Université d'Ottawa · University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

Hongyu GUO

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

M. Sc. (Systems Science)

GRADE - DEGREE

Systems Science

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

Multiple classifier combination through ensembles and data generation

H. Viktor

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

CO-DIRECTEUR DE LA THÈSE - THESIS CO-SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

N. Japkowicz

M. Turcotte

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

MULTIPLE CLASSIFIER COMBINATION THROUGH ENSEMBLES AND DATA GENERATION

A THESIS

SUBMITTED IN ACCORDANCE WITH THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

AT

THE UNIVERSITY OF OTTAWA

BY

HONG YU GUO

JULY 26, 2004

SUPERVISED BY PROF HERNA L VIKTOR

Your file Votre référence
ISBN: 0-494-01483-0
Our file Notre référence
ISBN: 0-494-01483-0

DECLARATION

I THE UNDERSIGNED HEREBY DECLARE THAT THE WORK CONTAINED IN THIS THESIS IS MY OWN ORIGINAL WORK AND HAS NOT PREVIOUSLY IN ITS ENTIRETY OR IN PART BEEN SUBMITTED AT ANY UNIVERSITY FOR A DEGREE

HONG YU GUO

26 JULY 2004

ABSTRACT

An ensemble of classifiers consists of a set of individually trained classifiers whose predictions are combined when classifying new instances. The resulting ensemble is generally more accurate than the individual classifiers it consists of. In particular, one of the most popular ensemble methods, the Boosting approach, improves the predictive performance of weak classifiers, which can achieve only an error rate of at least slightly better than random guessing. This improvement is achieved by focusing on examples that are hard to be correctly classified, i.e. the so-called hard examples. The Boosting approach has been applied to many real world domains, including medicine, bioinformatics, fraud detection and text classification, amongst others, with great success.

However, in many domains, Boosting algorithms frequently suffer from over-emphasizing these hard examples. That is, the Boosting algorithm puts too much effort on the hard examples, thus leading to poor training and test set accuracies. Moreover, the knowledge acquired from such hard examples may be insufficient to improve the overall accuracy of the ensemble. Also, when applied to two-class domains with imbalanced class frequencies, where the number of examples of one (majority) class is much higher than the other (minority), a traditional Boosting algorithm tends to produce a high predictive accuracy over the majority class, but poor accuracy over the minority class. This is due to a number of factors. Firstly, the learning algorithms tend to ignore small classes while concentrating on classifying the large ones accurately (i.e. the so-called learning bias). Secondly, the weight updating mechanism of the Boosting

algorithms may deteriorate the learning bias. Thirdly, examples from the minority class may provide insufficient knowledge for learning. In addition, the few examples from the minority class are much easier to be over-emphasized by the Boosting algorithms.

This thesis introduces new techniques to address the above-mentioned two problems. Two approaches, namely the DataBoost and DataBoost-IM algorithms, are provided to extend Boosting algorithms' predictive performance.

The DataBoost algorithm is designed to assist Boosting algorithms to avoid over-emphasizing hard examples. In the DataBoost algorithm, new synthetic data with bias information towards hard examples are added to the original training set when training the component classifiers. The DataBoost approach was evaluated against ten data sets, using both decision trees and neural networks as base classifiers. The experiments show promising results, in terms of overall accuracy when compared to a standard benchmarking Boosting algorithm.

The DataBoost-IM algorithm is developed to learn from two-class imbalanced data sets. In the DataBoost-IM approach, the class frequencies and the total weights against different classes within the ensemble's training set are rebalanced by adding new synthetic data. The DataBoost-IM method was evaluated, in terms of the *F-measures*, *G-mean* and *overall accuracy*, against seventeen highly and moderately imbalanced data sets using decision trees as base classifiers. The experimental results show that the DataBoost-IM method compares well in comparison with a base classifier, a standard benchmarking Boosting algorithm and three advanced Boosting-based algorithms for imbalanced data sets. Results indicate that the DataBoost-IM approach does not sacrifice one class in favor of the other, but produces high predictions against both minority and majority classes.

三人行,必有我师焉.
I can always be certain of learning from those I am with.
-Confucius (551-479 B.C. China)

ACKNOWLEDGEMENTS

I am indebted to a very long list of people who supported me during this journey. A big thank you goes to:

- Professor HERNA L VIKTOR who sparked my interest in this research and was a great supervisor.
- My family, especially my PARENTS for instilling in me the passion to learn and my wife YAYUAN TAN for her long-lasting understanding and support during the entire project.
- My friends and colleagues for their support, love and patience.

SUMMARY OF CONTENTS

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMOUS

Accuracy: accuracy expresses the percentage of testing examples correctly recognized by the system. It is defined as *(TP+TN) / (TP + FP +TN + FN)*. (*For a definition of these terms, see Confusion Matrix listed below.*)

Boosting: an ensemble method whose focus is to produce a series of dependent component classifiers. The aim is for each component classifier to be better able to predict examples for which the current ensemble's performance is poor.

Bagging: an ensemble method where each individual classifier is generated with a different random sampling of the same original training set.

Confusion matrix: the confusion matrix, as shown in following table, represents the typical metrics for evaluating the performance of machine learning algorithms. In the table, *TN* is the number of *True Negatives, FP* is the number of *False Positives*, *FN* is the number of *False Negatives*, and *TP* is the number of *True Positives.* The *TP Rate* and *FP Rate* are calculated as *TP / (FN+TP)* and *FP / (FP+TN).*

	Predicted Negative	Predicted Positive
Actual Negative	*TN*	*FP*
Actual Positive	*FN*	*TP*

Ensemble: an ensemble consists of a set of individually trained classifiers whose predictions are combined when classifying novel instances.

Error rate: defined as $1 - (TP+TN) / (TP + FP + TN + FN)$. It expresses the percentage of testing examples incorrectly recognized by the system.

