this thesis is organized as follows.

In chapter 2, we briefly discuss the concepts that are necessary to understand the idea of the thesis. We discuss about various classification tasks and how they can be solved. We discuss the two main approaches of binarization that are used to solve the multiclass classification problem in detail. We also describe how the result of the different base classifiers are combined for these two techniques. Since we are dealing with multiclass classification, we look at some popular supervised multiclass classifiers also. We discuss their main working principals

Chapter 3 presents the major contribution of this thesis. We introduce our framework that can be used to solve the class imbalance problem arising from binarization. We present the different components of the framework and discuss why each component is necessary and important in handling the problem.

In chapter 4, the experimental analysis regarding the performance of the proposed framework is presented. We first present a description of the various data sets that are used for our experiments. We then discuss the different performance measures and why we use them. It is followed by our experimental plan. We provide an extensive simulation and result analysis against multiple baseline algorithms. Moreover, for supervised classification we also compare our results with various state-of-the-art binarization algorithms. In order to establish the fact that our framework improves the performance of the base algorithm, we also perform statistical analysis here.

Finally, in chapter 5 we briefly conclude the thesis. We also provide some future aspects of this research. We try to provide some directions regarding how the presented framework can be modified and extended further.

Chapter 2

Background

2.1 Introduction

This chapter begins by introducing the classification task. We discuss the various categories of the classification task. The main two classification tasks are called binary classification and multiclass classification. Multiclass classification can be solved in a number of different approaches. One such approach is called binarization. We also provide a thorough analysis of binarization in this chapter. We then look at boosting and oversampling techniques as these are used in our proposed approach. Afterwards, we give an in-depth discussion of various popular multiclass classifiers. We look at their salient features, strengths and weaknesses. We end this chapter by giving a summary of the topics discussed.

2.2 Classification

Classification is the machine learning task of identifying to which of a set of classes a new data item belongs to, based on training data. An algorithm that implements classification is known as a classifier. Let $f(x)$ be the target function that is required to be learnt and y be the training data. Under the statistical learning framework [28], given a loss function $e(y, f(x))$ representing an error measure for classification, and a finite set of training data, T, drawn from an unknown joint probability density function $p(x, y)$, the goal of a classifier is to find the optimal $f_0(x)$ that minimizes the expected risk, $R[f]$

$$R[f] = \int_{X \times Y} e(y, f(x))p(x, y)dydx = E[e(y, f(x))] \tag{2.1}$$

over the class of functions $f : X \rightarrow Y$, provided by learning algorithm, that map examples from input space X to output space Y. Based on the number of labels or classes, the classification task can be divided into two groups:

- **Binary Classification:** When the number of classes for a given classification task is two, it is called binary classification. The task of the classifier is to differentiate between these two classes. Figure 2.1 shows an example of a binary classification task. There are only two classes of data items available here.

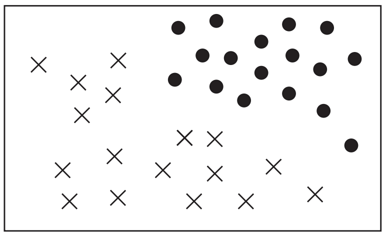


Figure 2.1: Data Items for Binary Classification Problem

- **Multiclass Classification:** When the number of classes for a given classification task is more than two, it is called multiclass classification. In general the decision boundary that a multiclass classifier has to learn is significantly more complex than that of a binary classifier. In Figure 2.2, we see an example of a multiclass classification task. Here the number of classes is more than two.

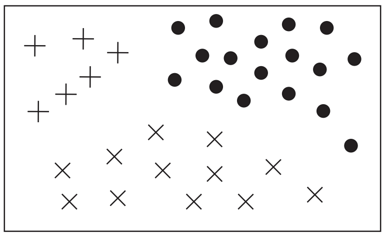


Figure 2.2: Data Items for Multiclass Classification Problem

The classification task can also be categorized on the basis of the availability of labeled data during the training phase of a classifier. Labeled data is a data item whose class label is known beforehand. These categories are also applicable on both binary and multiclass classification. They are described as follows:

- **Supervised Classification:** Supervised classification learns the classification function only from labeled data. In supervised classification, each data item in the training set is a pair containing attributes and class label. A supervised classifier analyzes the training data and produces an inferred function.

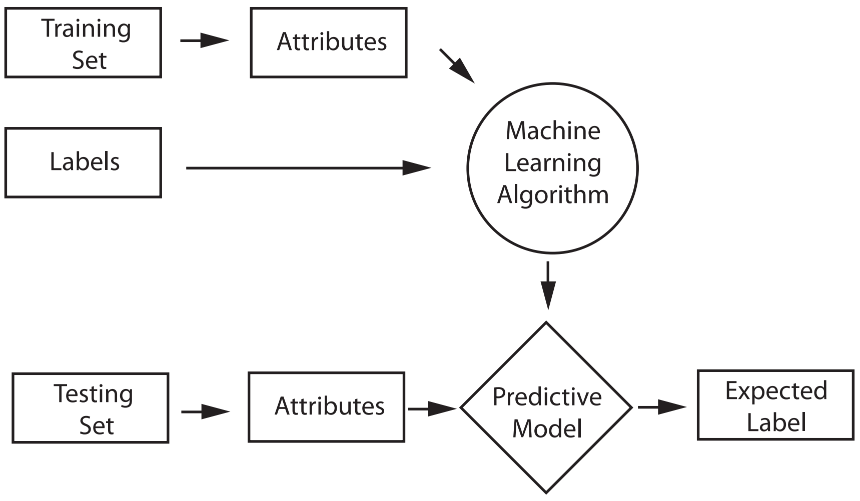


Figure 2.3: Flowchart for Supervised Learning

  The inferred function can then be used to predict the label of unseen data. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances. This requires the learning algorithm to generalize from the training data to unseen situations in a reasonable way.

  In order to solve a given classification problem, a supervised classifier has to perform the following steps:

  – Determine the feature vector or attributes of the data item. Each supervised classifier works with a fixed set of features from which the target function is approximated. These are also called attributes.

- Gather a training set. The training set needs to be representative of the real-world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, either from human experts or from measurements.
- Determine the structure of the learned function and corresponding learning algorithm from the attributes of the training set.
- Evaluate the accuracy of the learned function. After parameter adjustment and learning, the performance of the resulting function should be measured on a testing set that is separate from the training set.

Figure 2.3 shows an example of supervised learning. The attributes are first extracted from the training data and passed to the classifier with their labels. The classifier then tries to learn how to classify unseen data. When new data items arrive it tries to predict the class labels.

- **Unsupervised Classification:** As the name suggests, unsupervised classification learns the hidden structure in unlabeled data. Since the training data is unlabeled, there is no measure to evaluate the performance of the learned function. This criterion distinguishes unsupervised classification from supervised classification.

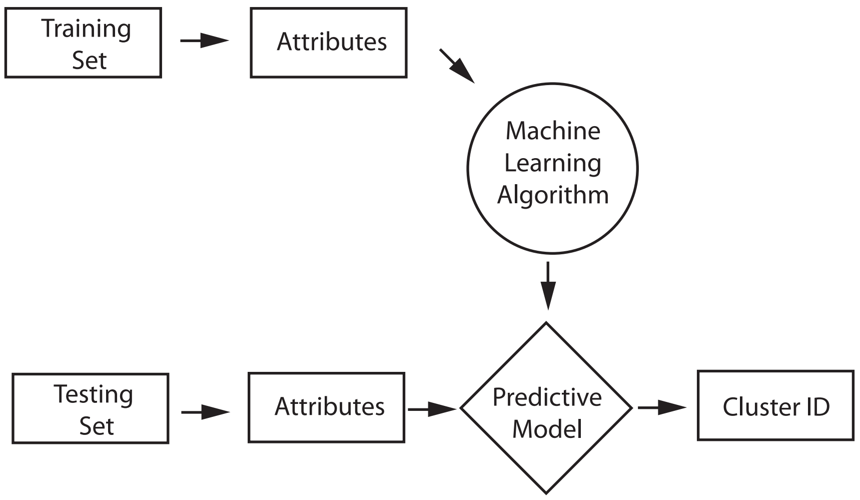


Figure 2.4: Flowchart for Unsupervised Learning

Unsupervised classification is also known as clustering. Given a set of labeled data, the task of the classifier is to group the data items in a way such that the data items in the same group (cluster) are more similar to one another than those in other groups. There are various different cluster algorithms available with different views of what constitutes a cluster. Popular notions of clusters include groups with small distances among the cluster members, dense areas of the data space, intervals or particular statistical distributions. Clustering can therefore be formulated as a multi-objective optimization problem.

Figure 2.4 shows an example of unsupervised learning. The algorithm takes a set of unlabeled data as input during the training phase. It is the task of the classifier to assign each training data item into a cluster. The maximum number of clusters to be assigned can be fixed beforehand, or can be left upto the classifier to discover. Each cluster is then assigned a Cluster ID. When a new data item arrives, the algorithm assigns the new data item to one of these clusters.

- **Semi-supervised Classification:** In the domain of classification acquiring unlabeled data is quite cheap. On the other hand, acquiring labeled data can be costly, time consuming and may require human intervention. Unfortunately the performance of unsupervised classification cannot match to that of supervised classification. But the performance of a supervised classifier may be improved by using unlabeled data. This type of classification, where both labeled and unlabeled data is used to train a classifier is called semi-supervised classification. Typically semi-supervised classifiers learn from a small amount of labeled data and a large amount of unlabeled data.

  As in the supervised classification framework, we are given a set of l independently identically distributed examples $x_1, \ldots, x_l \in X$ with corresponding labels $y_1, \ldots, y_l \in Y$. Additionally, we are given u unlabeled examples $x_{l+1}, \ldots, x_{l+u} \in X$. A semi-supervised classifier attempts to make use of this combined information to surpass the classification performance that could be obtained either by discarding the unlabeled data and doing supervised learning or by discarding the labels and doing unsupervised learning.

  There are two types of semi-supervised classification:

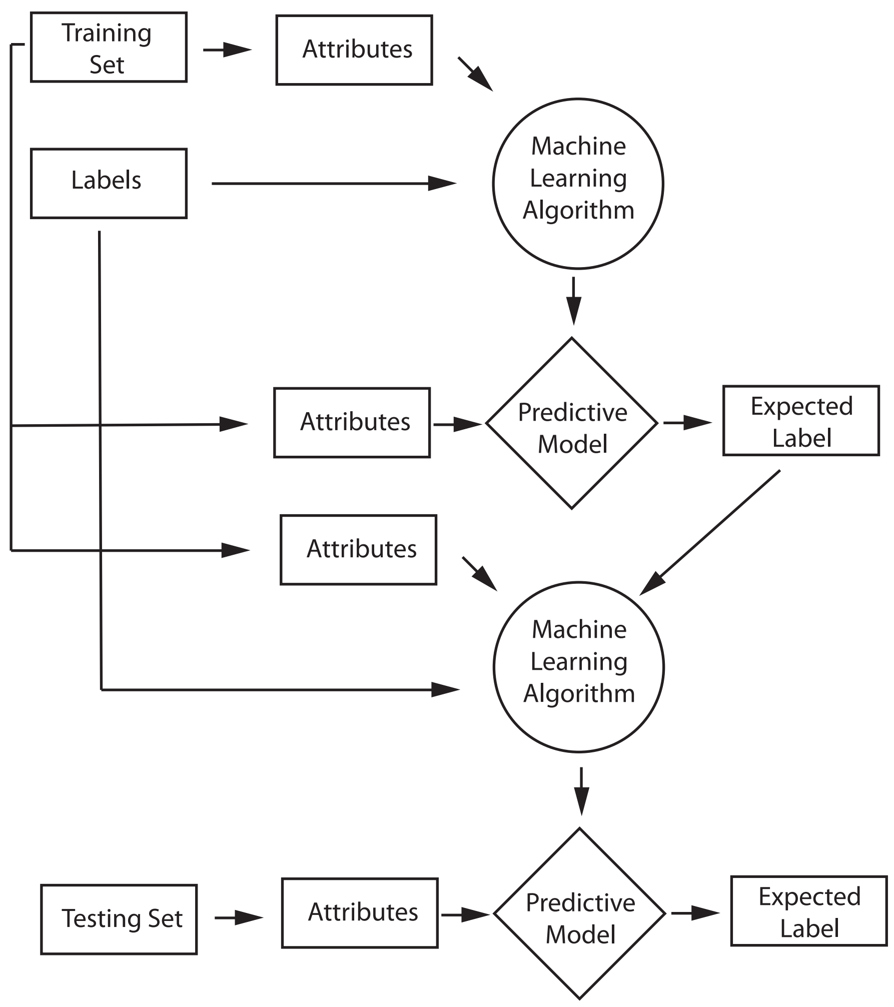


Figure 2.5: Flowchart for Semi-supervised Learning

- **Transductive Learning:** The goal of transductive classifiers is to infer the correct labels for the given unlabeled data $x_{l+1}, \ldots, x_{l+u}$ only.
- **Inductive Learning:** The goal of inductive classifiers is to infer the correct mapping from X to Y.

Figure 2.5 shows the flowchart of transductive semi-supervised classification. The classifier uses both unlabeled data along with labeled data to approximate the target function. Then labels are assigned to the unlabeled data items.

In order to make any use of unlabeled data, we must assume some structure to the underlying distribution of data. Semi-supervised learning algorithms make use of at least one of the following assumptions.

- **Smoothness Assumption:** Points which are close to each other are more likely to share a label. This is also generally assumed in supervised learning and yields a preference for geometrically simple decision boundaries. In the case of semi-supervised learning, the smoothness assumption additionally yields a preference for decision boundaries in low-density regions, so that there are fewer points close to each other but in different classes.
- **Cluster Assumption:** The data tend to form discrete clusters, and points in the same cluster are more likely to share a label (although data sharing a label may be spread across multiple clusters). This is a special case of the smoothness assumption and gives rise to feature learning with clustering algorithms.
- **Manifold Assumption:** The data lie approximately on a manifold of much lower dimension than the input space. In this case we can attempt to learn the manifold using both the labeled and unlabeled data to avoid the curse of dimensionality. Then learning can proceed using distances and densities defined on the manifold. The manifold assumption is practical when high-dimensional data are being generated by some process that may be hard to model directly, but which only has a few degrees of freedom. For instance, speech output is controlled by a series of vocal tubes, and images of various facial expressions are controlled by a few muscles. We would like in these cases to use distances and smoothness in the natural space of the generating problem, rather than in the space of all possible acoustic waves or images respectively.

2.3 Binarization

The multiclass classification problem can be solved in a number of ways. Unfortunately the best technique of doing so still remains an open issue [1]. One popular approach is to use a set of binary classifiers called *base classifiers*. As discussed before, a binary classification problem deals with only two classes. Thus a binary classifier learns to differentiate only between two classes. Moreover, the base classifiers can be trained in parallel. Using a set of binary classifiers to solve the multiclass problem is known as *binarizaiton*. Binarization not only helps in producing simple classifiers, but it also has the advantage of reduced training time. The two main binarization techniques are: one-vs-one (OVO) and one-vs-all (OVA).

- **OVO:** In the OVO approach, each binary classifier learns to differentiate between two classes only. The training set for each base classifier thus have the training data items for those two classes only. The results of these base classifiers are then combined to get the final output.

- **OVA:** As the name suggests OVA teaches each base classifier to learn exactly one class. Thus the training set for each base classifier is the same. But it contains only two classes, the class the base classifier needs to learn called the target class and the default class, which comprises of all the remaining classes.

As mentioned before, different OVO approaches vary mainly in their aggregation strategy. In recent years, different methods to combine the outputs of the base classifiers from these strategies have been developed. We assume that a base classifier that differentiates between two classes i and j gives two probability estimates r_{ij} and r_{ji} as output for a given data item. r_{ij} is the probability of that data item belonging to class i and r_{ji} is the probability of that data item belonging to class j, i.e., $1 - r_{ij}$. Also the number of total classes is m. The aggregation strategies are discussed as follows:

- **Voting Strategy (VOTE) [6]:** It is also known as binary voting and Max-wins rule. Each base classifier gives a vote for the predicted class. The votes given for each class is counted and the class with the highest number of votes is predicted as the output class.

$$Class = arg \max_{i=1,\ldots,m} \sum_{i \leq j \neq i \leq m} s_{ij} \tag{2.2}$$

where s_{ij} is 1 when $r_{ij} > r_{ji}$ and 0 otherwise.

- **Weighted Voting Strategy (WV):** This is a modification of the VOTE strategy. Each base classifier votes for both of the classes. A weight is assigned to the vote according to the confidence of the classifier in predicting the class. The class with the largest sum value is the final output class.

$$Class = arg \max_{i=1,\ldots,m} \sum_{i \le j \ne i \le m} r_{ij} \tag{2.3}$$

- **Classification by Pairwise Coupling (PC) [7]:** This method estimates the joint probability for all classes from the pairwise class probabilities of the binary classifiers. Hence, when r_{ij}=Prob($Class_i|Class_i$ or $Class_j$), the method finds the best approximation of the class posterior probabilities $\hat{p} = (\hat{p}_1, \ldots, \hat{p}_m)$ according to the classifiers outputs. The class with the largest posterior probability is predicted:

$$Class = arg \max_{i=1,\ldots,m} \hat{p}_i \tag{2.4}$$

To compute the posterior probabilities the Kullback Leibler (KL) distance between r_{ij} and μ_{ij} is minimized:

$$l(p) = \sum_{i \le j \ne i \le m} n_{ij} r_{ij} \log \frac{r_{ij}}{\mu_{ij}} = \sum_{i<j} n_{ij} \left(r_{ij} \log \frac{r_{ij}}{\mu_{ij}} + (1 - r_{ij}) \log \frac{1 - r_{ij}}{1 - \mu_{ij}} \right) \tag{2.5}$$

where $\mu_{ij} = p_i/(p_i + p_j)$, $r_{ij} = 1 - r_{ij}$ and n_{ij} is the number of training data in the ith and jth classes.