F-measure: an evaluation measure incorporated the *recall* and *precision* into a single number. It is defined as $((1+\beta^2) \times Recall \times Precision) / (\beta^2 \times Recall + Precision)$, where β corresponds to the relative importance of the *precision* versus the *recall*

Generalization: construct a general concept from a set of examples.

G-mean: an evaluation technique which is defined as $\sqrt{Positive\_Accuracy \times Negative\_Accuracy}$, the square root of the positive example accuracy rate times the negative example accuracy rate.

Hard examples: examples hard to be correctly classified.

Imbalanced data set: data set where the number of examples of one class is much higher than the others.

Recall: an evaluation measure defined as $TP / (TP + FN)$. It measures the 'completeness' of the classification, i.e. the probability that an instance belonging to a class is classified into this class.

Precision: An evaluation measure defined as $TP / (TP + FP)$. It measures the 'purity' of the classification, i.e. the probability that an instance predicted to be in a class truly belongs to this class.

ROC: (Receiver Operating Characteristic). A technique for summarizing a classifier's performance over a range of possible class frequencies, by considering the

tradeoffs between *TP Rate* and *FP Rate*. The X-axis represents the *FP* rate, calculated by *FP/ (TN+FP)* and the Y-axis represents the *TP* rate, calculated by *TP / (TP + FN)*.

Stacking: an ensemble method which contains 2 levels of generalizations. Firstly, the individual component classifiers are produced. Secondly, a meta level classifier is trained for combining component classifiers to form the final model.

Ten-fold cross validation: An evaluation technique where the data set is first partitioned into ten equal sized sets. Next each set is, in turn, used as the test set while the classifier trains on the other nine sets.

Weak classifier: classifiers which can achieve an error rate of at least slightly better than random guessing.

CHAPTER ONE

INTRODUCTION

CHAPTER ONE

INTRODUCTION

学而不思则罔，思而不学则怠。

To learn without thinking is labor in vain, to think without learning is desolation

-Confucius (551-479 B.C. China)

Machine learning refers to the ability of a machine to improve its performance automatically through experience [2]. The classification problem is one of the major challenges encountered in machine learning studies. In classification learning, a learning scheme takes a set of classified examples, from which it is expected to learn to classify unseen examples. In this learning, the examples are described by pairs [(x, c(x))], where x is a vector of attribute values and c(x) is the corresponding concept label. A typical classification learning scenario assumes the existence of a set of training examples from which the learning scheme produces a classifier whose performance is then assessed on a set of independent, previously unseen examples (test data). The performance is normally judged by the percentage of unseen examples correctly classified by the trained classifier, namely the accuracy rate.

An ensemble of classifiers consists of a set of individually trained classifiers whose predictions are combined when classifying new instances [1]. The resulting ensemble is generally more accurate than any of the individual classifiers making up the

ensemble. Recently, ensemble of classifiers has become an active research area in the machine learning community [1, 3, 4, 5, 6, 60].

One of the most widely used ensemble method is called the Boosting approach. The Boosting method is an ensemble method where the performance of weak classifiers, which can achieve an error rate of at least slightly better than random guessing [9], is improved by focusing on hard examples. Hard examples are those examples which are difficult to be correctly classified by the learning scheme [9]. The Boosting algorithm focuses on the production of a series of dependent classifiers, in which each classifier is better able to predict hard examples for which the previous classifier's performance was poor [1]. The outputs of these classifiers are combined using weighted voting in the final prediction of the model. Recent studies have indicated that, by focusing on improving the classification of hard examples, Boosting algorithms are applicable to a broad spectrum of problems with great success, including medicine, bioinformatics, language recognition and text classification, amongst others [1, 9].

Typically, learning schemes such as Boosting methods are expected to learn from a training data set where the training examples are reasonably balanced. That is, the classes of the concept label c(x) of the training examples are approximately equally represented. However, two-class imbalanced data sets, where the number of examples of one (majority) class is much higher than the other (minority), are encountered in many real world applications such as medical diagnostics, web mining, network intrusion detection, and fraud detection, amongst others [33, 34, 36]. Learning from such imbalanced data sets presents an important challenge to the machine learning community. Traditional machine learning algorithms tend to ignore small classes while concentrating on classifying the large ones accurately (the so-called learning bias), thus producing poor predictive accuracy over the minority class.

1.1 Motivation

Two research problems which have received extensive attention in the machine learning community, are of importance here. Firstly, improving the classification of domains which contain hard to learn examples using Boosting algorithms is an important research topic. Secondly, adapting Boosting algorithms for imbalanced data sets would go a long way to ensure their usefulness in real-world domains.

Firstly, previous research has shown that, when applied to domains with hard to learn examples, the Boosting algorithms frequently suffer from over-emphasizing these hard examples. That is, the Boosting algorithm puts too much effort on the hard examples, thus misleading the ensemble. This leads to poor training and test set accuracies. Over-emphasizing hard examples, which presents one of the most important disadvantages of the Boosting approach, can thus dramatically decrease the predictive performance of Boosting algorithms or even destroy the ensemble [1, 7, 9, 13]. Also, the knowledge acquired from such hard examples, on which the Boosting algorithms need to concentrate in order to achieve good predictive performance, may be insufficient to improve the overall accuracy of the ensemble.

Secondly, an interesting complication arises when applying Boosting algorithms to two-class imbalanced data sets. This is due to the fact that the Boosting approach may worsen the learning algorithms' learning bias towards the majority class. Also, the few examples from the minority class are much easier to be over-emphasized by the Boosting algorithms, due to the algorithms' extreme interest in hard examples. Furthermore, knowledge from the minority class, which contains few examples, may not be sufficient to aid the classifiers' learning process. Moreover, the weight updating mechanism of the traditional Boosting algorithm is in favor of the majority class. This implies that, during training and testing, the predictive accuracy against the minority class of a traditional Boosting ensemble may be poor.