- **Decision Directed Acyclic Graph (DDAG) [8]:** DDAG method constructs a rooted binary acyclic graph where each node is associated to a list of classes and a binary classifier. In each level a classifier discriminates between two classes, and the class that is not predicted is removed. The last class remaining on the list is the final output class.

- **Learning valued preference for classification (LVPC) [9]:** This method considers the score matrix as a fuzzy preference relation; based on fuzzy preference modeling, the original relation is decomposed into three new relations with different meanings, the strict preference, the conflict and the ignorance. A decision rule based on voting strategy is proposed to obtain the output class from them:

$$Class = arg \max_{i=1,\dots,m} \sum_{i \leq j \neq i \leq m} P_{ij} + \frac{1}{2}C_{ij} + \frac{N_i}{N_i + N_j} I_{ij} \tag{2.6}$$

where N_i is the number of examples from class i in the training data (and hence, an unbiased estimate of the class probability), C_ij is the degree of conflict (the degree to which both classes are supported),I_ij is the degree of ignorance (the degree to which none of the classes are supported) and finally, P_ij and P_ji are, respectively, the strict preference for i and j. Preference confidence and ignorance degrees are computed as follows:

$$\begin{aligned}
P_{ij} &= r_{ij} - \min\{r_{ij}, r_{ji}\} \\
P_{ji} &= r_{ji} - \min\{r_{ij}, r_{ji}\} \\
C_{ij} &= \min\{r_{ij}, r_{ji}\} \\
I_{ij} &= 1 - \max\{r_{ij}, r_{ji}\}
\end{aligned}$$

- **Preference relations solved by Non-Dominance Criterion (ND) [10]:** The Non-Dominance Criterion was originally defined for decision making with fuzzy preference relations.In this case, as in LVPC, the score matrix is considered as a fuzzy preference relation. The relation has to be normalized. Then the degree of non-dominance is computed(the degree to which the class i is dominated by none of the remaining classes) and the class with the largest degree is predicted.

- **Binary tree of classifiers (BTC):** The idea behind this method is to reduce the number of classifiers and increase the global accuracy using some of the binary classifiers that discriminate between two classes, to distinguish other classes at the same time. The tree is constructed recursively and in a similar way to the DDAG approach, each node has associated a binary classifier and a list of classes. But in this case, the decision of the classifier can distinguish other classes as well as the pair of classes used for training. So, in each node, when the decision is done, more than one classes can be removed from the list. In order to avoid false assumptions, a probability is used when the examples from a class are near the discriminant boundary, so the class cannot be removed from the lists in the following level.

- **Nesting one-vs-one (NEST) [11]:** This method is directly developed to tackle the unclassifiable region produced in voting strategy (it is easy to see that in a three class problem, if each binary classifier votes for a different class, there is no winner, so some tie-breaking technique has to be applied). Nesting OVO uses the voting strategy, but when there exist examples within the unclassifiable region, a new OVO system is constructed using only the examples in the region in order to make them classifiable. This process is made until no examples remain in the unclassifiable region of the nested OVO.

- **Probability estimates by pairwise coupling approach (PE) [12]:** PE is similar to PC, which also estimates the posterior probabilities (p) of each class starting from the pairwise probabilities. In this case, while the decision rule is equivalent (predicting the class with the largest probability), the optimization formulation is different. PE optimizes the following problem:

$$\min_p \sum_{i=1}^{m} \sum_{i \leq j \neq i \leq m} (r_{ji}p_i - r_{ij}p_j)^2 \text{ subject to } \sum_{i=1}^{k} p_i = 1, p_i \geq 0, \forall_i \tag{2.7}$$

Now we briefly describe the combination methods to obtain the predicted class for the OVA approach. In this case the aggregation is developed to deal with the ties when more than one classifiers give a positive answer, in other case the answer is given by the classifier giving a positive answer. A short description of each method which we have employed in the experimental study follows:

- **Maximum confidence strategy (MAX):** It is similar to the weighted voting strategy of the OVO systems. Each base classifier gives as output a probability for the target class. The class with the highest probability is the predicted output class.

$$Class = arg \max_{i=1,\ldots,m} r_i \tag{2.8}$$

- **Dynamically ordered one-vs-all (DOO) [4]:** This method does not base its decision on the confidence of OVA classifiers. In this case, a Naive Bayes classifier is also trained (using sample from all classes) together with all other classifiers. This new classifier establishes the order in which the OVA classifiers are executed for a given pattern. Then, the instance is submitted to each OVA classifier in that order until a positive answer is obtained, which indicates the predicted class. This is done dynamically for each example. In this manner, ties are avoided a priori by the Naive Bayes classifier instead of relying on the degree of confidence given by the outputs of the classifiers.

OVA techniques have been used on multiple problem domains over the years. Their implementation for these different domain vary significantly from one another. The authors of [4] proposed an OVA strategy to classify fingerprints. Their proposed algorithm uses support vector machine (SVM) for classification. They train a Naive Bayes classifier in parallel with the base classifiers. This classifier establishes a sequence in which the OVA base classifiers will be executed for a given data item. A data item is given to the base classifiers in the established sequence until a positive answer is obtained. This is the output class. The rest of the classifiers are not used for that data item. So, ties are avoided a priori by using the Naive Bayes classifier. [13] uses a text-query based method for Chinese handwriting detection using SVM. Each base classifier provides a probability for one class. The class with the maximum probability is the output class. The authors of [29] investigated the problem of video categorization and Delechaux et al. [14] use neural networks as their base classifier to recognize indoor activities.

It is quite apparent from the discussions above that binarization is an efficient approach to solve the multiclass problem. The main reason behind this is that each base classifier has to deal with a simpler decision boundary than the original problem. This is evident from Figure 2.6. Here the binarization approach used is OVA. As can be seen from the figure, the decision boundary of the original multiclass problem is a complex one. It may also not

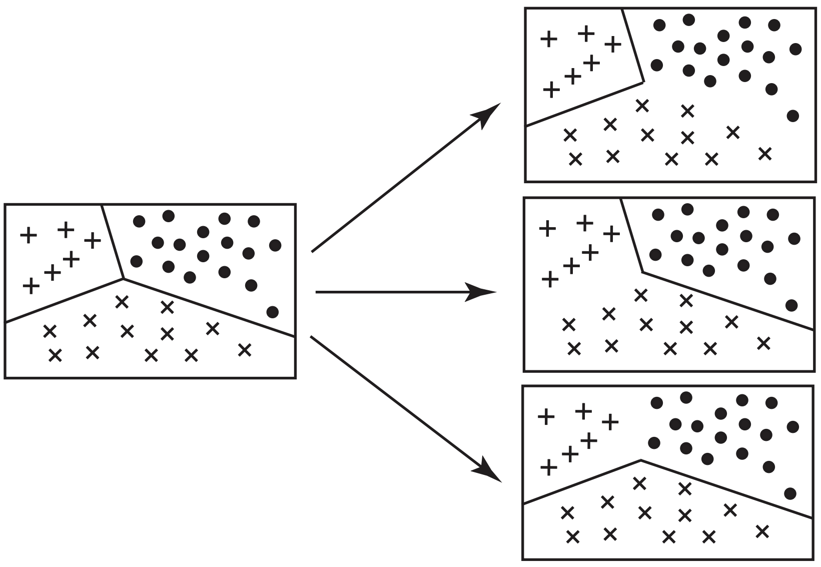


Figure 2.6: Decision Boundary for OVA

be linear. But individual base classifiers are given a data set with only two classes. So, the decision boundary that needs to be learned becomes much simpler as is evident from the figure. Hence, the classification task for binarization also becomes simpler. Moreover, there are many popular binary classifiers available. So, the individual classification task can be learnt with higher accuracy too. Binarization not only helps in producing simple classifiers, but it also helps in reducing time to train classifiers. This is because there is a chance to train the binary classifiers in parallel.

But binarization has its drawbacks too. It can also be seen from the figure after performing binarization for each base classifier, the number of data items for the target class is significantly less than that of the default class. This gives rise to the class imbalance problem.

In binary classification class imbalance [5] occurs when one of the two classes has significantly more training data available than the other class. The class with more training data items is called the *major* class and the class with lesser training data items the *minor* class. Class imbalance causes the bias of the algorithm to shift towards the major class, i.e., given a new data item the algorithm will more likely predict its class to be the major class irrespective of the attribute set. So, the algorithms shows very poor classification rates for the minor class. Class imbalance has been known to cause reduced accuracy among classifiers. This problem has severe repercussions for applications such as medical diagnosis, fraud detection, illegal network intrusions and predicting failures in technical equipment.

The class imbalance problem almost always occurs when binarization is performed. It reduces the chance of a base classifier to learn the target class properly. The problem will be more severe if the original data set is itself imbalanced. Since, the main goal of a base classifier is to learn a particular target class, class imbalance has significant impact on the overall performance. In this thesis, we hope to tackle this problem arising from binarization.

2.4 Boosting

Boosting [36] is a machine learning meta-algorithm for reducing bias. The main goal of applying boosting is to make a *weak* classifier stronger. A weak classifier is defined to be a classifier whose correlation to the true classification is weak. A strong classifier, on the other hand, is a classifier that is arbitrarily well-correlated with the true classification. Boosting tries to improve the performance of a weak classifier iteratively. The goal is to make the classifier more focused on the data items in the training set that it misclassifies. It can be said that these data items carry more weight during the next iteration.

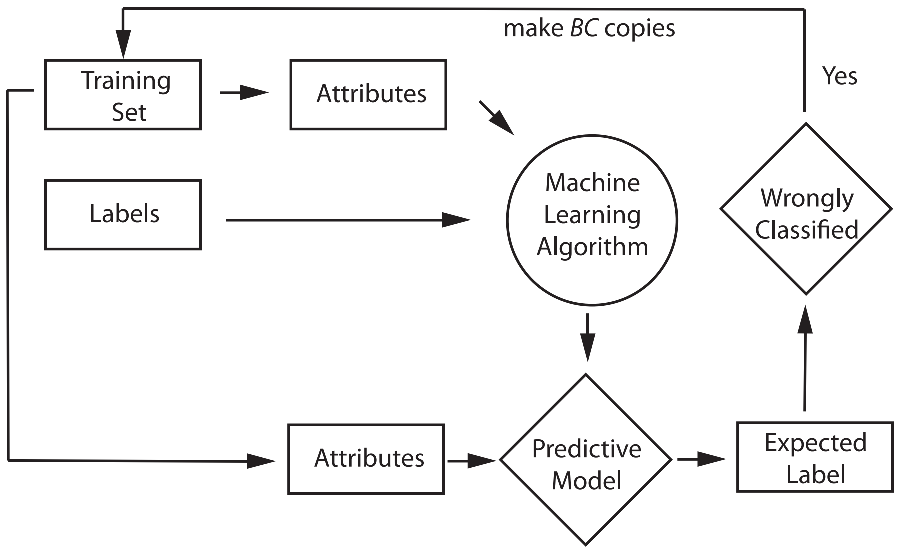


Figure 2.7: Flow chart for Boosting

One approach to achieve this can be to make multiple copies of the data items that have been misclassified in the previous iteration and include them in the training set in the next iteration. A flow chart of this approach is shown in Figure 2.7. The pseudocode is shown in Algorithm 2.1. At the beginning of each iteration, the classifier is trained. Then we check if data items are misclassified. If that is the case, then BC copies of those misclassified data items are made and added to the training set that is used to train the classifier. Thus these misclassified data items are given more weight during the next iteration.

Algorithm 2.1 Pseudocode for Boosting

```
1:  $C \leftarrow$ the classifier
2:  $I \leftarrow$ total number of iterations
3:  $BC \leftarrow$ number of times a data item would be copied for boosting
4:  $TS \leftarrow$ the training set
5:  $CTS \leftarrow \text{Copy}(TS)$    //making a copy of $TS$
6:  for $i = 1$ to $I$ do
7:      $C \leftarrow \text{Train}(C, CTS)$
8:      for $j = 1$ to $\text{Size}(TS)$ do
9:          if $\text{Test}(C, TS[j]) \neq TS[j].class$   //if misclassified then
10:             for $l = 1$ to $BC$ do
11:                 $CTS \leftarrow CTS \cup \text{Copy}(TS[j])$
12:             end for
13:         end if
14:     end for
15: end for
```

Boosting is not algorithmically constrained. So, there are many other approaches to improve the weight of certain data items. Some algorithms even decrease the weight of data items that are repeatedly misclassified by the classifier [37]. Some boosting approaches also decrease the weight of correctly classified data items.

2.5 Oversampling

Oversampling in data analysis is a specific technique to handle the class imbalance problem. This technique is used to adjust the class distribution of a data set. This involves using a bias to select more samples from one class than from another. Oversampling tries to realize the underlying distribution of data.

As stated before, class imbalance occurs the number of data items for one class is significantly less than the other class. Hence, the classifier does not properly learn to classify minor class data items. Oversampling is used to balance this situation. It tries to create synthetic data for the minor class. These new data items are then added to the training set. The increase in the number of data items for the minor class reduces the bias of the algorithm, which leads to better performance. There are various oversampling techniques available. One popular approach is called Synthetic Minority Over-sampling Technique (SMOTE) [38]. Another effective approach is Majority Weighted Minority Oversampling Technique for Imbalanced Data Set Learning (MWMOTE) [39].

Undersampling is also another popular technique to handle the class imbalance problem. It is opposite and roughly equivalent to oversampling. Undersampling does not rely on the creation of synthetic data. As the name suggests, undersampling techniques remove data items of the major class from the training set to create a balance between major and minor class. While removing the data items we also need to be careful not to remove boundary data items as these data items dictate the decision boundary for a class. So, choosing the correct data items to remove is the main challenge for undersampling.

2.6 Popular Multiclass Classifiers

In the early days of designing classifiers, the main focus used to be binary classification. But that trend has changed. Over the years many popular multiclass classifiers have been developed. The approach taken by different classifiers vary a great deal. Since, our thesis deals with multiclass classification, in this section, we discuss several popular multiclass classifiers. These classifiers belong to various classifier families. They are discussed as follows.

2.6.1 Neural Network

Artificial neural network also known as neural network (NN) [30] in short are computational models capable of handling a wide range of tasks related to machine learning. They are designed after the central nervous system, specially the brain. NNs are usually presented as systems of interconnected neurons. A large number of neurons work in unison to solve a specific problem. Development of NNs date back to early 1940s. They experienced an upsurge in popularity in late 1980s.

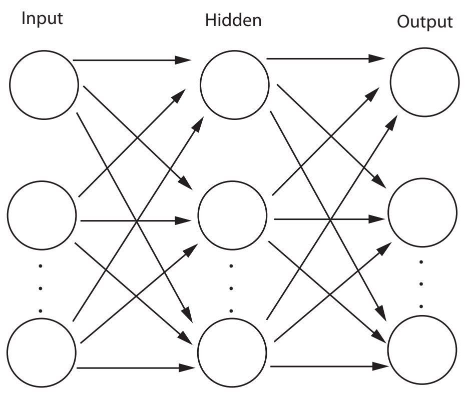


Figure 2.8: A Two Layer Artificial Neural Network

The salient feature of NN is that its computation is collective, asynchronous and parallel while memory is distributed, internalized short term and content addressable. Its learning process is adaptive and it can create its own organization of the information it receives during the training process. NNs are also fault tolerant. It is capable of handling redundancy very well. So, NNs perform quite well when the data is error prone. NNs are specially useful for classification and function approximation problems.

Figure 2.8 shows the typical organization of a two layer NN. There is an input layer, an output layer and a hidden layer. It is called a two layered organization because no actual processing is done in the input layer. This number of neurons for each layer can vary and are independent of each other. If the number of neurons in the output layer is two, then the NN is a binary classifier. If it is greater than two, then the NN is created for multiclass classification. It can be seen from the figure that each neuron from a given layer is connected to all the neurons in the next layer.

Each connection between neurons is associated with a weight. These weights are usually initialized at random. Then they are tuned by a learning algorithm which focuses on reducing the overall error. The adaptive weights are conceptually connection strengths between neurons, which are activated during training and prediction.

NNs have been a popular choice for researchers to solve the multiclass problem over the years. They are error tolerant and also capable of approximating non-linear function. The only drawback of NNs is that its hard to explain the output function of a NN.

2.6.2 Decision Tree

Decision tree is a popular branch of classification algorithm. This branch of algorithms was introduced to approximate discrete-valued target functions. As the name suggests the learned function is represented by a decision tree. These algorithms have been successfully applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.

A data item is classified in a decision tree by sorting them down the tree from the root to some leaf node. The leaf node provides the classification of the item. Each node in the tree specifies a test of one attribute of the data item. Each of the branches descending from that node corresponds to one of the possible values of that attribute. So, a data item is classified by starting at the root node, testing the attribute specified by the node, then moving down the branch which correspond to the value of the attribute for the data item. This process is repeated until a leaf node is reached. Then the instance is classified.

The main difference between the different decision tree algorithms is how they choose an attribute for a node in the decision tree. Usually an attribute that can give a high *gain* value in terms of making decisions should be near the top of the decision tree. This way the probability of making a the classification decision without going too deep in the tree becomes higher. One simple yet elegant method used for making decisions about attributes was invented by R. Quinlan [31]. His algorithm was known as Iterative Dichotomiser 3 (ID3). At each step ID3 calculates the entropy of each unused attribute. If the number of classes is m then the entropy of an attribute A can be calculated as follows:

$$Entropy(A) = \sum_{i=1}^{m} -p_i \log_2 p_i \tag{2.9}$$

Here p_i is the proportion of data items belonging to class i. The attribute with the least entropy i.e., highest gain is chosen as the attribute for that node. The training data is then split according to this attributes values. This process is repeated until a decision about the class can be made.

C4.5 is an extension of ID3. It also builds on the knowledge of entropy. But instead of using entropy directly, C4.5 uses the normalized information gain. C4.5 is the most popular decision tree algorithm. In case of C4.5 another decision can be made after building the tree. Sometimes, the trees built can be very large. In such cases the tree may be pruned. As the attributes used in the decision tree must be nominal, special provision must also be made for continuous attribute. This is done by splitting the continuous attribute into a series of ranges. C4.5 has been known to be robust in handling data items missing attributes and containing error.

2.6.3 *k*-Nearest Neighbor

k-nearest neighbor (*k*-NN) [32] algorithm is an instance-based learning method. Learning in such algorithms consists only in storing the training data. When a new data item needs to be classified, a set of similar related instances is retrieved from memory and used to classify the new query instance. In case of *k*-NN the *k* nearest neighbor of the given data item are retrieved. The majority of class of these *k* data items is predicted to be the class label of that given data item. So, the performance of this algorithm mainly depends on the value of *k*.

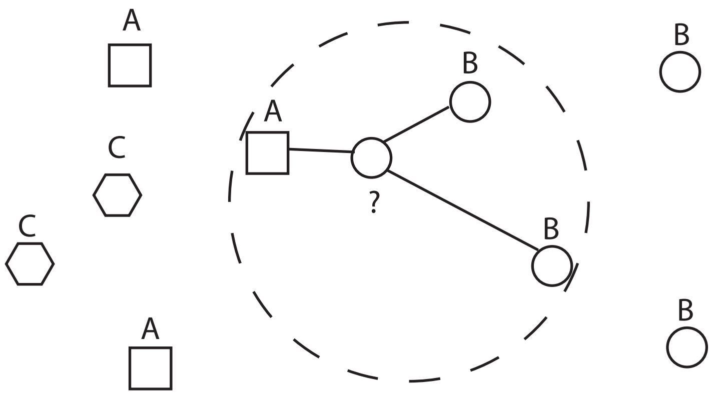


Figure 2.9: A *k*-NN (*k*=3) Classifier

Figure 2.9 shows an example of *k*-NN where *k*=3 i.e., *k*-NN (*k*=3). Here all data items except the one labeled "?" are training data. There are three classes: A, B and C. As discussed before, during the training phase these items are simply stored in memory. Now, a new data item "?" arrives that needs to be classified. As the value of k is 3, three nearest neighbors of "?" is chosen from the training set, identified by the dashed circle. The majority class among these data items is B. So, the class of the data item is predicted to be B. In case all the attributes of the data items are numerical then a simple Euclidean distance measure can be used find the nearest neighbors. But that is not the case if some or all of the attributes are nominal. In such cases a distance measure that can normalize the nominal attribute values must be used. There are several approaches for this. One of the most widespread measure that is used is called Heterogeneous Value Difference Metric (HVDM) [33].

k-NN is also known as a lazy learning method, known as such because it defers the decision of how to generalize beyond the training data until each new data item is encountered. The main difference of k-NN with other approaches is that, it can construct a different approximation to the target function for each new data item. It does not try to find a general approximation over the entire data space. This has significant advantages when the target function is complex but still can be described with a collection of less complex local approximations. The main disadvantage of k-NN is that it requires a large storage memory as the whole training set needs to be saved all the time. Moreover, if all the attributes of the data items are not pertinent to classification then the performance of k-NN can be significantly bad.

2.6.4 Ripper

Repeated Incremental Pruning to Produce Error Reduction (Ripper) [34] is a popular rule based algorithm. As evident by the name rule based algorithms generate a set of rules from the training set. This rule set has the advantage of being easily understandable. Their performance is also usually better than decision tree learners. Moreover, they are easy to implement and prior knowledge can also be included. Their main disadvantage is that they scale poorly with training data and do not handle noisy data well.

Ripper was developed with the vision to tackle the problems of rule based algorithms. With the increase in training data other rule based classifiers suffer from overfitting. To overcome this shortcoming pruning can be used. There are several efficient pruning techniques available. Ripper uses Incremental Reduced Error Pruning (IREP) [35], which is an improvement of Reduced Error Pruning (REP), as its pruning algorithm of choice. REP uses a divide and conquer approach. It divides the available data into training and validation set. After developing the rules the pruning starts. The impact of pruning a rule is tested on the validation set. A rule is pruned if accuracy improves the highest. These procedure continues until accuracy decreases. IREP is an iterative version of REP. The data set is randomly split. Rules are also pruned directly after devising them.

Ripper uses IREP to obtain a rule set. In the next step, rule optimization takes place. Then IREP is used again to cover the remaining positive examples. Ripper creates a smaller initial rule set than other rule based algorithms or decision trees. Its optimization phase also takes place in linear time. Moreover, as it uses IREP and optimization, Ripper is also efficient in handling noisy data.

2.7 Summary

In this chapter, we gave a brief overview of the various classification tasks. We described both binary and multiclass classification problems. Then we showed the binarization approach of solving the multiclass problem. A comparative analysis of the two binarization approaches were given. Then the problems that arise due to binarization were introduced. Finally, we gave brief descriptions of the working principles of different popular multiclass classifiers.

Chapter 3

Proposed Method

3.1 Introduction

As discussed in the previous chapter, the main issue that arises from binarization is the class imbalance problem which is known for reduced accuracy. We here present a new framework for handling the problems of binarization. Although several techniques (e.g. oversampling and undersampling) have been proposed for solving the class imbalance problem, they are not used in conjunction with binarization. In this chapter, we look at the various main components of our framework and how they can be helpful in handling the class imbalance problem. Then we look at the major differences of our framework from other existing work.

3.2 Our Framework

Our framework uses a combination of *boosting* and *oversampling* to handle the class imbalance problem. Henceforth, we call our framework Binarization with Boosting and Oversampling (BBO). Boosting has not been used previously to solve the class imbalance problem. But in the BBO framework it is the main tool to solve this problem. This is done by applying boosting from the very beginning of our framework, while versampling is applied later. By this way boosting is given more importance.

Figure 3.1 shows an overview of how the BBO framework works. At first the training

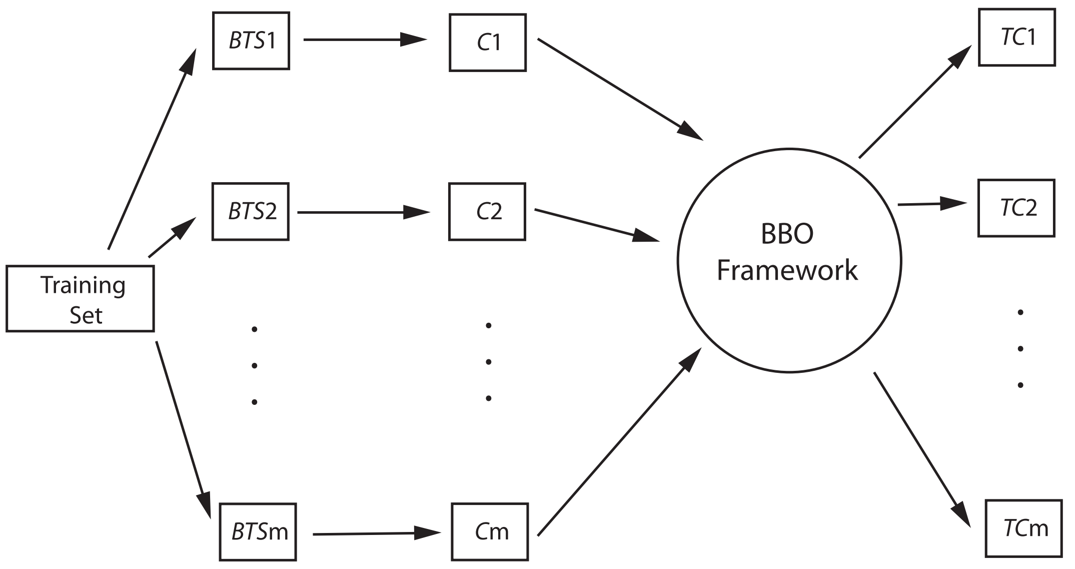


Figure 3.1: Flow Chart for BBO framework. Here $BTSi$, Ci and TCi refer to the i-th binarized training set, base classifier and trained base classifier respectively.

set is binarized. Let us assume that the number of classes are m. Since, BBO framework follows the OVA approach, the training set is divided into m binarized training sets. These training sets are created by copying the original training set and relabeling some of the data items. Each binarized training set has only two classes, the target class and the default class. Target class is the class that a base classifier tries to learn. All data items not belonging to the target class are labeled as the default class. The binarized training sets are identified as $BTS1$, $BTS2$, $\dots$, $BTSm$. These training sets are then paired with the individual base classifiers $C1$, $C2$, $\dots, Cm$. Each classifier and training set pair are then passed to the BBO framework for training. After training we get a set of m trained base classifiers namely $TC1$, $TC2$, $\dots$, TCm. Now we look at the various components of the BBO framework.

3.2.1 Training Procedure

We first look at how the BBO framework incorporates boosting and oversampling during the training of the base classifiers. The first step of binarization is to binarize the training set for each base classifier. During this step, the class imbalance problem arises as all data items not belonging to the target class is labeled as the default class. Let us assume that the number of classes is m. We also assume that in the training set each class has equal number of data items, n. After binarization of the training set, the target class will then contain n data items. However, the default class will have $n \times (m - 1)$ data items. So, the number of data items belonging to the default class becomes significantly greater compared to that of the target class. If the target class is imbalanced beforehand, the problem increases even more. This causes the base classifier to learn the target class poorly even though it is the goal of the base classifier to learn one particular target class.

The main goal of each base classifier is to learn the decision boundary for the target class while tackling the class imbalance problem. In the BBO framework, we solely focus on this goal. The classification task can thus be said to learn this decision boundary better with each iteration. To help the base classifier learn this decision boundary better, we change the training set in between iterations by using boosting and oversampling. After each iteration we check which data items of the target class in the training set the base classifier is not capable of correctly classifying. Then the focus of the base classifier should be to classify these misclassified data items of the target class during the next iteration. This is done through two steps.

We first apply boosting. For this purpose, in between iterations we include multiple copies of these misclassified target class data items in the training set so that the classifier learns these items better during the next iteration. The number of copies that would be made is predetermined. This value is denoted by BC. Since the classifier is taught the same target class data item multiple times we can assume that the chance of it being correctly classified will increase. But boosting alone can lead to overfitting. Overfitting occurs when a classifier has a significantly higher accuracy on the training set than on the testing set. As we are training the classifier to learn the same target class data item multiple times it is possible that the base classifier may overfit and learn only to classify those data items. Thus the overall accuracy would be significantly lower. Overfitting may affect the classifier even more severely if there is error in the training data. Then we might make multiple copies of the data items in the training set that have error which will in turn lead us away from the true decision boundary.

To reduce the chances of overfitting we also oversample the misclassified data items and also add these new synthetic data items into the training data set. This gives the base classifier some new data items to train itself with. This way the base classifier learns to classify the misclassified data items better. But oversampling can lead to another problem. Oversampling adds new synthetic data items to the data set. These data items may not be representative of the respective class. So, the decision boundary that the base classifier learns may not be ideal. That is why we give oversampling less importance than boosting. We do this by introducing oversampling only during the latter iterations of the learning period. Boosting is used in between each iteration. But oversampling is introduced after I_1 iterations. The percentage of oversampled data items is called OP. This value is also significantly lower compared to BC. Thus oversampling has far less importance than boosting.

Figure 3.2 shows how individual base classifiers in the BBO framework is trained. At first a copy of the training set ($BTSi$) is made called $CBTSi$. Let us assume that the base classifier, Ci will be trained for a total I iterations. Now, we use $CBTSi$ to train Ci. Then we test the data items from $CBTSi$ with Ci. The misclassified data items of the target class are stored in $FBTSi$. If the total number of iteration is less than I_1 then we apply boosting to $FBTSi$ and add these data items to $CBTSi$. Otherwise we apply both boosting and oversampling and add the data items $CBTSi$. Then Ci is trained again with $CBTSi$. This process is repeated for I iterations. The whole training process is also repeated for each base classifier.

Now let us look at the pseudocode of the whole training process for individual base classifiers. It is presented in Algorithm 3.1. As it can be seen a copy of the original training set is stored in $CBTSi$. This copy is used to train the base classifier. This is done because the performance of the base classifier is judged on the original training set. Moreover, boosting and oversampling is also applied on the original training set $BTSi$ and not on $CBTSi$. The new data items are then added to $CBTSi$. The main training process starts from line 10. At first, the base classifier is trained using $CBTSi$ (line 11). Then we check if the classifier can correctly classify the data items belonging to target class in the actual data set. If not, then the data items that are classified incorrectly are stored in $FBTSi$ (lines 13-19). These data items are then copied BC times and then added to $CBTSi$ (lines 20-24). If the number of iterations is greater than I_1 then oversampling is also applied to the dataset $FBTSi$ and the

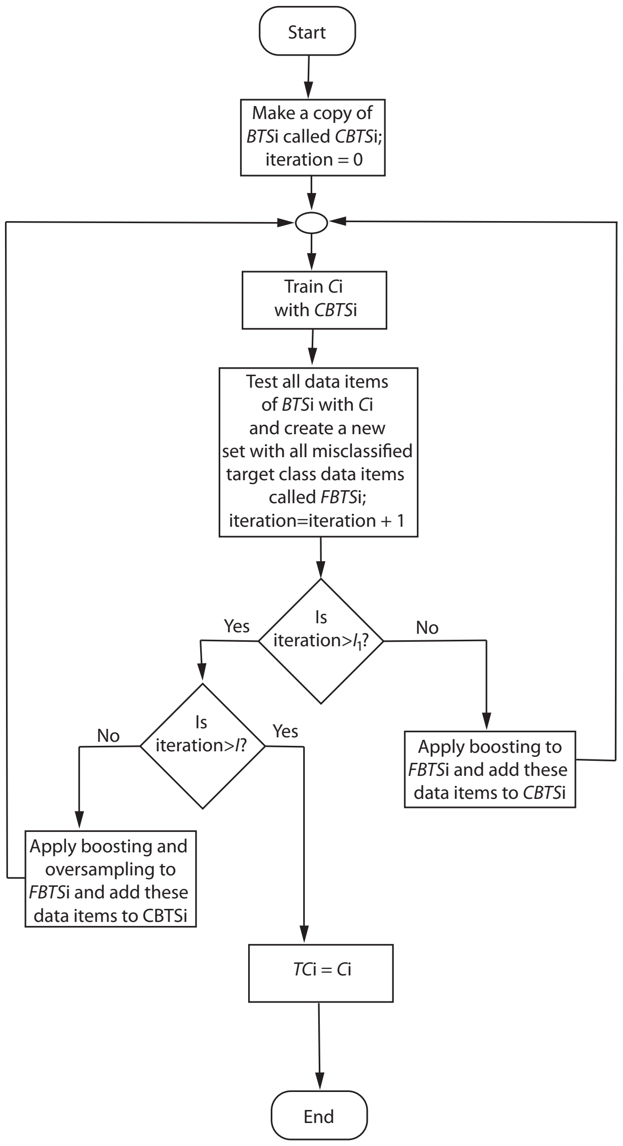

Start
Make a copy of *BTS*i called *CBTS*i; iteration = 0
Train *C*i with *CBTS*i
Test all data items of *BTS*i with *C*i and create a new set with all misclassified target class data items called *FBTS*i; iteration=iteration + 1
Is iteration>*I*1?
Yes
No
Is iteration>*I*?
No
Yes
Apply boosting to *FBTS*i and add these data items to *CBTS*i
Apply boosting and oversampling to *FBTS*i and add these data items to CBTSi
*TC*i = *C*i
End

Figure 3.2: Flow Chart for Training Base Classifiers

new synthetic data items are also added to $CBTSi$ (lines 25-27).

As discussed before, the BBO framework can also be applied for semi-supervised classification. However, the algorithmic framework for this case remains almost same. The main difference occurs during the selection of the data items of the training set on which boosting and oversampling is applied. Semi-supervised classification uses both labeled and unlabeled data during training. As we do not know the class of the unlabeled data, boosting and oversampling is not performed on them even though they can be said to be a part of the training data set.