The focus of this thesis is twofold. It is towards extending the Boosting method to be effective, firstly, against hard to learn examples and, secondly, against imbalanced

data sets. The goal of the work presented here is to introduce new approaches to address the two above-mentioned problems. More specifically, two new algorithms will be presented in this thesis. The first DataBoost algorithm is able to extend Boosting algorithms to avoid over-emphasizing hard examples, thus improve the prediction performance of the Boosting algorithms when learning from domains with hard to learning examples. In the DataBoost algorithm, new synthetic data with bias information towards hard examples are added to the original training set to train the component classifier, during each of the iterations of the Boosting algorithm. The second DataBoost-IM algorithm is a method used to assist Boosting algorithms to focus on not only hard examples, but also minority class examples, thus extending Boosting algorithms' predictive performance when learning from imbalanced data sets. That is, by not sacrificing one class in favor of the other, high predictions against both minority and majority classes are produced. In the DataBoost-IM approach, the class frequencies and the total weights against different classes within the ensemble's training set are rebalanced by adding new synthetic data, during all iterations of the Boosting algorithm.

1.2 Thesis Layout

This thesis includes six chapters and is organized as follows. Chapter 2 provides a review of the current literature pertaining to ensemble approaches and includes a detailed discussion of Boosting algorithms. This is followed, in Chapter 3, with an overview of the background related to the imbalanced class frequencies problem and a description of some challenging issues when learning from imbalanced data sets. Chapter 4 presents the DataBoost approach, which integrates synthetic data creation into the Boosting approach, together with an experimental evaluation. Chapter 5 describes the DataBoost-IM algorithm, where the class frequencies and the total weights against different classes within the ensemble's training set are rebalanced by adding new synthetic data. Also, this chapter includes experimental results used to

evaluate the DataBoost-IM approach. Finally, Chapter 6 concludes the thesis and highlights the suggested directions for future research.

The next chapter provides an overview of ensembles of classifiers and includes a detailed discussion of one of the most popular ensemble methods, namely the Boosting approach.

CHAPTER TWO

ENSEMBLES OF CLASSIFIERS

CHAPTER TWO

ENSEMBLES OF CLASSIFIERS

Recall that an ensemble consists of a set of individually trained classifiers whose predictions are combined when classifying novel instances. The resulting ensemble is generally more accurate than any of the individual classifiers making up the ensemble. Ensemble approaches are an active area of research in machine learning and the related field of data mining and knowledge discovery, with a number of research groups investigating techniques for combining the predictions of multiple classifiers. This chapter provides background information and a review of the current literature pertaining to ensemble approaches.

This chapter is organized as follows. Section 2.1 provides a summary of the ensemble method. Section 2.2 includes an overview of popular ensemble methods, namely the Bagging, Boosting, and Stacking approaches. This is followed, in Section 2.3, with a description of one of the most popular ensemble algorithms, namely the Boosting method. Section 2.3 also includes a discussion of one of the most challenging issues confronting the Boosting algorithms. That is, the problem of over-emphasizing hard examples, where Boosting algorithms put too much efforts on trying to correctly classify harder examples.

2.1 Overview

Many researchers have shown that ensembles can greatly improve the generalization accuracy when compared to individual classifiers, and that weak learners can be converted into strong ones by using the ensemble approach [1, 3, 4, 5, 6, 60]. According to Freund and Schapire [7], "an ensemble is a way of combining the performance of many weak classifiers to produce a powerful committee". This provides a shift in mind set for the learning-system designer. That is, instead of trying to design a learning algorithm that is accurate over the entire space, the learning system designer can instead focus on finding weak learning algorithms that only need to be able to achieve an error rate of better than random guessing.

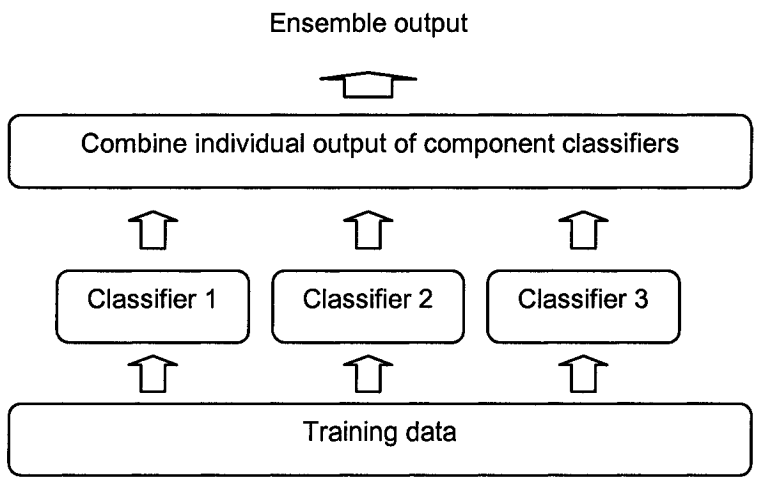


Fig. 1. An ensemble consisting of 3 classifiers

The basic idea of an ensemble of classifiers is as follows. Initially, each classifier in the ensemble learns individually from a set of training instances to obtain a hypothesis. Then, for each novel instance, the predicted output of each of these classifiers is combined to produce the output of the ensemble. Figure 1 illustrates the basic framework for an ensemble consisting of three classifiers. Initially, classifiers 1,

2 and 3 in the ensemble are trained to individually produce a hypothesis about the training data. Then, for each novel instance to be classified, the predicted output of classifiers 1, 2 and 3 are combined to produce the final output of the ensemble.