Algorithm 3.1 Training Base Classifiers for Supervised Learning in the BBO Framework

```
1: $Ci \leftarrow$ the base classifier
2: $TCi \leftarrow$ target class that should be learned
3: $DCi \leftarrow$ all classes except the target class
4: $I \leftarrow$ total number of iterations
5: $BC \leftarrow$ number of times a data item would be copied for boosting
6: $I_1 \leftarrow$ iteration after which oversampling will start
7: $OP \leftarrow$ percentage of oversampled data created
8: $BTSi \leftarrow$ the binarized training set
9: $CBTSi \leftarrow$ Copy($BTSi$)   //making a copy of $BTSi$
10: **for** $k = 1$ to $I$ **do**
11:    $Ci \leftarrow$ Train($Ci, CBTSi$)
12:    $FBTSi \leftarrow \phi$
13:    **for** $j = 1$ to Size($BTSi$) **do**
14:       **if** $BTSi[j].class = TCi$   //if belongs to target class **then**
15:          **if** Test($Ci, BTSi[j]$) $\neq TCi$   //if misclassified **then**
16:             $FBTSi \leftarrow S \cup TS[j]$
17:          **end if**
18:       **end if**
19:    **end for**
20:    **for** $j = 1$ to Size($FBTSi$) **do**
21:       **for** $l = 1$ to $BC$ **do**
22:          $CBTSi \leftarrow CBTSi \cup$ Copy($FBTSi[j]$)
23:       **end for**
24:    **end for**
25:    **if** $k > I_1$ **then**
26:       $CBTSi \leftarrow CBTSi \cup$ Oversample($FBTSi, OP$)
27:    **end if**
28: **end for**
29: $TCi \leftarrow Ci$
30: **return** $Ci$
```

Boosting and oversampling is performed only on the labeled data. The rest of the training process remains the same. This process of training base classifiers presented in Algorithm 3.2. Here, LDi is the labeled data and UDi is the unlabeled data for the i-th base classifier. A copy of LDi called $CBTSi$ is made. Both $CBTSi$ and UDi are used to train the base classifier Ci. But we only use LDi to test the performance of Ci. Those data items of LDi that are misclassified are used for boosting and oversampling.

Algorithm 3.2 Training Base Classifiers for Semi-supervised Learning in the BBO Framework

```
1: $Ci \leftarrow$ the base classifier
2: $TCi \leftarrow$ target class that should be learned
3: $DCi \leftarrow$ all classes except the target class
4: $I \leftarrow$ total number of iterations
5: $BC \leftarrow$ number of times a data item would be copied for boosting
6: $I_1 \leftarrow$ iteration after which oversampling will start
7: $OP \leftarrow$ percentage of oversampled data created
8: $LDi \leftarrow$ the binarized labeled data set
9: $UDi \leftarrow$ the binarized unlabeled data set
10: $CBTSi \leftarrow$ Copy($LDi$)   //making a copy of $LDi$
11: for $k = 1$ to $I$ do
12:    $Ci \leftarrow$ Train($Ci, CBTSi, UDi$)
13:    $FBTSi \leftarrow \phi$
14:    for $j = 1$ to Size($LD$) do
15:       if $LD[j].class = TCi$   //if belongs to target class then
16:          if Test($Ci, LDi[j]$) $\neq TCi$   //if misclassified then
17:             $FBTSi \leftarrow S \cup LDi[j]$
18:          end if
19:       end if
20:    end for
21:    for $j = 1$ to Size($FBTSi$) do
22:       for $l = 1$ to $BC$ do
23:          $CBTSi \leftarrow CBTSi \cup$ Copy($S[j]$)
24:       end for
25:    end for
26:    if $k > I_1$ then
27:       $CBTSi \leftarrow CBTSi \cup$ Oversample($S, OP$)
28:    end if
29: end for
30: $TCi \leftarrow Ci$
31: return $TCi$
```

3.2.2 Aggregation Strategy

Since, DOO aggregation strategy is a bit uncommon for OVA binarization, we now look at how DOO works. As stated in Chapter 2, OVA mainly uses the MAX strategy. Here, all base classifiers provide a confidence level for their target class. The class with the highest confidence level is predicted as the output class. But letting the confidence level of individual base classifier to decide the predicted class can be a faulty approach since the base classifiers have limited information about the classes. Moreover if ties arise, then the predicted class is arbitrarily declared from the tied classes. DOO handles these problems of MAX. In MAX, a Naive Bayes classifier is trained alongside the base classifier. This classifier decides the order in which the base classifiers will be tested. The base classifier that gives a positive answer for its target first, predicts the output class. So, ties are avoided apriori. Furthermore, unlike the base classifiers, the Naive Bayes classifier has complete information about the training data. So, letting it choose the order of the base classifiers yields better results.

Algorithm 3.3 Dynamically Ordered One-vs-All (DOO) Algorithm

```
1: $B \leftarrow$ the Naive Bayes lassifier
2: $m \leftarrow$ number of classes
3: $BList \leftarrow$ the list of base classifiers
4: $CList \leftarrow$ an index array of size $m$ with values 1, 2, ..., $m$, to identify each base classifier
5: $TR \leftarrow$ the training set
6: $TE \leftarrow$ the test set
7: $B \leftarrow \text{Train}(B, T)$
8: $Output \leftarrow \phi$    //the list of output labels that will be predicted for $TE$
9: for $i = 1$ to $TE.size()$ do
10:    $PList \leftarrow B.\text{probablitiyDistribution}(TE[i])$    //storing a list of probability for the data
       item for each class
11:    $SList \leftarrow \text{sort}(CList, PList)$    //sorting the index array according to descending order
       of probability
12:    for $j = 1$ to $m$ do
13:       if $\text{Test}(BList[SList[j]], TE[i]) = SList[j]$ then
14:          $Output[i] \leftarrow Slist[j]$
15:          break
16:       end if
17:    end for
18: end for
19: return $Output$
```

Algorithm 3.3 shows the pseudocode for DOO aggregation. Here, B is the Naive Bayes classifier. We assume that the number of classes is m. After training the base classifiers are stored in an array called the *BList*. *SList* is an integer array. It stores the index of the base classifiers. This is the array that is rearranged to find the order in which the base classifiers will be tested. Q is the output array where the predicted class values are stored. At first B is trained with the training data. When a new data item needs to be classified, it is first passed to

B to find a probability distribution for each class for that data item. That distribution is stored in $PList$. Now, the $CList$ is sorted according to $PList$ from highest to lowest probability, i.e., the index of the class with the highest probability is stored in the lowest index of the $SList$ and the index of the class with the lowest probability is stored in the highest index of $SList$. This is done using the *sort* function. Then the base classifiers are tested in the order provided in $SList$. When one provides a positive answer, then the predicted class is saved and the testing stops. So, ties never arise.

3.2.3 Modification for Non-iterative Algorithms

The main idea of the BBO framework is to change the training data for the algorithm in between iterations. But unfortunately there are many algorithms that do not use iteration. Many popular algorithms like decision trees, k-nearest neighbor and others do not iteratively update a loss function. The target function is approximated as a whole. BBO framework can also be modified for such algorithms.

In cases like this, since it is not possible to update the training set while the algorithm is learning, the algorithm is actually retrained multiple times. The training set is updated after the algorithm has completed its learning each time. This process is repeated. The rest of the framework remains the same.

3.3 Difference with Existing Works

In this section, we look at how the BBO framework is different from other research work. They are discussed as follows:

- Class imbalance problem is a known issue that causes reduced performance among classifiers. But it has not been handled in the case of binarization before. BBO framework tackles this issue. Most of the literature mainly uses either oversampling and undersampling to handle the class imbalance problem. But unlike those approaches, BBO framework uses a combination of boosting and oversampling to handle the problem.

- Even though oversampling has been the choice tool of researchers to handle the class imbalance problem, BBO framework actually gives more focus to boosting than oversampling. Oversampling is introduced during the latter stages of the BBO framework while boosting is applied all through out. Moreover, the percentage of boosting is significantly higher than oversampling. In these ways, BBO framework gives more focus to boosting.

- Most literature that deals with classification usually works with one training set. The original training set may be updated or modified but the data items used to train a classifier does not change. That is not the case in the BBO framework. In the BBO framework the training set is constantly modified to improve the deficiencies of a base classifier. It is modified such that, the modified train set can help the base classifier learn misclassified training data items of the target class better. This is also a significant difference between the BBO framework and other existing classifiers. Furthermore, BBO framework can be implemented for any supervised classifier, irrespective of whether it learns iteratively or not.

- Moreover, as discussed earlier the BBO framework can be modified to be used with semi-supervised classification. Recently semi-supervised classification has received a lot of attention from researchers. As unlabeled data is cheap and easy to come by, they can help to improve the performance of classifiers significantly. But binarization has not been used with semi-supervised classification. BBO framework introduces binarization to semi-supervised classification. At the same time it also handles the class imbalance problem that may occur.

- Unlike most literature available on OVA binarization, BBO uses the DOO aggregation approach. DOO not only handles ties apriori, it is also known to give better performance than MAX. So, BBO framework is different from other OVA approaches in this regard also.

- Most of the available semi-supervised classifiers are binary classifiers. But here we show an approach to handle multiclass classification using semi-supervised classifiers.

Chapter 4

Experimental Studies

4.1 Introduction

This chapter evaluates BBO framework's performance on several well known data sets. We also compare our results found with several well known benchmark algorithms. Furthermore, the evaluation and comparison of our framework on semi-supervised classification problems are presented. We discuss the data sets, experimental details, results and comparisons as follows.

4.2 The Dataset

In this study, we selected 15 data sets from the UCI machine learning repository [40]. A summary of the data sets is given in Table 4.1. Since, we work with binarization all data sets have multiple classes. The table shows the number of examples, number of attributes, the number of numerical and nominal attributes and the number of classes. Some of the larger data sets (led7digit, nursery, pageblocks, penbased, satimage and shuttle) were stratified sampled at 10% in order to reduce the computation time. The instances that had missing values were removed before partitioning. Some data sets like led7digit, penbased, segment and others have an almost even distribution of data items for each class. For example, each of the seven classes of segment has an equal 330 data items. At the same time, there are data sets like autos, nursery, pageblocks and others where the class imbalance problem is present. For example, nursery has five classes. Three of them have over 400 data items, while the other two only have 1 and 32 data items respectively. This way we can deduce the effect of using BBO framework on data sets that suffer from class imbalance problem beforehand and those that do not. These data sets were also used for experimentation in [3]. We downloaded the data sets from http://sci2s.ugr.es/ovo-ova.

Table 4.1: Summary description of data sets.

Data Set	#Example	#Attributes	#Numeric	#Nominal	#Classes
autos	159	25	15	10	6
car	1728	6	6	0	4
cleveland	297	13	5	8	5
dermatology	366	33	1	32	6
ecoli	336	7	7	0	8
flare	1389	10	0	10	6
led7digit	500	7	0	7	10
lymphography	148	18	3	15	4
nursery	1296	8	0	8	5
pageblocks	548	10	10	0	5
penbased	1099	16	16	0	10
satimage	643	36	36	0	7
segment	2310	19	19	0	7
shuttle	2175	9	9	0	7
zoo	101	16	0	16	7

4.3 Performance Measures

In this section, we introduce the performance measures that would be used to evaluate the performance of the BBO framework with multiclass data sets. There are many well-known accuracy measures like: classification rate (accuracy), precision, sensitivity, specificity, G-mean, F-score, AUC, Youden's index Γ, Cohen's Kappa, etc. Different measures show the different properties of a classifier. For our experimental purposes we have chosen three measures namely classification rate, Cohen's Kappa and G-mean. We explain them below:

- **Accuracy** also called the classification rate, is the number of correctly classified instances compared to the total number of instances. It is the most commonly used metric for assessing performance of classifiers.

- **Cohen's Kappa** is an alternative measure to classification rate. It compensates for random correct classifications. Cohen's Kappa evaluates the portion of hits that can be attributed to the classifier itself relative to all the classifications that cannot be attributed to chance alone. It can be calculated by making use of the resulting confusion matrix (Table 4.2) in a classification task.

  Cohen's Kappa is calculated as follows:

$$\text{Kappa} = \frac{n \sum_{i=1}^{m} h_{ii} - \sum_{i=1}^{m} T_{ri} T_{ci}}{n^2 - \sum_{i=1}^{m} T_{ri} T_{ci}} \tag{4.1}$$

  where h_{ii} is the number of true positives for each class, n is the total number of examples, m is the number of class labels and T_{ri} and T_{ci} are the rows' and columns' total counts respectively. Cohen's Kappa ranges from -1 through 0 to 1. These values indicate total disagreement, random classification and perfect agreement respectively.

- **G-mean** is a measure of classification performance where each class is equally represented in the evaluation measure. It has been known to be an effective measure to evaluate the impact of imbalanced data. G-mean can be defined as follows for multiple classes:

$$\text{G-mean} = \left(\prod_{i=1}^{m} R_i \right)^{1/m} \tag{4.2}$$

  Here m is the number of classes. R_i represents the recall of the i-th class which is defined as:

$$R_i = \frac{h_{ii}}{T_{ri}} \tag{4.3}$$

where h_{ii} is the number of true positives for class i. T_{ri} represents the total number of instances for class i.

Table 4.2: Confusion Matrix.

	C_1	C_2	$\dots$	C_m	Total
C_1	h_{11}	h_{12}	$\dots$	h_{1m}	T_{r1}
C_2	h_{21}	h_{22}	$\dots$	h_{2m}	T_{r2}
$\vdots$			$\ddots$		$\vdots$
C_m	h_{m1}	h_{m2}	$\dots$	h_{mm}	T_{rm}
Total	T_{c1}	T_{c2}	$\dots$	T_{cm}	T

4.3.1 Statistical Analysis

We used Wilcoxon signed-rank test [41] for statistical hypothesis test on the experimental results. It is a popular non-parametric statistical hypothesis test which is an alternative to the paired Student's t-test. In this test, each pair of samples are first compared with each other. Based on the difference of these samples, a ranking is provided. Tied pairs are ignored. Based on these ranks and the difference, a sum of of the signed ranks is made. Then, using empirical data we can find if the null hypothesis, which states that there is no relationship between two measured phenomena, can be rejected. The whole process is discussed as follows.

Let N be the number of pairs. Also for $i = 1, \dots, N$, let $x_{1,i}$ and $x_{2,i}$ denote the measurements. Now we calculate $|x_{2,i} - x_{1,i}|$ and $\text{sgn}(x_{2,i} - x_{1,i})$ for all pairs where sgn is the sign function. Pairs with $|x_{2,i} - x_{1,i}| = 0$ are excluded. Let the reduced number of pairs be N_r. These pairs are then ordered according to their absolute difference value from smallest to largest. They are then ranked. Let the rank of pair i be denoted by R_i. Now, W is calculated as $W = |\sum_{i=1}^{N_r}[\text{sgn}(x_{2,i} - x_{1,i}) \times R_i]|$. The sampling distribution of W converges to a normal distribution with increasing N_r. When $N_r \geq 10$, a z-score can be calculated as $z = \frac{W-0.5}{\sigma_w}$ where $\sigma_w = \sqrt{\frac{N_r(N_r+1)(2N_r+1)}{6}}$. The null hypothesis can be rejected if this z-score is greater than a certain critical value. If, $N_r < 10$, W is compared with a critical value from a reference table.

4.4 Experimental Setup

We devised three experimental setups namely Plan 1, Plan 2 and Plan 3 for our experimental study. Plan 1 is for supervised classification. Plan 2 and Plan 3 are for semi-supervised classification. We used the popular SMOTE [38] algorithm for oversampling. For all our experimental studies we used the Waikato Environment for Knowledge Analysis (Weka) tool [42]. It is data mining tool written in java. The setups are described as follows.

- **Plan 1**

  As mentioned above Plan 1 is for supervised classification. We try to find the effect of our framework on various popular supervised classifiers like neural network, C4.5, k-nearest neighbor and Ripper. Among these classifiers only NN has natural iterations. For other algorithms we use the modification mentioned in Chapter 3.2.3. For each classifier we find the performance measure values for their binarized version (B), binarization with boosting version (BB), binarization with oversampling version (BO) and BBO framework. We also compare the results of the BBO framework with other popular binarazied algorithms. For NN, I was set to 1000 for all algorithms. I_1 was set to 500. BC was set to 100 for BB-NN and BBO-NN. For BO-NN and BBO-NN, OP was set to 400. For the other algorithms I and I_1 was set to 10 and 5 respectively. The other parameters remain the same. The performance measures were obtained by means of a five-fold cross-validation. The data partitions used can be found in [43] and http://sci2s.ugr.es/ovo-ova.

- **Plan 2**

  Plan 2 is for semi-supervised classification. We modify the supervised algorithms mentioned in Plan 1, so that they become semi-supervised classifiers. The modification is

Table 4.3: Parameter Specification for the BBO Framework.

Plan	Classifier	I	I_1	BC	OP
Plan 1	NN	1000	500	100	400
	C4.5, k-NN, Ripper	10	5	100	400
Plan 2	NN	1000	500	100	400
	C4.5, k-NN, Ripper	10	5	100	400
Plan3	LLGC	3	2	100	400

pretty simple. The steps are as follows:

- The unlabeled data is split into two sets $T1$ and $T2$
- $T1$ and $T2$ are randomly labeled according to the class distribution of the labeled data
- Two supervised classifiers are trained using (labeled data + $T1$) and (labeled data + $T2$)
- The classifier with the highest class probability for an unlabeled data item decides the class of that data item

The performance measures for each data set was averaged over 10 trials. For each trial, each data set was divided into partitions with 10%, 20%, 30%, 40% and 50% labeled data randomly. The task of the classifier was to find the label of the unlabeled data set i.e., transductive learning. For each classifier we find the performance measure values for their multiclass version (M), binarized version (B), binarization with boosting version (BB), binarization with oversampling version (BO) and BBO framework. The parameters for BBO framework does not change from Plan 1 to Plan 2.

- **Plan 3**

  Plan 3 is also designed for semi-supervised classification. But unlike Plan 2 we do not modify supervised classifiers in Plan 3. We use a popular semi-supervised classifier for

Table 4.4: Parameter Specification for Base Classifiers.

Algorithm	Parameters
NN	Learning rate =0.3
	Momentum = 0.2
C4.5	Prune=true
	Confidence level = 0.25
	Minimum number of item-sets per leaf = 2
k-NN	Distance metricheterogeneous value difference metric (HVDM)
Ripper	Size of growing subset = 100%
	Repetitions of the optimization stage = 2
LLGC	$\alpha = 0.99$
	$\sigma = 1.0$

Plan 3. This classifier is LLGC. We chose LLGC as our semi-supervised classifier of choice as it is the only available multi-class semi-supervised classifier available in WEKA. As LLGC does not have any natural iterations we retrain the classifier after each iteration. The total number of iterations (I) used for LLGC was 3. BC was set to 100, I_1 to 1 and OP to 400. We have compared our algorithm with four baseline algorithms: a multi-class LLGC (M), a binarized LLGC (B), a binarized LLGC that uses only boosting (BB) and a binarized LLGC that uses only oversampling (BO). For the latter two algorithms I was set to 3. BC was set to 100 for BB-LLGC. For BO-LLGC, OP was set to 400 The performance measures for each data set was averaged over 10 trials. For each trial, each data set was divided into partitions with 10%, 20%, 30%, 40% and 50% labeled data randomly. The task of the classifier was to find the label of the unlabeled data set i.e., transductive learning.

In Table 4.3, we present the parameters used in the BBO framework. Table 4.4 shows individual parameters used for each base classifiers. The selected values are common for all experimental purposes. They were selected according to the recommendation of the corresponding authors of each algorithm. We considered four configurations for the k-NN algorithms: using 1, 3, 5 and 7 neighbors respectively.

4.5 Results

This section presents our experimental results for all three plans discussed in the previous section.

4.5.1 Results for Plan 1

Table 4.5 and 4.6 show a summary of the performance measures for Plan 1. The best result for each algorithm, for each measure is shown in bold. We denote the binarzied version of the algorithm, binarized with boosting version of the algorithm, binarized with oversampling version of the algorithm and BBO framework version of the algorithm with B, BB, BO and BBO respectively.

As it can be seen from the tables, the BBO framework performs the best for every algorithm other than k-NN (k=7) for all three performance measures. For k-NN (k=7) the binarized version of the algorithm with boosting performs the best. As the number of k increases, the chance of using more synthetic data for classification also increases for k-NN. This may have lead to reduced performance by the BBO framework in case of k-NN (k=7).

Now we give our statistical analysis for Plan 1. We provide our Wilcoxon's signed-rank test results. We choose α value 0.1 for our analysis. Table 4.7 and 4.9 show if the null hypothesis can be rejected for this value for each classifier. "R" means that the null hypothesis can be rejected in favor of the second algorithm. "A" means that the null hypothesis cannot be rejected. Since our target is to improve the performance of the binarized classifier, we compare between binarezed vs binarization with boosting (B vs BB), binarized vs binarization with oversampling (B vs BO), and binarized vs BBO framework (B vs BBO). Since the overall performance of binarization with boosting is quite good as seen from the given results we also compare between binarization with boosting vs BBO framework (BB vs BBO).

As can be seen from Table 4.7-4.9, for NN, the null hypothesis can be rejected for B vs

Table 4.5: Summary result for Plan 1 based on accuracy and Cohen's Kappa. The best results are shown in bold.

Algorithm	Accuracy (%)				Cohen's Kappa			
	B	BB	BO	BBO	B	BB	BO	BBO
NN	84.86 ± 4.12	84.77 ± 4.24	85.22 ± 5.01	**87.41** ± **3.55**	0.7746 ± 0.0751	0.7765 ± 0.0812	0.7657 ± 0.0906	**0.7851** ± **0.0785**
C4.5	83.14 ± 4.16	84.42 ± 3.75	82.49 ± 3.90	**85.03** ± **3.57**	0.7340 ± 0.0633	0.7656 ± 0.0611	0.7355 ± 0.0617	**0.7720** ± **0.0594**
k-NN (*k*=1)	79.87 ± 3.53	79.87 ± 3.53	81.53 ± 3.46	**81.53** ± **3.46**	0.6450 ± 0.0624	0.6450 ± 0.0624	0.6903 ± 0.0580	**0.6903** ± **0.0580**
k-NN (*k*=3)	84.85 ± 3.57	85.70 ± 3.96	84.36 ± 3.80	**86.29** ± **4.26**	0.7643 ± 0.0717	0.7810 ± 0.0794	0.7590 ± 0.0771	**0.7877** ± **0.0846**
k-NN (*k*=5)	84.55 ± 3.60	84.64 ± 3.63	83.32 ± 4.29	**85.99** ± **4.46**	0.7624 ± 0.0735	0.7686 ± 0.0693	0.7469 ± 0.0823	**0.7853** ± **0.0796**
k-NN (*k*=7)	83.42 ± 4.61	**84.93** ± **4.30**	82.67 ± 4.55	84.15 ± 4.49	0.7445 ± 0.0817	**0.7691** ± **0.0735**	0.7387 ± 0.0842	0.7629 ± 0.0763
Ripper	81.73 ± 4.04	83.76 ± 3.99	82.29 ± 4.83	**84.54** ± **3.69**	0.7435 ± 0.0568	0.7380 ± 0.0637	0.7370 ± 0.0675	**0.7674** ± **0.0568**

BBO for all the performance measures. As our goal is to improve the performance of binarized classifier, this shows that BBO framework improves the handling of imbalanced data for NN. For C4.5, the null hypothesis can again be rejected for all three performance measures for B vs BBO. So, the performance definitely improves while using the BBO framework. The null hypothesis can also be rejected for accuracy and Cohen's Kappa for BB vs BBO. So, BBO-C4.5 performs better for these two measures than BB-C4.5. For *k*-NN (*k*=1), the null hypothesis can be rejected for G-mean and Cohen's Kappa measure for both B vs BO and B vs BBO. So, BBO framework handles the class imbalance problem as expected. For *k*-NN (*k*=3), BBO framework also improves performance for all three measures. The same is also true for *k*-NN (*k*=5). For *k*-NN (*k*=7), this is true for B-*k*-NN (*k*=7). So, the performance

Table 4.6: Summary result for Plan 1 based on G-mean. The best results are shown in bold.

Algorithm	B	BB	BO	BBO
NN	0.7835 ± 0.0552	0.7989 ± 0.0587	0.7862 ± 0.0560	**0.7990** ± **0.0548**
C4.5	0.7410 ± 0.0634	0.7789 ± 0.0591	0.7486 ± 0.0603	**0.7819** ± **0.0592**
k-NN (*k*=1)	0.6829 ± 0.0541	0.6829 ± 0.0541	0.7264 ± 0.0519	**0.7264** ± **0.0519**
k-NN (*k*=3)	0.7661 ± 0.0573	0.7919 ± 0.0534	0.7611 ± 0.0634	**0.7975** ± **0.0607**
k-NN (*k*=5)	0.7610 ± 0.0598	0.7814 ± 0.0531	0.7495 ± 0.0693	**0.7935** ± **0.0635**
k-NN (*k*=7)	0.7496 ± 0.0657	**0.7797** ± **0.0574**	0.7447 ± 0.0725	0.7775 ± 0.6157
Ripper	0.7549 ± 0.6127	0.7659 ± 0.0614	0.7614 ± 0.0635	**0.7906** ± **0.0580**

can be improved if binarization with boosting is used. In case of Ripper, BBO framework also improves performance for all three measures.

As discussed in Chapter 4.4 we also compare our framework with other popular binarized algorithms. The results were collected from [3]. The results are presented in Table 4.10. The performance measures shown are accuracy and Cohen's Kappa. G-mean is not shown as that data was not available in [3]. The first half of the table shows the supervised classifiers that were used in the BBO framework. The authors of [3] present the binarization results for some classifiers with different aggregation strategies. The second half of the table shows the result for those classifier with only the best aggregation strategy.

Table 4.7: Wilcoxon signed-rank test result for Plan 1 based on accuracy. Here “R” and “A” means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	A	A	R	R
C4.5	A	A	R	A
k-NN (*k*=1)	A	A	A	A
k-NN (*k*=3)	A	A	R	R
k-NN (*k*=5)	A	A	R	R
k-NN (*k*=7)	R	A	A	A
Ripper	R	A	R	R

Table 4.8: Wilcoxon signed-rank test result for Plan 1 based on Cohen’s Kappa. Here “R” and “A” means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	A	A
C4.5	R	A	R	R
k-NN (*k*=1)	A	R	R	R
k-NN (*k*=3)	A	A	R	R
k-NN (*k*=5)	A	A	R	R
k-NN (*k*=7)	R	A	A	A
Ripper	A	A	R	R

Table 4.10: Performance Comparison with Other Binarization Algorithms.

Algorithm	Accuracy (%)	Cohen’s Kappa
BBO-NN	87.41	0.7765
BBO-C4.5	85.03	0.7720
BBO-*k*-NN (*k*=1)	81.53	0.6903
BBO-*k*-NN (*k*=3)	86.29	0.7877
BBO-*k*-NN (*k*=5)	85.99	0.7853
BBO-*k*-NN (*k*=7)	84.15	0.7629
BBO-Ripper	84.54	0.7674

Table 4.9: Wilcoxon signed-rank test result for Plan 1 based on G-mean. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	R	A
C4.5	R	R	R	A
k-NN (k=1)	A	R	R	R
k-NN (k=3)	R	A	R	R
k-NN (k=5)	A	A	R	R
k-NN (k=7)	R	A	R	A
Ripper	A	A	R	R

As it can be seen from the table, BBO-NN has the highest accuracy among all the classifiers. But for Cohen's Kappa some other classifiers perform better. The accuracy and Cohen's Kappa of BBO-k-NN (k=3) is better than all classifiers found in [3]. So, it performs better than all those classifiers for both measures. Other classifiers using the BBO framework also perform very well. So, it can be said that the classifiers using BBO framework perform comparatively well compared to other binarization algorithms.

4.5.2 Results for Plan 2

Here we present the results for Plan 2. Table 4.11-4.20 show a summary of the performance measures for Plan 2 for different percentages of labeled data. The best result for each algorithm, for each measure is shown in bold. We denote the multiclass version of the algorithm, binarzied version of the algorithm, binarized with boosting version of the algorithm, binarized with oversampling version of the algorithm and BBO framework version of the algorithm with M, B, BB, BO and BBO respectively.

Table 4.11 and 4.12 show the comparison of performance for 10% labeled data. The multi-class version of the algorithms perform the worst for all algorithms. BBO framework performs

Table 4.11: Summary result for Plan 2 based on accuracy and Cohen's Kappa for 10% labeled data. The best results are shown in bold.

Algorithm	Accuracy (%)					Cohen's Kappa				
	M	B	BB	BO	BBO	M	B	BB	BO	BBO
NN	50.80 ± 6.77	61.42 ± 6.43	70.19 ± 4.74	62.14 ± 6.54	**70.54 ± 5.08**	0.1743 ± 0.0881	0.4278 ± 0.0840	0.5163 ± 0.0782	0.4384 ± 0.0796	**0.5316 ± 0.0993**
C4.5	43.33 ± 6.03	57.21 ± 5.86	68.45 ± 5.01	57.75 ± 6.93	**68.56 ± 4.95**	0.1078 ± 0.0792	0.3708 ± 0.0889	0.4938 ± 0.0916	0.3696 ± 0.0930	**0.4946 ± 0.0906**
k-NN (k=1)	38.88 ± 4.18	50.17 ± 5.41	65.26 ± 4.24	50.39 ± 4.30	**65.68 ± 4.72**	0.0907 ± 0.0384	0.2722 ± 0.0635	0.4426 ± 0.0729	0.2743 ± 0.0527	**0.4485 ± 0.0776**
k-NN (k=3)	46.74 ± 4.13	58.44 ± 5.80	**69.25 ± 4.57**	58.65 ± 5.96	69.14 ± 4.65	0.1459 ± 0.0626	0.3828 ± 0.0734	**0.5009 ± 0.0799**	0.3808 ± 0.0755	0.5004 ± 0.0803
k-NN (k=5)	49.54 ± 4.70	61.35 ± 6.30	**69.87 ± 4.81**	61.64 ± 6.15	69.79 ± 4.74	0.1685 ± 0.0734	0.4231 ± 0.0790	**0.5107 ± 0.0815**	0.4230 ± 0.0792	0.5102 ± 0.0831
k-NN (k=7)	50.49 ± 5.46	63.33 ± 5.98	70.31 ± 4.96	62.91 ± 6.14	**70.47 ± 4.87**	0.1734 ± 0.0807	0.4462 ± 0.0846	0.5188 ± 0.0884	0.4397 ± 0.0790	**0.5227 ± 0.0868**
Ripper	43.01 ± 6.48	61.30 ± 8.48	70.04 ± 5.20	61.95 ± 8.16	**70.31 ± 5.13**	0.0690 ± 0.0808	0.4257 ± 0.1122	0.5187 ± 0.1060	0.4332 ± 0.1119	**0.5246 ± 0.1019**

the best in most cases except k-NN (k=3) and k-NN (k=5). For these two algorithms, binarization with boosting and binarization performs the best respectively. The performance of binarization with boosting is quite similar to that of the BBO framework. Binarization with oversampling improves the results over binarization only in most cases.

Table 4.13 and 4.14 show the comparison of performance for 20% labeled data. The multiclass version of the algorithms again perform the worst for all algorithms. It is the worst by a long distance. BBO framework performs the best in most cases. For some lazy algorithms, i.e., k-NN (k=1), k-NN (k=5) and k-NN (k=7) binarization with boosting performs better. The performance of binarization with boosting is quite similar to that of the BBO framework. Binarization with oversampling improves the results over binarization only in most cases. But

Table 4.12: Summary result for Plan 2 based on G-mean for 10% labeled data. The best results are shown in bold.

Algorithm	M	B	BB	BO	BBO
NN	0.2077 ± 0.0759	0.3891 ± 0.0675	0.4217 ± 0.0744	0.3918 ± 0.0641	**0.4240** ± **0.0875**
C4.5	0.1917 ± 0.0638	0.3459 ± 0.0639	0.3954 ± 0.0758	0.3506 ± 0.0607	**0.3965** ± **0.0765**
k-NN (k=1)	0.2291 ± 0.0413	0.3479 ± 0.0570	0.3912 ± 0.0647	0.3476 ± 0.0565	**0.3928** ± **0.0679**
k-NN (k=3)	0.2223 ± 0.0563	0.3842 ± 0.0677	**0.4146** ± **0.0805**	0.3813 ± 0.0732	0.4134 ± 0.0796
k-NN (k=5)	0.2237 ± 0.0576	0.3940 ± 0.0735	**0.4194** ± **0.0774**	0.3947 ± 0.0714	0.4178 ± 0.0781
k-NN (k=7)	0.2213 ± 0.0612	0.4055 ± 0.0722	0.4203 ± 0.0767	0.4018 ± 0.0731	**0.4230** ± **0.0789**
Ripper	0.1271 ± 0.0500	0.3644 ± 0.0709	0.4001 ± 0.0819	0.3671 ± 0.0739	**0.4091** ± **0.0744**

the improvement is not very significant. The performance of all the classifiers significantly improve over that of 10% labeled data.

Table 4.15 and 4.16 show the comparison of performance for 30% labeled data. As before the multiclass version of the algorithms perform significantly worse than the other versions. The best results are obtained by either using the BBO framework or binarization with boosting. The performance of binarization with boosting is quite similar to that of the BBO framework. Binarization with boosting actually performs slightly better than BBO framework for the lazy classifiers. Binarization with oversampling performs similarly to that of binarization only. The performance of all the classifiers significantly improve over that of 20% labeled data.

Table 4.17 and 4.18 show the comparison of performance for 40% labeled data. The multiclass version of the algorithms still perform the worst in most cases. But the difference in

Table 4.13: Summary result for Plan 2 based on accuracy and Cohen's Kappa for 20% labeled data. The best results are shown in bold.