Ensemble can, more often than not, increase the predictive performance over a single model [1, 3]. In practice, ensembles only make sense if the individual classifiers make errors in different parts of the problem space. Therefore, an ideal ensemble consists of highly correct classifiers that disagree as much as possible [8]. Research also shows that most of the reduction in error for ensemble methods occurs with the first few additional classifiers [1, 9]. That is, there is not really a significant gain when adding more component classifiers. This issue will be further explored in Section 2.3.

The following section provides a review of three popular ensemble methods.

2.2 Popular Ensemble Methods

The most prominent methods for combining models of classifiers are the Bagging, Boosting, and Stacking approaches. These three popular ensemble methods are discussed next.

2.2.1 The Bagging Method

The Bagging method combines classifiers of the same type and exploits the instability inherent in learning algorithms [5]. For instable algorithms, small changes in the training set result in large changes in predictions [1].

Bagging stands for "bootstrap aggregating", which implies that in the Bagging method, each individual classifier in the ensemble is generated with a different random sampling of the same original training set. To this end, the instances in each training set are randomly sampled, with replacement, from the original data set to create a new data set of the same size. The final prediction is based on majority

voting (for classification) or average (for prediction) of the outputs of the classifiers in the ensembles. In Bagging, re-sampling of the training set is not dependent on the performance of the earlier classifiers, and each individual model is built separately [5].

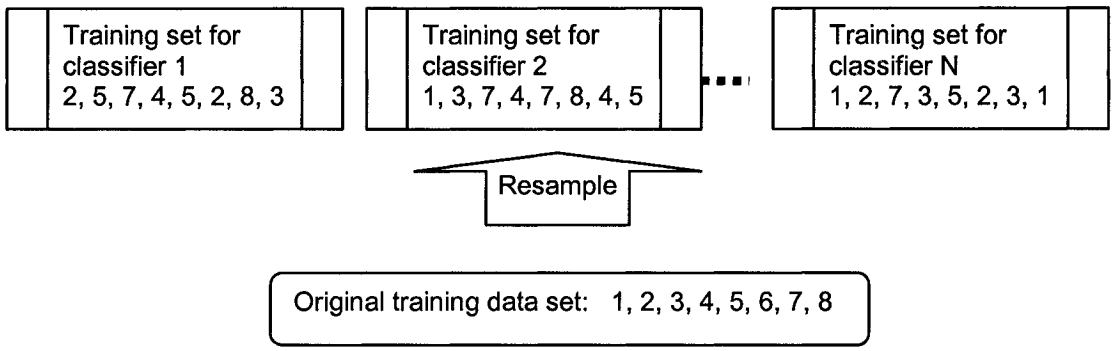


Fig. 2. Bagging re-sampling original data set for each component classifier

Figure 2 gives a sample of applying the Bagging method against an imaginary set of data. Recall that the Bagging method re-samples its training data set. Figure 2 shows that some examples are represented multiple times while others are left out. That is, the training set for classifier 1 might contain examples 2 and 5 twice, but does not contain either example 1 or 6. As a result, each component classifier in the Bagging method is trained with an independent sample of the original data set. In this way, a diverse set of classifiers are generated, which are subsequently combined in a majority voting scheme as the ensemble output.

Research shows that the Bagging method is usually more accurate than a single model built from the original training data [1, 3]. The Bagging method can be applied to many classification learning algorithms. It is resilient to noise and does not easily overfit the training data. On the other hand, the Bagging method doesn't work well for classifiers that are stable, i.e. classifiers whose output is insensitive to small changes in the input [3]. Also, poor classifiers can be transferred into worse ensembles by the Bagging method, because of the majority voting scheme of the

Bagging method. In this case, the majority of the component classifiers have poor predictive performance [3].

In practice, the Bagging method works effectively on unstable learning algorithms where small changes in the training set result in large changes in predictions [10]. The next section introduces the Boosting method, one of the most popular ensemble approaches.

2.2.2 The Boosting Method

The Boosting [9] method, as briefly introduced in Chapter 1, encompasses a family of method, which focus on producing a series of dependent classifiers. The final prediction is based on voting.

The Boosting method combines classifiers of the same type and each new classifier is influenced by the performance of those built previously. Each iteration of the Boosting approach focuses on producing a new classifier. The new classifier is better able to predict examples for which the current ensemble's performance is poor. In other words, examples which were incorrectly predicted by previous classifiers in the series are chosen more often, or have more effect, on the prediction results in the next classifier, than examples that were correctly predicted. This produces a set of "easy" instances, and a set of "hard" ones. During each next iteration, a classifier is built for the re-weighted or re-sampled data, which consequently focuses on classifying the hard instances correctly.

Research has shown that the Boosting method is usually more accurate than the Bagging method [1, 9, 13]. When using Boosting methods, a powerful classifier can be built by combining very simple ones. The Boosting method is suitable for combining classifiers, and usually does not preclude the use of poor predictors as the Bagging method does [3, 13]. However, the Boosting method can only combine

classifiers with the same type such as a set of decision trees. Also, the Boosting methods always concentrate on correctly classifying the misclassified data, which may lead to overfitting noisy data [3, 9, 13]. Therefore, the performance of the Boosting algorithms is very dependents on the characteristics of the data set being examined [1, 17, 18]. The next section introduces a two-level ensemble scheme, named the Stacking method.

2.2.3 The Stacking Method

Stacking stands for Stacked Generalization, which means that there are two levels of generalization [4]. Firstly, the individual component classifiers are produced. Secondly, a meta-level classifier is trained for combining the component classifiers to form the final model.