Algorithm	Accuracy (%)					Cohen's Kappa				
	M	B	BB	BO	BBO	M	B	BB	BO	BBO
NN	62.84 ± 5.15	69.02 ± 5.20	**75.98** ± **3.49**	69.17 ± 4.45	75.65 ± 3.98	0.3497 ± 0.1032	0.5332 ± 0.0752	**0.6105** ± **0.0608**	0.5353 ± 0.0644	0.6058 ± 0.0684
C4.5	52.51 ± 5.31	64.15 ± 5.88	74.01 ± 4.04	64.44 ± 5.66	**74.12** ± **3.93**	0.2354 ± 0.0885	0.4625 ± 0.0889	**0.5784** ± **0.0773**	0.4697 ± 0.0843	0.5776 ± 0.0681
k-NN (k=1)	44.02 ± 3.95	55.20 ± 3.95	**69.73** ± **3.69**	55.68 ± 4.56	69.50 ± 3.43	0.1626 ± 0.0491	0.3395 ± 0.0514	**0.5079** ± **0.0608**	0.3446 ± 0.0572	0.5047 ± 0.0587
k-NN (k=3)	57.24 ± 5.25	67.25 ± 4.68	**75.32** ± **3.47**	67.33 ± 5.36	75.18 ± 3.25	0.3063 ± 0.0880	0.5034 ± 0.0676	**0.5946** ± **0.0588**	0.5036 ± 0.0859	0.5897 ± 0.0624
k-NN (k=5)	61.28 ± 4.91	71.15 ± 4.59	**76.76** ± **3.33**	71.38 ± 4.84	76.53 ± 3.29	0.3462 ± 0.0897	0.5575 ± 0.0688	**0.6160** ± **0.0620**	0.5585 ± 0.0816	0.6129 ± 0.0623
k-NN (k=7)	63.48 ± 5.50	72.72 ± 4.55	**77.13** ± **3.05**	72.91 ± 4.71	76.99 ± 3.32	0.3692 ± 0.0918	0.5775 ± 0.0734	**0.6233** ± **0.0580**	0.5797 ± 0.0738	0.6209 ± 0.0611
Ripper	50.67 ± 6.86	69.86 ± 6.55	75.34 ± 4.39	68.33 ± 7.33	**75.59** ± **4.43**	0.1837 ± 0.1162	0.5376 ± 0.1116	0.6001 ± 0.0896	0.5219 ± 0.1131	**0.6044** ± **0.0896**

performance is not as significant as before. The best results for accuracy and Cohen's Kappa are again obtained by either using the BBO framework or binarization with boosting. But for G-mean the best performance comes from either binarization with oversampling or BBO framework. The performance of binarization with boosting is quite similar to that of the BBO framework. Binarization with oversampling performs similarly to that of binarization only. The performance of all the classifiers improve over that of 30% labeled data for all performance measures.

Table 4.19 and 4.20 shows the comparison of performance for 50% labeled data. They are similar to that of 40% labeled data. The multiclass version of the algorithms again perform the worst but not by much. Similarly, the best results are obtained by either using the BBO

Table 4.14: Summary result for Plan 2 based on G-mean for 20% labeled data. The best results are shown in bold.

Algorithm	M	B	BB	BO	BBO
NN	0.3507 ± 0.0663	0.5010 ± 0.0699	0.5224 ± 0.0728	0.5089 ± 0.0691	**0.5392** ± **0.0775**
C4.5	0.3009 ± 0.0815	0.4391 ± 0.0775	0.4931 ± 0.0778	0.4498 ± 0.0761	**0.4992** ± **0.0748**
k-NN (k=1)	0.2992 ± 0.0514	0.4238 ± 0.0593	**0.4729** ± **0.0613**	0.4239 ± 0.0593	0.4720 ± 0.0647
k-NN (k=3)	0.3517 ± 0.0713	0.5032 ± 0.0664	0.5276 ± 0.0697	0.4997 ± 0.0646	**0.5315** ± **0.0656**
k-NN (k=5)	0.3679 ± 0.0650	0.5319 ± 0.0666	**0.5432** ± **0.0697**	0.5296 ± 0.0718	0.5409 ± 0.0728
k-NN (k=7)	0.3794 ± 0.0756	0.5360 ± 0.0661	**0.5453** ± **0.0664**	0.5370 ± 0.0664	0.5450 ± 0.0685
Ripper	0.2215 ± 0.0702	0.4815 ± 0.0760	0.5036 ± 0.0806	0.4661 ± 0.0756	**0.5057** ± **0.0917**

framework or binarization with boosting. Their performance remains almost similar also. But BBO framework performs the best for G-mean in most cases. Binarization with oversampling performs similarly to that of binarization only. .

Now we give our statistical analysis for Plan 2. We provide our Wilcoxon's signed-rank test results. We choose α value 0.1 for our analysis. Table 4.21-4.35 shows if the null hypothesis can be rejected for this value for each classifier. "R" means that the null hypothesis can be rejected in favor of the second algorithm. "A" indicates that the null hypothesis cannot be rejected. Again as our target is to improve the performance of the binarized classifier, we compare between binarezed vs binarization with boosting (B vs BB), binarized vs binarization with oversampling (B vs BO), and binarized vs BBO framework (B vs BBO). Since the overall performance of binarization with boosting is quite good as seen from the given results we also

Table 4.15: Summary result for Plan 2 based on accuracy and Cohen's Kappa for 30% labeled data. The best results are shown in bold.

Algorithm	Accuracy (%)					Cohen's Kappa				
	M	B	BB	BO	BBO	M	B	BB	BO	BBO
NN	71.09 ± 4.19	73.88 ± 5.15	**79.31** ± **3.33**	73.74 ± 4.65	79.18 ± 3.18	0.5152 ± 0.0774	0.6055 ± 0.0798	**0.6673** ± **0.0638**	0.6049 ± 0.0684	0.6663 ± 0.0727
C4.5	64.00 ± 4.20	70.51 ± 4.77	76.93 ± 3.41	70.74 ± 4.37	**76.99** ± **3.43**	0.4379 ± 0.0759	0.5605 ± 0.0745	0.6275 ± 0.0617	0.5624 ± 0.0687	**0.6345** ± **0.0598**
k-NN (*k*=1)	49.59 ± 3.18	59.09 ± 4.57	**72.66** ± **3.55**	58.87 ± 3.30	72.29 ± 2.86	0.2407 ± 0.0476	0.3912 ± 0.0655	**0.5546** ± **0.0553**	0.3897 ± 0.0462	0.5492 ± 0.0521
k-NN (*k*=3)	67.72 ± 4.61	73.32 ± 4.20	**79.22** ± **3.13**	73.22 ± 4.39	79.10 ± 3.23	0.4756 ± 0.0737	0.5937 ± 0.0632	**0.6612** ± **0.0563**	0.5920 ± 0.0627	0.6599 ± 0.0540
k-NN (*k*=5)	72.63 ± 3.97	76.44 ± 3.82	**80.48** ± **3.25**	76.37 ± 4.45	80.34 ± 3.22	0.5383 ± 0.0668	0.6381 ± 0.0590	**0.6820** ± **0.0575**	0.6388 ± 0.0658	0.6798 ± 0.0571
k-NN (*k*=7)	74.37 ± 4.33	77.58 ± 3.73	**80.75** ± **2.90**	77.26 ± 3.96	80.46 ± 2.85	0.5518 ± 0.0736	0.6552 ± 0.0571	**0.6867** ± **0.0485**	0.6521 ± 0.0599	0.6828 ± 0.0484
Ripper	66.17 ± 5.86	73.52 ± 5.50	**78.46** ± **3.56**	72.91 ± 5.43	78.25 ± 3.66	0.4431 ± 0.0962	0.5982 ± 0.0954	0.6511 ± 0.0795	0.5925 ± 0.0851	**0.6523** ± **0.0744**

compare between binarization with boosting vs BBO framework (BB vs BBO).

Table 4.21-4.23 give the statistical results for 10% labeled data. In case of NN, the null hypothesis can be rejected for B vs BBO for all three measures. This shows that BBO framework improves the handling of imbalanced data for NN. The same is also true for BB-NN. BBO-NN performs better than BB-NN for both accuracy and Cohen's Kappa. For C4.5, the null hypothesis can be rejected for all three performance measures for B vs BBO and B vs BB. So, the performance definitely improves while using the BBO framework. But the hypothesis is not rejected for all three measures for BB vs BBO. For *k*-NN (*k*=1), the null hypothesis can be rejected for all three measures for both B vs BB and B vs BBO. So, again it can be said that BBO framework handles the class imbalance problem as expected. For *k*-NN (*k*=3), BBO

Table 4.16: Summary result for Plan 2 based on G-mean for 30% labeled data. The best results are shown in bold.

Algorithm	M	B	BB	BO	BBO
NN	0.4991 ± 0.0769	0.5785 ± 0.0801	0.5933 ± 0.0817	0.5811 ± 0.0707	**0.6211** ± **0.0839**
C4.5	0.4569 ± 0.0758	0.5250 ± 0.0707	**0.5525** ± **0.0761**	0.5279 ± 0.0736	0.5520 ± 0.0738
k-NN (*k*=1)	0.3791 ± 0.0495	0.4761 ± 0.0643	**0.5355** ± **0.0633**	0.4726 ± 0.0475	0.5278 ± 0.0640
k-NN (*k*=3)	0.4869 ± 0.0587	0.5858 ± 0.0729	0.5993 ± 0.0687	0.5824 ± 0.0726	**0.6182** ± **0.0712**
k-NN (*k*=5)	0.5211 ± 0.0581	0.6089 ± 0.0712	**0.6143** ± **0.0718**	0.6073 ± 0.0759	0.6138 ± 0.0716
k-NN (*k*=7)	0.5272 ± 0.0567	0.6131 ± 0.0729	**0.6154** ± **0.0693**	0.6139 ± 0.0719	0.6122 ± 0.0691
Ripper	0.4302 ± 0.0841	0.5467 ± 0.0862	**0.5689** ± **0.0805**	0.5407 ± 0.0818	0.5665 ± 0.0741

framework also improves performance for three measures except G-mean. For the remaining three algorithms the null hypothesis can be rejected for both B vs BB and B vs BBO. So, the BBO framework does improve performance.

Table 4.24-4.26 give the statistical results for 20% labeled data. The null hypothesis can be rejected for algorithms for all performance measures in case of B vs BB. In case of B vs BBO the null hypothesis cannot be rejected for G-mean for *k*-NN (*k*=7). This shows that class imbalance does arise and it can be handled pretty well using the BBO framework. The results show that binarization with boosting and BBO framework performs almost the same. The same is also true for binarization with oversampling and binarizaiton only.

Table 4.27-4.29 give the statistical results for 30% labeled data. In case of NN, the null hypothesis can be rejected for B vs BBO for all three measures. This shows that using BBO

Table 4.17: Summary result for Plan 2 based on accuracy and Cohen's Kappa for 40% labeled data. The best results are shown in bold.

Algorithm	Accuracy (%)					Cohen's Kappa				
	M	B	BB	BO	BBO	M	B	BB	BO	BBO
NN	77.37	77.71	**81.08**	77.80	80.96	0.6314	0.6648	0.6970	0.6666	**0.6986**
	± 3.45	± 3.33	± **3.19**	± 3.83	± 3.60	± 0.0614	± 0.0483	± 0.0503	± 0.0548	± **0.0684**
C4.5	73.21	74.66	**79.04**	74.66	**79.04**	0.5920	0.6217	0.6700	0.6209	**0.6728**
	± 4.01	± 4.37	± **3.37**	± 3.75	± **3.38**	± 0.0671	± 0.0661	± 0.0555	± 0.0577	± **0.0538**
k-NN (*k*=1)	54.71	62.70	73.89	62.57	**74.05**	0.3098	0.4389	0.5750	0.4355	**0.5776**
	± 3.59	± 4.04	± 3.81	± 3.63	± **4.17**	± 0.0560	± 0.0594	± 0.0586	± 0.0520	± **0.0693**
k-NN (*k*=3)	76.31	77.86	**81.87**	78.02	81.78	0.6118	0.6589	**0.7038**	0.6613	0.7026
	± 4.27	± 3.22	± **2.79**	± 3.19	± 2.72	± 0.0743	± 0.0530	± **0.0465**	± 0.0482	± 0.0459
k-NN (*k*=5)	79.65	79.54	**82.49**	79.62	82.23	0.6563	0.6865	**0.7168**	0.6851	0.7119
	± 2.77	± 3.28	± **2.89**	± 2.64	± 2.59	± 0.0490	± 0.0462	± **0.0457**	± 0.0401	± 0.0422
k-NN (*k*=7)	80.21	79.77	82.11	79.70	**82.28**	0.6608	0.6890	0.7116	0.6888	**0.7144**
	± 3.03	± 3.03	± 2.47	± 3.35	± **3.16**	± 0.0603	± 0.0442	± 0.0425	± 0.0461	± **0.0407**
Ripper	74.21	76.30	79.96	76.17	**80.56**	0.5763	0.6404	0.6860	0.6346	**0.6939**
	± 5.18	± 4.53	± 3.96	± 4.02	± **3.87**	± 0.0876	± 0.0757	± 0.0753	± 0.0733	± **0.0674**

framework improves the handling of imbalanced data for NN. The same is also true for C4.5. So, the performance definitely improves while using the binarization with boosting and over-sampling. For *k*-NN (*k*=1), the null hypothesis can be rejected for all three measures for B vs BB and B vs BBO. So, BBO framework definitely improves the performance. For *k*-NN (*k*=3), BBO framework also improves performance for all three measures. But for *k*-NN (*k*=5) the hypothesis is not rejected for G-mean. The performance is also bad for *k*-NN (*k*=7). In case of Ripper, BBO framework also improves performance for all three measures.

Table 4.30-4.32 give the statistical results for 40% labeled data. In case of NN, the null hypothesis can be rejected for B vs BBO for all the measures. This shows that using BBO framework improves the performance for NN. For C4.5, the null hypothesis can be rejected for

Table 4.18: Summary result for Plan 2 based on G-mean for 40% labeled data. The best results are shown in bold.

Algorithm	M	B	BB	BO	BBO
NN	0.6107 ± 0.0690	0.6543 ± 0.0690	0.6564 ± 0.0652	0.6591 ± 0.0647	**0.6777** ± **0.0736**
C4.5	0.5876 ± 0.0797	0.5981 ± 0.0783	0.6089 ± 0.0742	0.5967 ± 0.0682	**0.6107** ± **0.0729**
k-NN (k=1)	0.4420 ± 0.0495	0.5265 ± 0.0595	0.5743 ± 0.0641	0.5294 ± 0.0605	**0.5795** ± **0.0717**
k-NN (k=3)	0.6070 ± 0.0702	0.6465 ± 0.0653	0.6627 ± 0.0640	0.6592 ± 0.0662	**0.6711** ± **0.0618**
k-NN (k=5)	0.6276 ± 0.0618	0.6724 ± 0.0634	0.6712 ± 0.0643	**0.6746** ± **0.0646**	0.6693 ± 0.0653
k-NN (k=7)	0.6246 ± 0.0675	0.6706 ± 0.0626	0.6653 ± 0.0619	**0.6718** ± **0.0655**	0.6677 ± 0.0634
Ripper	0.5511 ± 0.0818	0.6085 ± 0.0737	0.6231 ± 0.0696	0.6055 ± 0.0728	**0.6296** ± **0.0752**

accuracy and Cohen's Kappa performance measures for B vs BBO. For k-NN (k=1), the null hypothesis can be rejected for all three measures for B vs BBO. So, BBO framework definitely improves the performance. For k-NN (k=3), BBO framework also improves performance for all three measures. But for k-NN (k=5) the hypothesis is not rejected for both G-mean and Cohen's Kappa. The performance is also bad for k-NN (k=7). In case of Ripper, BBO framework also improves performance for all three measures.

Table 4.33-4.35 give the statistical results for 50% labeled data. In case of NN, the null hypothesis can be rejected for B vs BBO for only accuracy. This shows that using BBO framework improves only accuracy for NN. For C4.5, the null hypothesis can be rejected for all three performance measures for B vs BBO. So, the performance definitely improves using the BBO framework. For k-NN (k=1), the null hypothesis can be rejected for all three measures for

Table 4.19: Summary result for Plan 2 based on accuracy and Cohen's Kappa for 50% labeled data. The best results are shown in bold.

Algorithm	Accuracy (%)					Cohen's Kappa				
	M	B	BB	BO	BBO	M	B	BB	BO	BBO
NN	77.14 ± 3.79	78.37 ± 3.49	**81.21** ± **3.61**	78.09 ± 3.60	81.10 ± 3.29	0.6275 ± 0.0704	0.6717 ± 0.0755	**0.7022** ± **0.0668**	0.6695 ± 0.0606	0.7004 ± 0.0701
C4.5	72.81 ± 4.07	74.72 ± 3.95	78.98 ± 4.02	74.14 ± 4.10	**79.70** ± **3.63**	0.5820 ± 0.0683	0.6219 ± 0.0626	0.6706 ± 0.0712	0.6120 ± 0.0635	**0.6777** ± **0.0669**
k-NN (k=1)	54.46 ± 3.82	62.42 ± 4.06	73.46 ± 3.69	62.82 ± 3.78	**73.89** ± **3.88**	0.3028 ± 0.0566	0.4415 ± 0.0567	0.5726 ± 0.0611	0.4452 ± 0.0529	**0.5789** ± **0.0697**
k-NN (k=3)	76.06 ± 3.99	78.32 ± 3.50	**81.75** ± **3.43**	78.08 ± 3.24	81.64 ± 3.35	0.6095 ± 0.0655	0.6679 ± 0.0542	**0.7057** ± **0.0584**	0.6660 ± 0.0528	0.7026 ± 0.0564
k-NN (k=5)	79.32 ± 3.28	79.58 ± 3.44	82.33 ± 2.84	79.57 ± 3.60	**82.39** ± **2.94**	0.6517 ± 0.0639	0.6879 ± 0.0516	0.7153 ± 0.0492	0.6874 ± 0.0508	**0.7167** ± **0.0488**
k-NN (k=7)	79.79 ± 3.92	79.76 ± 2.84	**82.43** ± **2.67**	79.98 ± 3.16	82.30 ± 2.81	0.6558 ± 0.0713	0.6918 ± 0.0435	**0.7180** ± **0.0473**	0.6932 ± 0.0492	0.7164 ± 0.0475
Ripper	74.03 ± 4.57	76.60 ± 4.20	80.25 ± 3.95	76.95 ± 3.64	**80.61** ± **3.77**	0.5716 ± 0.0884	0.6476 ± 0.0714	0.6913 ± 0.0773	0.6524 ± 0.0591	**0.6947** ± **0.0743**

B vs BBO. So, BBO framework definitely improves the performance. For k-NN (k=3), BBO framework also improves performance for all three measures. But for k-NN (k=5) and k-NN (k=7) the performance is bad. In case of Ripper, BBO framework also improves performance for all three measures.

An overall analysis shows that BBO framework improves the overall performance of the classifier for NN, C4.5, k-NN (k=1), k-NN (k=3) and Ripper. This is true for k-NN (k=5) and k-NN (k=7) only when the number of labeled data is low. The BB version of the algorithms also improves the performance of the classifier in almost all the cases. But in most cases it cannot be differentiated from the BBO framework.

Table 4.20: Summary result for Plan 2 based on G-mean for 50% labeled data. The best results are shown in bold.

Algorithm	M	B	BB	BO	BBO
NN	0.6014 ± 0.0677	0.6565 ± 0.0789	0.6545 ± 0.0778	0.6534 ± 0.0717	**0.6674** ± **0.0895**
C4.5	0.5738 ± 0.0675	0.5875 ± 0.0716	0.6047 ± 0.0844	0.5793 ± 0.0674	**0.6132** ± **0.0803**
k-NN (k=1)	0.4329 ± 0.0569	0.5176 ± 0.0558	0.5645 ± 0.0660	0.5243 ± 0.0579	**0.5683** ± **0.0680**
k-NN (k=3)	0.5955 ± 0.0652	0.6423 ± 0.0717	0.6574 ± 0.0721	0.6547 ± 0.0668	**0.6629** ± **0.0726**
k-NN (k=5)	0.6121 ± 0.0637	0.6633 ± 0.0675	**0.6643** ± **0.0683**	0.6613 ± 0.0656	0.6638 ± 0.0711
k-NN (k=7)	0.6130 ± 0.0670	0.6661 ± 0.0638	0.6635 ± 0.0701	0.6637 ± 0.0656	**0.6639** ± **0.0702**
Ripper	0.5455 ± 0.0870	0.5952 ± 0.0723	0.6173 ± 0.0802	0.6020 ± 0.0747	**0.6254** ± **0.0779**

4.5.3 Results for Plan 3

In this subsection we look at the experimental results gained for Plan 3. The average best result for each performance measure is shown in bold. M represents the multiclass version of the LLGC. BBO is the LLGC that uses the BBO framework. B, BB and BO represent binarized LLGC, binarization with boosting LLGC and binarization with oversampling LLGC respectively.

Table 4.36 and 4.37 show the result comparisons for Plan 3. It can be seen from the results that the BBO framework gives the best overall results for G-mean in all cases. Its result is second best to binarization with boosting only for Cohen's Kappa for 10% labeled data. So, it can be said that the BBO framework is the most effective approach to handle the class

Table 4.21: Wilcoxon signed-rank test result for Plan 2 based on accuracy for 10% Labeled Data. Here “R” and “A” means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	R	R	R
C4.5	R	A	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	A
k-NN (k=5)	R	A	R	A
k-NN (k=7)	R	A	R	A
Ripper	R	A	R	A

Table 4.22: Wilcoxon signed-rank test result for Plan 2 based on Cohen’s Kappa for 10% Labeled Data. Here “R” and “A” means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	R	R	R
C4.5	R	A	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	A
k-NN (k=5)	R	A	R	A
k-NN (k=7)	R	A	R	A
Ripper	R	A	R	A

imbalance problem for LLGC. The worst performance is acquired for the multiclass version. So, binarization gains improved results over its multiclass counterpart. Binarization with boosting also improves the result of binarization only. Binarization with oversampling also improves the performance of binarization but not as much as binarization with boosting.

Now we present our statistical analysis for Plan 3. We provide our Wilcoxon’s signed-rank

Table 4.23: Wilcoxon signed-rank test result for Plan 2 based on G-mean for 10% Labeled Data. Here “R” and “A” means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	R	A
C4.5	R	A	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	A	A
k-NN (k=5)	R	A	R	A
k-NN (k=7)	R	A	R	A
Ripper	R	A	R	R

Table 4.24: Wilcoxon signed-rank test result for Plan 2 based on accuracy for 20% Labeled Data. Here “R” and “A” means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	R	A
C4.5	R	A	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	A
k-NN (k=5)	R	A	R	A
k-NN (k=7)	R	A	R	A
Ripper	R	A	R	R

test results. We choose α value 0.1 for our analysis. Table 4.38-4.40 show if the null hypothesis can be rejected for this value for each classifier. “R” means that the null hypothesis can be rejected. “A” means that the null hypothesis cannot be rejected in favor of the second algorithm. As before, since our target is to improve the performance of the binarized classifier, we compare between binarezed vs binarization with boosting (B vs BB), binarized vs binarization

Table 4.25: Wilcoxon signed-rank test result for Plan 2 based on Cohen's Kappa for 20% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	R	A
C4.5	R	R	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	A
k-NN (k=5)	R	A	R	A
k-NN (k=7)	R	A	R	A
Ripper	R	A	R	A

Table 4.26: Wilcoxon signed-rank test result for Plan 2 based on G-mean for 20% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	R	R	R
C4.5	R	R	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	A
k-NN (k=5)	R	A	R	A
k-NN (k=7)	R	A	A	A
Ripper	R	A	R	A

with oversampling (B vs BO), and binarized vs BBO framework (B vs BBO). Since the overall performance of binarization with boosting is quite good as seen from the given results we also compare between binarization with boosting vs BBO framework (BB vs BBO).As it can be seen from Table 4.38 and 4.40,

As it can be seen from the tables BB-LLGC and BBO-LLGC perform better than B-LLGC

Table 4.27: Wilcoxon signed-rank test result for Plan 2 based on accuracy for 30% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	R	A
C4.5	R	A	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	A
k-NN (k=5)	R	A	R	A
k-NN (k=7)	R	A	A	A
Ripper	R	A	R	A

Table 4.28: Wilcoxon signed-rank test result for Plan 2 based on Cohen's Kappa for 30% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	R	A
C4.5	R	A	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	A
k-NN (k=5)	R	A	R	A
k-NN (k=7)	A	A	A	A
Ripper	R	A	R	A

in all cases for all measures. This is not true for the other comparisons.

Table 4.29: Wilcoxon signed-rank test result for Plan 2 based on G-mean for 30% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	R	R
C4.5	R	A	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	R
k-NN (k=5)	A	A	A	A
k-NN (k=7)	A	A	A	A
Ripper	R	A	R	A

Table 4.30: Wilcoxon signed-rank test result for Plan 2 based on accuracy for 40% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	R	A
C4.5	R	A	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	A
k-NN (k=5)	R	A	R	R
k-NN (k=7)	A	A	A	R
Ripper	R	A	R	A

4.6 Discussion

We now discuss our findings from the previous section. For Plan 1, the experimental results showed that the BBO framework improved the results over the simple binarized version of the algorithms. Since the value of G-mean value of the performance measures increased for

Table 4.31: Wilcoxon signed-rank test result for Plan 2 based on Cohen's Kappa for 40% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	A	A	R	A
C4.5	R	A	R	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	A	R	A
k-NN (k=5)	R	A	A	A
k-NN (k=7)	A	A	A	R
Ripper	R	A	R	A

Table 4.32: Wilcoxon signed-rank test result for Plan 2 based on G-mean for 40% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	A	R	R	R
C4.5	A	A	A	A
k-NN (k=1)	R	A	R	A
k-NN (k=3)	R	R	R	A
k-NN (k=5)	A	A	A	A
k-NN (k=7)	A	A	A	R
Ripper	R	A	R	A

all algorithms while using the BBO framework, it can be safely said that binarization does give rise to the class imbalance problem. Using a combination of boosting and oversampling is an effective way to tackle this problem. Also, the standard deviation was not very high for the performance measures in all cases. In case of k-NN (k=1), the performance was same for binarization and binarization with boosting, and that of binarization with oversampling

Table 4.33: Wilcoxon signed-rank test result for Plan 2 based on accuracy for 50% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	R	A	R	A
C4.5	R	A	R	R
k-NN (*k*=1)	R	R	R	R
k-NN (*k*=3)	R	A	R	A
k-NN (*k*=5)	R	A	R	A
k-NN (*k*=7)	A	A	A	A
Ripper	R	A	R	R

Table 4.34: Wilcoxon signed-rank test result for Plan 2 based on Cohen's Kappa for 50% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	A	A	A	A
C4.5	R	A	R	R
k-NN (*k*=1)	R	A	R	R
k-NN (*k*=3)	R	A	R	A
k-NN (*k*=5)	A	A	A	A
k-NN (*k*=7)	A	A	A	A
Ripper	R	A	R	A

and BBO framework. *k*-NN (*k*=1) only uses one nearest neighbor to make its prediction. So, boosting is incapable of making any changes to the results. But oversampling does improve the result. This is why the performances in these two cases are the same.

Another interesting point that can be seen from the results is that the results for the BB version of the algorithms are also quite close to that of the BBO framework. Oversampling

Table 4.35: Wilcoxon signed-rank test result for Plan 2 based on G-mean for 50% Labeled Data. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Algorithm	B vs BB	B vs BO	B vs BBO	BB vs BBO
NN	A	A	A	R
C4.5	R	A	R	R
k-NN (k=1)	R	R	R	A
k-NN (k=3)	R	R	R	A
k-NN (k=5)	A	A	A	A
k-NN (k=7)	A	A	A	A
Ripper	R	A	R	R

Table 4.36: Summary result for Plan 3 based on accuracy and Cohen's Kappa. The best results are shown in bold.

Labeled Data	Accuracy (%)					Cohen's Kappa				
	M	B	BB	BO	BBO	M	B	BB	BO	BBO
10%	39.71 ± 7.45	40.96 ± 7.07	**58.61** ± **5.59**	43.63 ± 7.19	**58.61** ± **5.59**	0.2919 ± 0.0969	0.3282 ± 0.0866	**0.5173** ± **0.0845**	0.3547 ± 0.0876	0.5169 ± 0.1092
20%	40.74 ± 5.67	40.45 ± 5.55	60.42 ± 3.70	42.62 ± 4.90	**60.94** ± **4.37**	0.2943 ± 0.1136	0.3124 ± 0.0785	0.5158 ± 0.0633	0.3346 ± 0.0709	**0.5215** ± **0.0752**
30%	40.30 ± 4.60	41.17 ± 5.62	62.00 ± 4.00	43.77 ± 5.12	**62.99** ± **3.50**	0.2933 ± 0.0851	0.3102 ± 0.0835	0.5352 ± 0.0705	0.3398 ± 0.0753	**0.5476** ± **0.0800**
40%	41.45 ± 3.80	42.05 ± 4.26	60.17 ± 3.72	43.11 ± 4.21	**62.25** ± **3.96**	0.2992 ± 0.0675	0.3171 ± 0.0574	0.5201 ± 0.0557	0.3245 ± 0.0603	**0.5388** ± **0.0752**
50%	40.89 ± 4.17	42.25 ± 4.34	60.82 ± 4.10	44.69 ± 3.96	**63.88** ± **3.62**	0.2802 ± 0.0774	0.3189 ± 0.0809	0.5247 ± 0.0712	0.3431 ± 0.0666	**0.5541** ± **0.0771**

and undersampling has been the technique of choice used by researchers to handle the class imbalance problems. But these results indicate that boosting is also an effective technique, if

Table 4.37: Summary result for Plan 3 based on G-mean. The best results are shown in bold.

Labeled Data	M	B	BB	BO	BBO
10%	0.3007 ± 0.0834	0.3308 ± 0.0738	0.4739 ± 0.0811	0.3576 ± 0.0706	**0.4753** ± **0.0963**
20%	0.2838 ± 0.0729	0.3166 ± 0.0732	0.4628 ± 0.0768	0.3392 ± 0.0760	**0.4702** ± **0.0852**
30%	0.3029 ± 0.0846	0.3166 ± 0.0867	0.4660 ± 0.0866	0.3465 ± 0.0777	**0.4755** ± **0.0922**
40%	0.3310 ± 0.0759	0.3430 ± 0.0733	0.4776 ± 0.0702	0.3509 ± 0.0712	**0.4902** ± **0.0810**
50%	0.3166 ± 0.0745	0.3480 ± 0.0826	0.4866 ± 0.0815	0.3731 ± 0.0789	**0.5173** ± **0.0985**

Table 4.38: Wilcoxon signed-rank test result for Plan 3 based on accuracy. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively in favor of the second algorithm.

Labeled Data	B vs BB	B vs BO	B vs BBO	BB vs BBO
10%	R	A	R	A
20%	R	A	R	A
30%	R	A	R	A
40%	R	A	R	A
50%	R	A	R	A

not better than oversampling, to handle the class imbalance problem. So, the decision to give more importance to boosting in the BBO framework has been vindicated.

As mentioned earlier, for Plan 2 we modified the supervised classifiers to act as semi-supervised classifiers. Figure 4.1-4.3 and Figure 4.4[a-c] also show these results in graph form. The x-axes shows the percentage of labeled data and the y-axes shows the value of the performance measure. The first thing that can be noted from these experimental results of Plan

Table 4.39: Wilcoxon signed-rank test result for Plan 3 based on Cohen's Kappa. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively.

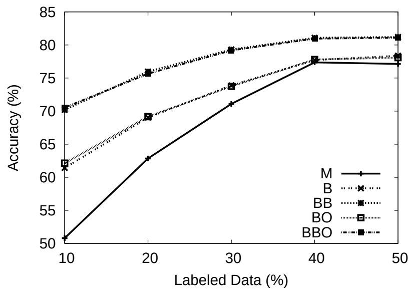
Labeled Data	B vs BB	B vs BO	B vs BBO	BB vs BBO
10%	R	A	R	A
20%	R	A	R	A
30%	R	A	R	A
40%	R	A	R	A
50%	R	A	R	A

Table 4.40: Wilcoxon signed-rank test result for Plan 3 based on G-mean. Here "R" and "A" means that the null hypotheisis can be rejected and not rejected respectively.

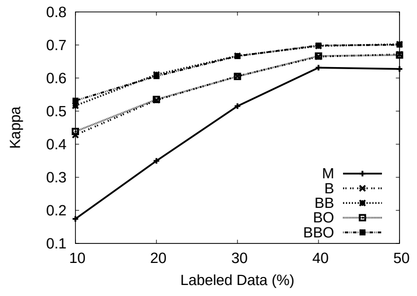
Labeled Data	B vs BB	B vs BO	B vs BBO	BB vs BBO
10%	R	A	R	A
20%	R	A	R	A
30%	R	A	R	A
40%	R	A	R	A
50%	R	R	R	A

2 is that the multiclass version of these classifiers perform the worst by a long distance. So, it can be said that using binarization can be an effective technique in case of semi-supervised classification also. At the same time we see that the performance of the BBO framework is still significantly better than that of the binarized algorithm. This again indicates that binarization does cause data to become imbalanced and BBO framework can help to improve the results. Moreover, the standard deviation was not high in any case. Unlike Plan 1, the results for the various classifiers are different for k-NN (k=1). This can be attributed to the way the supervised k-NN (k=1) is converted to a semi-supervised algorithm. As discussed in Chapter 4.4 the unlabeled data are labeled randomly for Plan 2. This randomness is what causes the results to be different.