The basic scheme of the Stacking method architecture, as shown in Figure 3, consists of an ensemble of component classifiers that form the level-0 model, and of a single combining classifier of the meta-level model namely, the level-1 model. Initially, each of the component classifiers is trained to form the level-0 classifiers. Once the level-0 classifiers have been built, they are used to give their predictions of the instances and to form the level-1 or meta-level training data. Then, the combining classifier uses the predictions of the component classifiers to derive the final prediction of the entire composite classifier. In summary, this method thus includes two phases. First of all, the component classifiers are created. Secondly, the combining classifier is trained using a meta-level training set which is composed of the predictions of the component classifiers and the original training set.

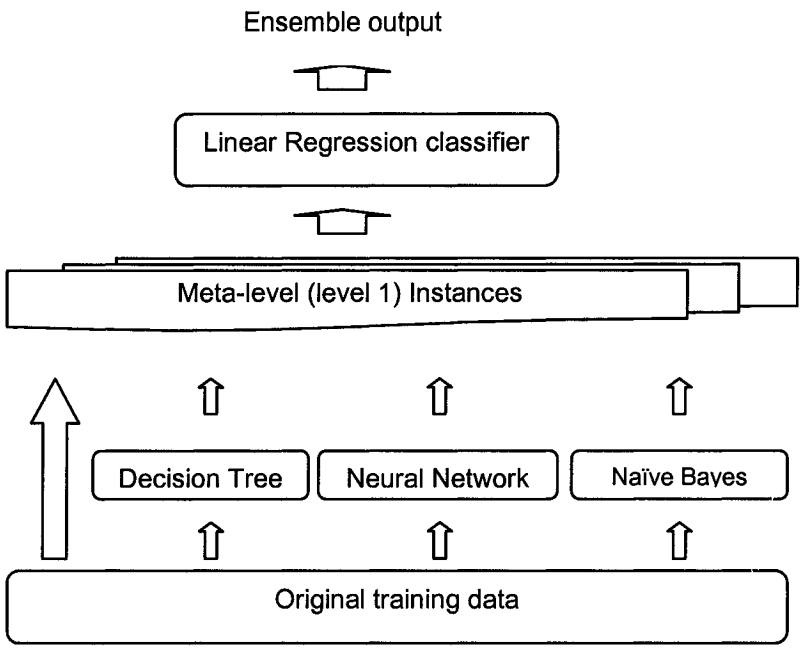


Fig. 3. A Stacking ensemble consisting of Decision Tree, Neural Network, and Naïve Bayes as Level-0 learners and Linear Regression as Level-1 learner

Figure 3 illustrates the basic framework for a Stacking ensemble. In this example, we have three level-0 learners, a Decision Tree (DT), a Naïve Bayes technique (NB), and a Neural Network (NN); and one level-1 learner, namely a Linear Regression algorithm (LR). Firstly, the level-0 learners are trained using the original training data. Secondly, the level-1 learner, LR, algorithm is trained using the meta-level instances. These instances contain the outcome of the three level-0 learners together with some of the original data set, for example the test data set. In other words, the training data set used to train the level-1 learner will contain the original data and the predictions of the level-0 learners. Many methods of how to use the original data set and the predictions from the level-0 classifiers to form the level-1 training data have been investigated [3, 12]. For example, an intuitive approach is to include the predictions of component classifiers as part of the attributes of the new training examples.

Recall that the Stacking method combines base classifiers in a non-linear fashion through training a meta-level classifier (Meta-classifier) rather than voting [4]. In the

Stacking method, the level-1 learner try to learn which classifiers are the reliable ones, using another learning algorithm, to discover how best to combine the output of the base learners. This method includes two training phases, implying that some training instances are needed to hold out to train the meta-level classifier [1, 4].

The Stacking method may combine classifiers with different types and usually has better accuracy than the best level-0 model. However, it is less widely used than the Bagging or Boosting methods. This is due to the fact that it is difficult to analyze theoretically, and there is no generally accepted "best way" of producing a Stacking ensemble [3]. In the Stacking method, it is very difficult to know exactly what data should be used to train the meta-level classifier. The optimal ensemble size is also difficult to be determined. Also, determining which algorithms are suitable for the level-0 and level-1 learners still needs to be further investigated [1, 4].

Research shows that, in practice, a simple algorithm for the level-1 generalizers will perform well [3, 4]. The basic idea of Stacking can be applied in many different variations, such as dynamically selecting the best classifiers based on the characteristics of the new instance. For example, Puuronen et al. developed a variant of the Stacking approach, called the dynamic integration algorithm, for an ensemble of classifiers [12]. This approach dynamically selects the best classifier or combines the component classifiers by taking into account characteristics of a new instance to be classified [12]. Some new variations use the probabilities to form the level-1 data to improve the performance of stacking [3].

The next section provides a comparative summary of the Bagging, Stacking, and Boosting methods.

2.2.4 Comparative Summary

Table 1 shows a summary of the three popular ensemble methods, namely the Bagging, Boosting, and Stacking methods, and highlights their main differences.

Firstly, we focus on the type of learner for which each method is most suitable. As shown in Table 1, the Bagging method needs unstable weak learners. That is, small changes to the training set will cause large changes in the learned classifier. In contrast, the Boosting method requires less unstable algorithms. The Boosting method actively tries to force the weak learning algorithm to change its hypotheses by constructing a "hard" distribution over the examples, based on the performance of previously generated hypotheses [9]. Unlike the Bagging and Boosting methods, the Stacking method is not normally used to combine models of the same type [3].

Secondly, as indicated in Table 1, these three methods differ in the way of combining the component classifiers. In the Bagging and Boosting methods, voting is used to combine outputs of component classifiers. Instead, the Stacking method attempts to learn which component classifiers are the reliable ones, using the meta-level learner.

Table 1. Summary of three popular ensemble methods

	Bagging	Boosting	Stacking
Learners used	▪ unstable weak learner ▪ the same type	▪ the same type	▪ Combine different types of weak learners
The way of combining models	▪ voting	▪ voting	▪ learns which component classifiers are the reliable ones, using another learning algorithm
Predictive performance	▪ Resilient to noisy data	▪ Usually produce greater accuracies than Bagging ▪ Less resilient to noisy data	▪ Very depend on the learners used in the ensembles
In practice	▪ Less widely used than Boosting method	▪ The most widely used ensemble method	▪ Less widely used than Boosting and Bagging methods

Thirdly, in term of predictive performance, the Boosting method usually performs better than the Bagging method [1, 9, 13]. However, it has been found that it is less

resilient with regards to noise in the training data than the Bagging method. In particular, overfitting noisy data is one of the most serious disadvantages in Boosting methods [1, 13]. Such a problem make the Boosting method very depend on the data. The predictive performance of the Stacking method is very depends on the kind of level-0 and level-1 learners. Therefore, it is hard to predict the predictive performance of Stacking methods.

In practice, the Boosting method is the most popular approach [7]. The Stacking method is less widely used than the Bagging and Boosting methods, since it is very difficult to analyze the Stacking ensembles theoretically, as noted earlier [3]. Also, there is no generally accepted "best way" to combine component classifiers in the Stacking ensembles.

The next section provides a detailed discussion of the Boosting approach, which is the focus of this research.

2.3 Discussion of Boosting Approaches

The most widely used Boosting method is the AdaBoost approach [1, 9, 13, 14]. Since it has been introduced in 1995 by Freund and Schapire [18], the AdaBoost algorithm has been widely used. The AdaBoost algorithm is fast, robust, simple and easy to program. It has no parameters to tune (except for the number of iterations), requires no prior knowledge about the weak classifier and may be flexibly combined with other methods suitable for finding weak hypotheses [7]. The following sections will focus on the AdaBoost approach.

2.3.1 The AdaBoost Approach

Recall from Section 2.2.2 that the focus of the Boosting method is to produce a series of dependent classifiers in which a new classifier is influenced by the performance of those built previously. The AdaBoost algorithms use one of two methods, i.e. re-

sampling with weight or re-weighting, to obtain the training data for the next component classifier. In re-sampling with weight, "hard" examples, which are difficult to be classified correctly, will occur more in later training sets. This method is used if the base classifier is not able to handle weighted examples. In the re-weighting mechanism, the original training data set will be re-weighted based on the current classifier's performance. "Hard" example will have higher weight when it occur in the later training set, thus higher weighted examples will affect the classifier's error more than those examples with lower weights.

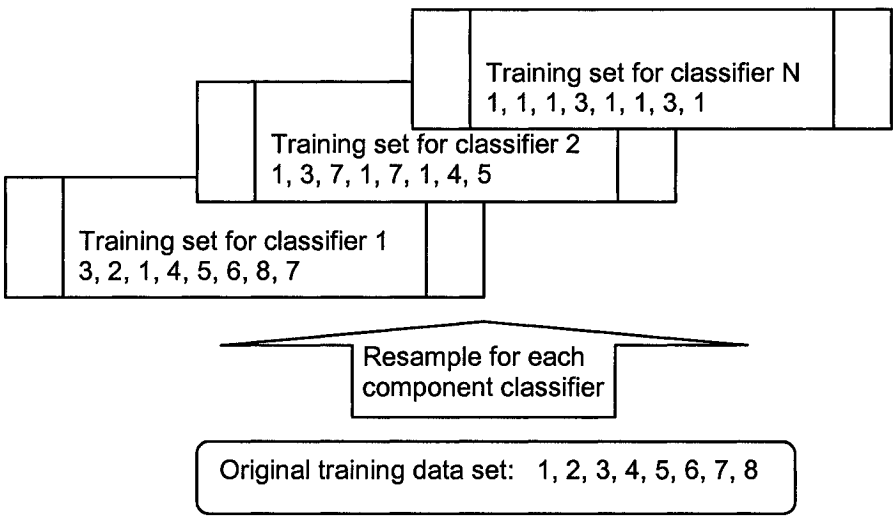


Fig. 4. Boosting with re-sampling

Figure 4 shows how the AdaBoost method obtains training data for each component classifier using re-sampling. We assume there are eight training instances, and example 1 is an "outlier" which is hard to classify correctly. In Figure 4, the AdaBoost method uses the re-sampling technique to deal with the training instances. We may see that the outlier (example 1) occurs more often than other examples in the later iterations of the boosting procedures. This is due to the fact that the AdaBoost method needs to concentrate on correctly classifying the hard examples in order to achieve good predictive performance.

Figure 5 shows how the previous training set, as used in Figure 4, changes in an AdaBoost ensemble which uses re-weighting to handle the training data. The outlier, example 1, will obtain a dominating weight in the later iterations of the boosting procedure. This is indicated in Figure 5, where the high weights are indicated in bold. In this way, the boosting procedure will be able to focus on correctly classifying example 1 due to its dominate weight.