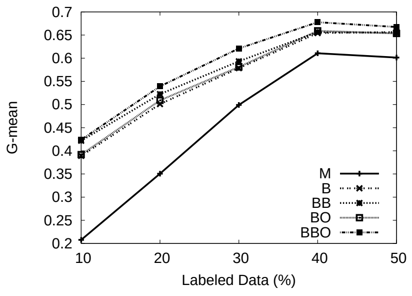
Again like Plan 1, it can be seen that binarization with boosting performs really well and



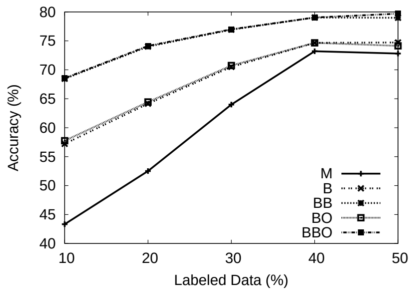
(a) Accuracy:NN



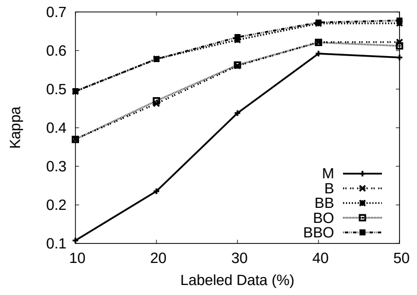
(b) Cohen's Kappa:NN



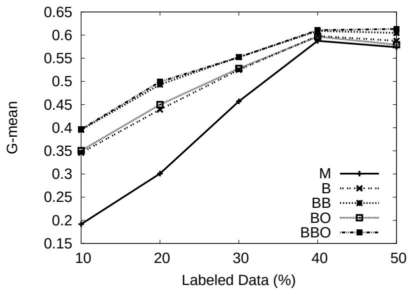
(c) G-mean:NN



(d) Accuracy for C4.5

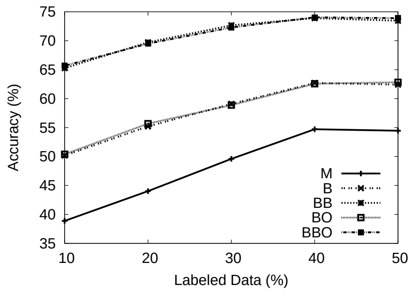


(e) Cohen's Kappa:C4.5

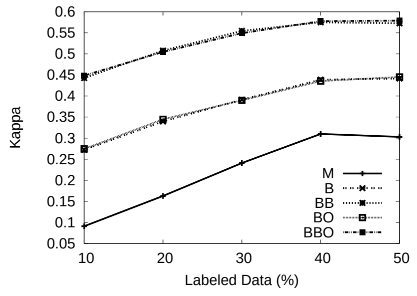


(f) G-mean:C4.5

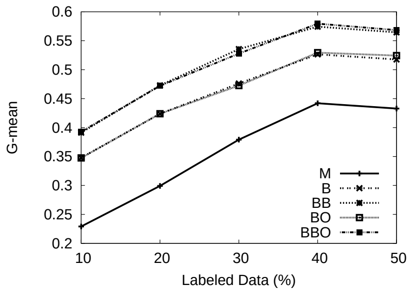
Figure 4.1: Performance Measures for NN and C4.5



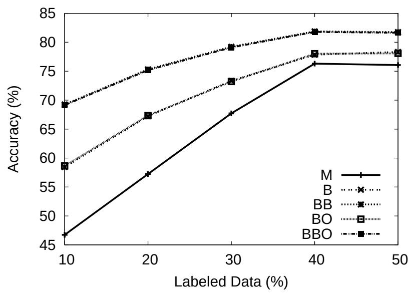
(a) Accuracy:k-NN (k=1)



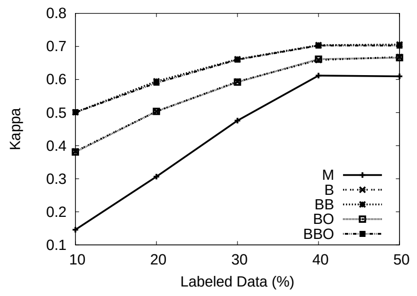
(b) Cohen's Kappa:k-NN (k=1)



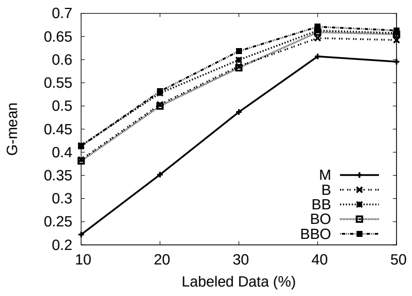
(c) G-mean:k-NN (k=1)



(d) Accuracy:k-NN (k=3)

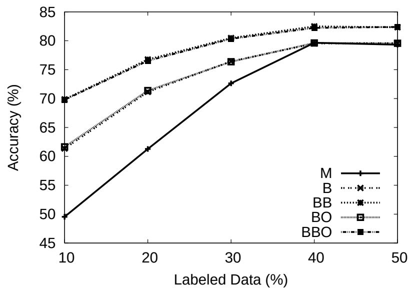


(e) Cohen's Kappa:k-NN (k=3)

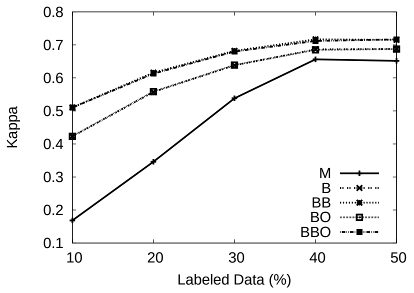


(f) G-mean:k-NN (k=3)

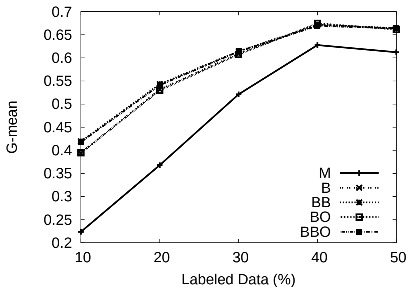
Figure 4.2: Performance Measures for k-NN (k=1) and k-NN (k=3)



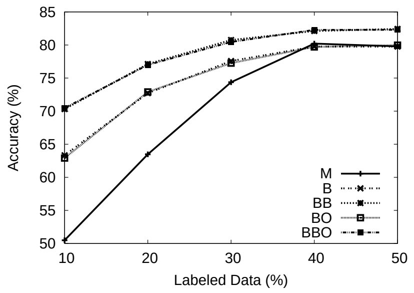
(a) Accuracy:k-NN (k=5)



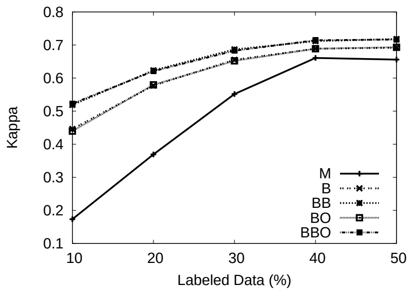
(b) Cohen's Kappa:k-NN (k=5)



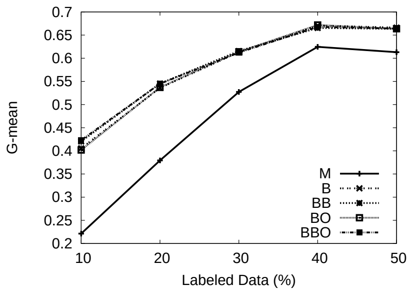
(c) G-mean:k-NN (k=5)



(d) Accuracy:k-NN (k=7)



(e) Cohen's Kappa:k-NN (k=7)

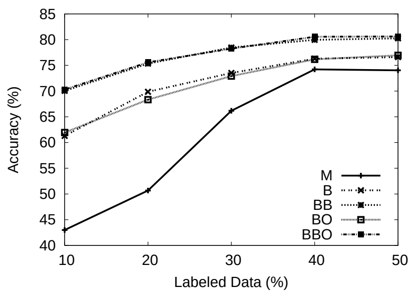


(f) G-mean:k-NN (k=7)

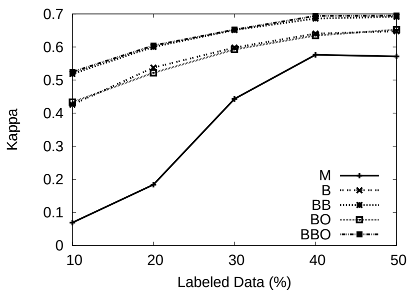
Figure 4.3: Performance Measures for k-NN (k=5) and k-NN (k=7)

even sometimes beats the performance of the BBO framework. This happens specially for the lazy algorithm k-NN. So, boosting can be said to be a more effective tool than oversampling for semi-supervised classification to handle the imbalance problem. Moreover, we see that the increase in performance of the BBO framework over binarized algorithms decreases with the increase of labeled data. In fact, the performance when the number of labeled data is 40% and 50% is almost the same. It proves that the BBO framework is more effective when the number of labeled data is low. This is also an advantage of using the BBO framework because usually the number of unlabeled data used in semi-supervised classification is significantly higher than of labeled data. So, in such situations the BBO framework will produce significantly better results. Finally, it was seen from the experimental results that the BBO framework repeatedly performed worse than the binarized algorithms for a few specific data sets like autos, shuttle, lymphography etc. For other data sets the performance was better for the BBO framework. So, the underlying properties of data sets may also contribute to the performance of the BBO framework.

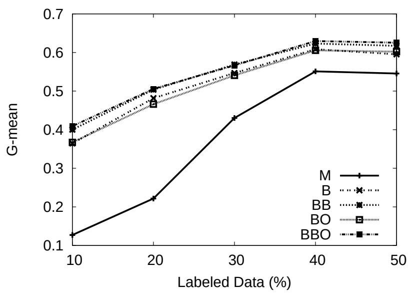
For Plan 3, we used only the LLGC algorithm. The results are shown in graph form in



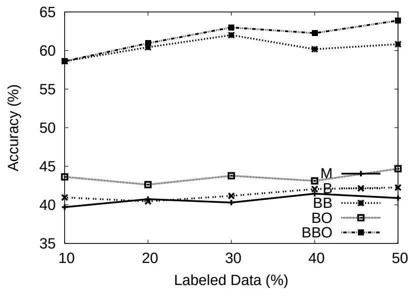
(a) Accuracy:Ripper



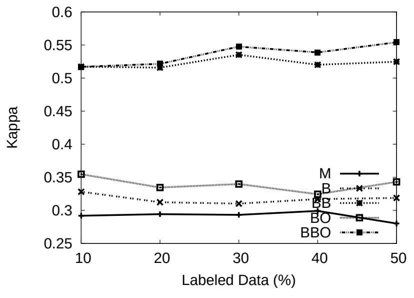
(b) Cohen's Kappa:Ripper



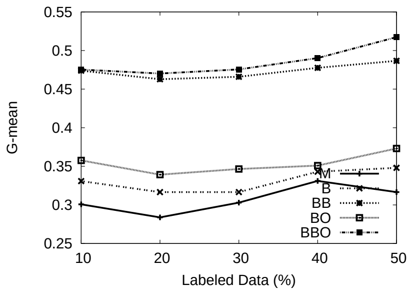
(c) G-mean:Ripper:LLGC



(d) Accuracy:LLGC



(e) Cohen's Kappa:LLGC



(f) G-mean:LLGC

Figure 4.4: Performance Measures for Ripper and LLGC

Figure 4.4[d-f]. Like Plan 2, the performance of the multiclass version of LLGC performed the worst. BBO framework also performed better than the binarized version of LLGC. So, we can come to the conclusion that binarization can be an effective approach to semi-supervised classification and BBO framework can help improve the results even more. Again the standard deviation was not very high. Unlike Plan 2 though, the performance of all the classifiers remained almost the same for different fractions of labeled data. This is a property of the LLGC framework. Its performance varies very little with labeled data as can be seen from [22]. Boosting also performs similarly to the BBO framework. Furthermore, like Plan 2 the BBO framework performs worse than the binarized version of LLGC for the some fixed data sets irrespective of the percentage of labeled data. This again suggests that the underlying properties of the data set may impact the performance of the BBO framework.

Chapter 5

Conclusion

5.1 Conclusion

In this thesis, we proposed a framework called BBO based on the OVA binarization approach to solve the multiclass classification problem. Binarization is a popular approach to solve the multiclass problem. However, it gives rise to the class imbalance problem which results in reduced performance. The salient feature of the BBO framework is that it can handle this class imbalance problem. To achieve this goal BBO framework uses a combination of boosting and oversampling.

Most other research works that deal with the class imbalance problem, usually use oversampling or undersampling as their technique of choice. But unlike those approaches BBO framework gives more focus on boosting. This is done by introducing boosting a lot earlier than oversampling in the BBO framework. Furthermore, the approach to use binarization has been implemented in the domain of supervised classification thus far. We here use the BBO framework with various semi-supervised classifiers.

We performed a multitude of experiments to evaluate the performance of the BBO framework. Various popular supervised classifiers like NN, C4.5, k-NN and Ripper were used with the BBO framework. Experiments were also performed with semi-supervised versions of the same algorithms as well as LLGC. The first important thing that could be noted from the acquired experimental results is that binarization does give rise to class imbalance problem. It can also be seen that boosting is an effective tool, more effective than oversampling, to solve the class imbalance problem. But usually a combination of boosting and oversampling gives the best results. Moreover, in case of semi-supervised classification the performance of the binarized version of classifiers is quite impressive compared to that of the simple multiclass version of the classifiers. So, BBO framework can also be a desired approach in case of solving multiclass semi-supervised classification problem.

5.2 Future Work

The BBO framework is applied using a combination of boosting and oversampling. Boosting is not algorithmically constrained. It can be applied in a multitude of ways. For future research, we would like to apply other variations of boosting in the BBO framework. Similarly, there are various popular oversampling algorithms that can also be used with the BBO framework. As mentioned earlier, undersampling is a popular approach to handle the class imbalance problem. But it is not incorporated in the BBO framework. In future, we would also like to find out if boosting can be combined with undersampling.

In this thesis, we have introduced binarization in the domain of semi-supervised classification. We would like to try out other semi-supervised classification with the BBO framework. Moreover, more thought can be given about how binarization can be introduced to the domain of unsupervised classification.

Bibliography

[1] Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

[2] Johannes Fürnkranz. Round robin ensembles. *Intelligent Data Analysis*, 7(5):385–403, 2003.

[3] Mikel Galar, Alberto Fernández, Edurne Barrenechea Tartas, Humberto Bustince Sola, and Francisco Herrera. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition*, 44(8):1761–1776, 2011.

[4] Jin-Hyuk Hong, Jun-Ki Min, Ung-Keun Cho, and Sung-Bae Cho. Fingerprint classification using one-vs-all support vector machines dynamically ordered with naive bayes classifiers. *Pattern Recognition*, 41(2):662–671, 2008.

[5] Rangachari Anand, Kishan G. Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka. An improved algorithm for neural network classification of imbalanced training sets. *IEEE Transactions on Neural Networks*, 4(6):962–969, 1993.

[6] J Friedman. Another approach to polychotomous classification. Technical report, Technical report, Stanford University, Department of Statistics, 1996.

[7] Trevor Hastie and Robert Tibshirani. Classification by pairwise coupling. In *NIPS*, 1997.

[8] John C. Platt, Nello Cristianini, and John Shawe-Taylor. Large margin dags for multiclass classification. In *NIPS*, pages 547–553, 1999.

[9] Jens C. Huhn and Eyke Hüllermeier. Fr3: A fuzzy rule learner for inducing reliable classifiers. *IEEE T. Fuzzy Systems*, 17(1):138–149, 2009.

[10] Alberto Fernández, Marıa Calderón, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. Enhancing fuzzy rule based systems in multi-classification using pairwise coupling with preference relations. *EUROFUSE*, 9:39–46, 2009.

[11] Bo Liu, Zhifeng Hao, and Xiaowei Yang. Nesting algorithm for multi-classification problems. *Soft Comput.*, 11(4):383–389, 2007.

[12] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.

[13] Heng Zhang, Da-Han Wang, and Cheng-Lin Liu. Keyword spotting from online chinese handwritten documents using one-vs-all trained character classifier. In *ICFHR*, pages 271–276, 2010.

[14] Benoît Delachaux, Julien Rebetez, Andrés Pérez-Uribe, and Héctor Fabio Satizábal Mejia. Indoor activity recognition by combining one-vs.-all neural network classifiers exploiting wearable and depth sensors. In *IWANN (2)*, pages 216–223, 2013.

[15] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[16] Zhi-Hua Zhou and Ming Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Trans. Knowl. Data Eng.*, 17(11):1529–1541, 2005.

[17] Yanjuan Li and Maozu Guo. A new relational tri-training system with adaptive data editing for inductive logic programming. *Knowl.-Based Syst.*, 35:173–185, 2012.

[18] Xiaojin Zhu. Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison*, 2:3, 2006.

[19] Thorsten Joachims. Transductive learning via spectral graph partitioning. In *ICML*, pages 290–297, 2003.

[20] Roberto Basili. Learning to classify text using support vector machines: Methods, theory, and algorithms by thorsten joachims. *Computational Linguistics*, 29(4):655–661, 2003.

[21] Sounak Chakraborty. Bayesian semi-supervised learning with support vector machine. *Statistical Methodology*, 8(1):68–82, 2011.

[22] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *NIPS*, 2003.

[23] Deming Zhai, Hong Chang, Shiguang Shan, Xilin Chen, and Wen Gao. Multiview metric learning with global consistency and local smoothness. *ACM TIST*, 3(3):53, 2012.

[24] Rodrigo G. F. Soares, Huanhuan Chen, and Xin Yao. Semisupervised classification with cluster regularization. *IEEE Trans. Neural Netw. Learning Syst.*, 23(11):1779–1792, 2012.

[25] Pavan Kumar Mallapragada, Rong Jin, Anil K. Jain, and Yi Liu. Semiboost: Boosting for semi-supervised learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(11):2000–2014, 2009.

[26] Hamed Valizadegan, Rong Jin, and Anil K. Jain. Semi-supervised boosting for multi-class classification. In *ECML/PKDD (2)*, pages 522–537, 2008.

[27] Ke Chen and Shihai Wang. Semi-supervised learning via regularized boosting working on multiple semi-supervised assumptions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):129–143, 2011.

[28] Vladimir Cherkassky. The nature of statistical learning theory. *IEEE Transactions on Neural Networks*, 8(6):1564, 1997.

[29] Peng Xu, Yangyang Shi, and Martha A. Larson. Tud at mediaeval 2012 genre tagging task: Multi-modality video categorization with one-vs-all classifiers. In *MediaEval*, 2012.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[31] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[32] Geoffrey McLachlan. *Discriminant analysis and statistical pattern recognition*, volume 544. Wiley. com, 2004.

[33] D. Randall Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *J. Artif. Intell. Res. (JAIR)*, 6:1–34, 1997.

[34] William W. Cohen. Fast effective rule induction. In *ICML*, pages 115–123, 1995.

[35] Johannes Fürnkranz and Gerhard Widmer. Incremental reduced error pruning. In *ICML*, pages 70–77, 1994.

[36] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, 1990.

[37] Yoav Freund. An adaptive version of the boost by majority algorithm. *Machine Learning*, 43(3):293–318, 2001.

[38] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res. (JAIR)*, 16:321–357, 2002.

[39] Sukarna Barua, Md. Monirul Islam, Xin Yao, and Kazuyuki Murase. Mwmote-majority weighted minority oversampling technique for imbalanced data set learning. *IEEE Trans. Knowl. Data Eng.*, 26(2):405–425, 2014.

[40] Arthur Asuncion and David J Newman. Uci machine learning repository, 2007.

[41] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1(6):80–83, 1945.

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

[43] Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, and Salvador García. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. *Multiple-Valued Logic and Soft Computing*, 17(2-3):255–287, 2011.