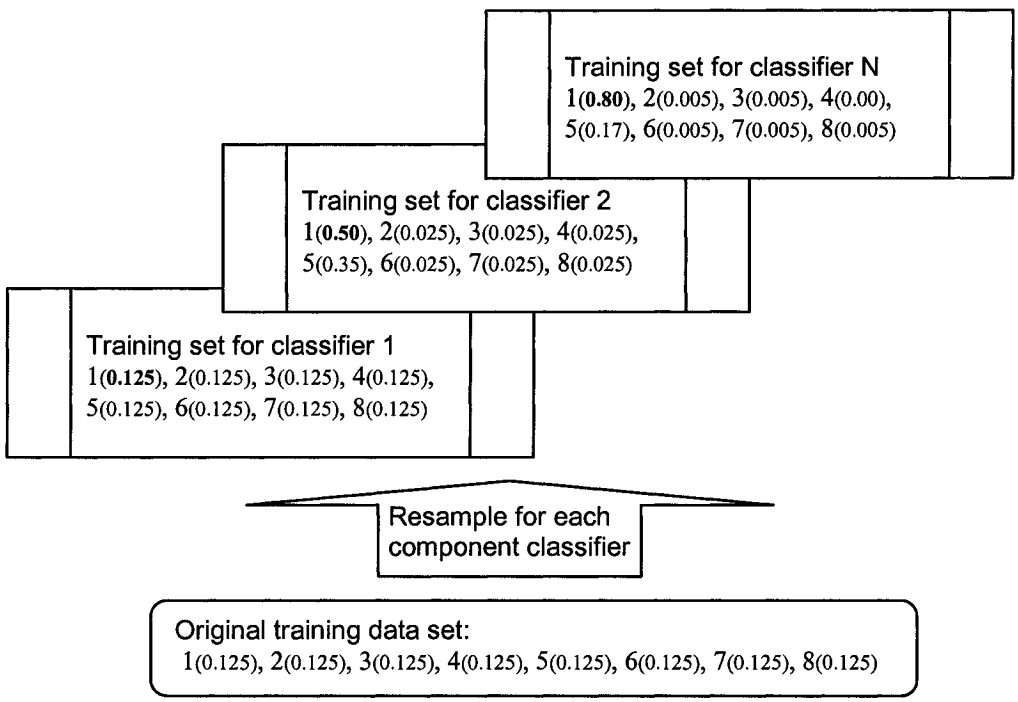


Fig. 5. Boosting with re-weighting, each instance is followed by a weight

The AdaBoost algorithm concentrates on the harder examples, and tends to challenge the weak learning algorithm to perform well on these harder parts of the sample space. Empirical results show an interesting property of the AdaBoost method. That is, the performance of the AdaBoost algorithm on the test set improves significantly after the error on the training set has become zero. Shapire et al. observed, as shown in Figure 6, that after just five classifiers have been combined, the training error of

the combined hypothesis has already dropped to zero, but the test error continues to drop until iteration 1000 [17].

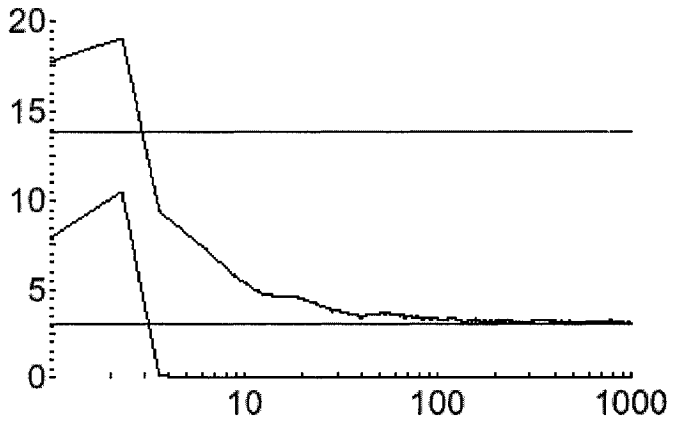


Fig. 6. Accuracy change per training iteration. Shown are the training and test error curves (lower and upper curves, respectively) of Boosting C4.5 decision trees [17].

There are many variants based on the AdaBoost approach. The most widely used AdaBoost algorithm is the AdaBoost.M1 method [9]. The AdaBoost.M1 method is an extension of the original AdaBoost algorithm which includes the multi-class case. Several more sophisticated methods have also been developed to extend the AdaBoost method to deal with multi-class problems. These methods generally work by reducing the multi-class problem to a larger binary problem. For example, the AdaBoost.M2 method uses the concept of *pseudo-loss* instead of error rate in the AdaBoost.M1 method [18]. The AdaBoost.M2 method modifies the weight distribution not only by considering whether the classifier correctly or incorrectly classified the training examples, but also by taking into account the confidence of labeling the examples. Another variation is the AdaBoost.R method, which was designed to be used for regression [63]. The LogitBoost algorithm is based on the observation that the

AdaBoost approach is, in essence, fitting an additive logistic regression model to the training data [19]. In the Arc-X4 method, the AdaBoost approach does not have the weighted voting mechanism, i.e., all component classifiers have the same weight during voting [16].

In the next part of this section, a detailed discussion of a popular and widely used AdaBoost algorithm, the AdaBoost.M1 method, is provided.

The AdaBoost.M1 Method

As depicted in Figure 7, the final AdaBoost.M1 classifier is constructed by a weighted vote of all the weak classifiers. The algorithm proceeds as follows. The initial weights of all examples are set to be *1/m*. After each trained classifier is added to the ensemble, the weights of examples being incorrectly classified by the current trained classifier are left unchanged. Otherwise, the weights are multiplied by a factor β_t. As shown in line 5 of Figure 7, the factor β_t corresponds to the error rate of the current trained classifier. The weights are then renormalized by dividing them by the normalization constant. Thus 'easy' examples, which are correctly classified by many of the previous trained classifiers, get lower weights, and hard examples, which tend often to be misclassified, get higher weights.

ALGORITHM AdaBoost.M1

Input: Sequence of m examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ with labels $y_i \in Y = \{1, \ldots, k\}$

Weak learning algorithm **WeakLearn**

Integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i.

Do for $t = 1, 2, \ldots, T$

1. Call **WeakLearn**, providing it with the distribution D_t
2. Get back a hypothesis $h_t : X \to Y$.
3. Calculate the error of $h_t : \varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$. If $\varepsilon_t > 1/2$, then set $T = t - 1$ and abort loop.
4. Set $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$.
5. Update distribution $D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \leftarrow \text{if } h_t(x_i) = y_i \\ 1 & \leftarrow \text{otherwise} \end{cases}$, where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis: $h_{fin}(x) = \underset{y \in Y}{\operatorname{argmax}} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$

Fig. 7. The AdaBoost.M1 algorithm [9]

An example to show how the AdaBoost.M1 method builds a good ensemble through focusing on hard examples and weighted voting is provided next. Following is an actual run of the AdaBoost.M1 algorithm using C4.5 decision trees as component classifiers [20]. In this illustrative example, three component classifiers are produced.

Table 2 shows the original data set used for the AdaBoost.M1 method. The data set contains two attributes (x and y), ten instances and two classes (+ and -). Tables 3, 4 and 5 show the training data sets used for training and some key numbers associated with the trained classifiers in the first, second and third iteration of the AdaBoost.M1 algorithm. Shown are (1) the error rate ε_t obtained by the current classifier, (2) the

current classifier's weight $log(\frac{1}{\beta_t})$, which is used to weighted vote in the final ensemble model, (3) the normalization constant Z_t, which is used to update the weights of misclassified instances, and (4) the weights of training instances before the training, after training and before renormalization, and after renormalization.

Table. 2. Original Training Data Set

Attribute1 (x)	**Attribute2 (y)**	**Class**
1	5	+
2	3	+
3	2	-
4	6	-
4	7	+
5	9	+
6	5	-
6	7	+
8	5	-
8	8	-

Table. 3. Training data after 1st iteration of boosting

Iteration 1: ε_t {Error Rate}:0.1,

$log(\frac{1}{\beta_t})$ { $\beta_t = \varepsilon_t / 1 - \varepsilon_t$ }: 2.1972, Z_t {Normalization Constant}: 0.9

Att1	Attr2	Weight Before Training	If classified correctly	Weight After Training	Weight After Renormalize
1	5	0.1	Yes	0.1	0.0555
2	3	0.1	Yes	0.1	0.0555
3	2	0.1	Yes	0.1	0.0555
4	6	0.1	Yes	0.1	0.0555
4	7	0.1	Yes	0.1	0.0555
5	9	0.1	Yes	0.1	0.0555
6	5	0.1	Yes	0.1	0.0555
6	7	0.1	Yes	0.1	0.0555
8	5	0.1	Yes	0.1	0.0555
8	8	0.1	No	0.9	0.5
Total Weights		1.0		1.8	1.0

Table. 4. Training data after 2nd iteration of boosting

Iteration 2: ε_t {Error Rate}: 0.1666,

$log(\frac{1}{\beta_t})$ { $\beta_t = \varepsilon_t / 1 - \varepsilon_t$ }: 1.6094, Z_t {Normalization Constant}: 0.4999

Att1	Attr2	Weight Before Training	If classified correctly	Weight After Training	Weight After Renormalize
1	5	0.0555	yes	0.0555	0.0333
2	3	0.0555	Yes	0.0555	0.0333
3	2	0.0555	No	0.2777	0.1666
4	6	0.0555	No	0.2777	0.1666
4	7	0.0555	Yes	0.0555	0.0333
5	9	0.0555	Yes	0.0555	0.0333
6	5	0.0555	No	0.2777	0.1666
6	7	0.0555	Yes	0.0555	0.0333
8	5	0.0555	Yes	0.0555	0.0333
8	8	0.5	Yes	0.5	0.3000
Total Weights		1.0		1.6666	1.0

Table. 5. Training data after 3rd iteration of boosting

Iteration 3: ε_t {Error Rate}: 0.1666,

$log(\frac{1}{\beta_t})$ { $\beta_t = \varepsilon_t / 1 - \varepsilon_t$ }: 1.6094, Z_t {Normalization Constant}: 0.4999

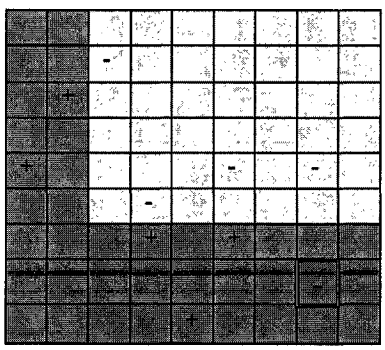
Att1	Attr2	Weight Before Training	If classified correctly	Weight After Training	Weight After Renormalize
1	5	0.0333	No	0.1666	0.0999
2	3	0.0333	No	0.1666	0.0999
3	2	0.1666	Yes	0.1666	0.0999
4	6	0.1666	Yes	0.1666	0.0999
4	7	0.0333	No	0.1666	0.0999
5	9	0.0333	No	0.1666	0.0999
6	5	0.1666	Yes	0.1666	0.0999
6	7	0.0333	No	0.1666	0.0999
8	5	0.0333	Yes	0.03333	0.02
8	8	0.3000	Yes	0.3000	0.1800
Total Weights		1.0		1.6666	1.0

After the three component classifiers have been trained, the final model is combined using the weighted vote as

Sign [2.1972*(classifier1? 1|-1) + 1.6094*(classifier2? 1|-1) + 1.6094*(classifier3? 1|-1)]

The *sign* here is interpreted as '+' or '-', which belongs to the label space of the training data. Using a weighted vote of the three component classifiers, the AdaBoost.M1 ensemble can achieve an error rate of 0%. The first trained component classifier of the AdaBoost.M1 ensemble achieved an error rate of 10%. The second and third classifiers both achieved error rates of 16.66%. Next, these results are further explained.

The above example can be visually demonstrated as follows. Figures 8, 9 and 10 show the hypotheses that the classifiers obtained in the learning space. The entire learning space is presented by the large square. The small dark gray square presents the hypothesis that are classified as class +, and the small light gray square presents the hypothesis that are classified as class -. In the left hand side of the figure, we also show the decision trees obtained by the component C4.5 classifier in each iteration of the AdaBoost.M1 algorithm.



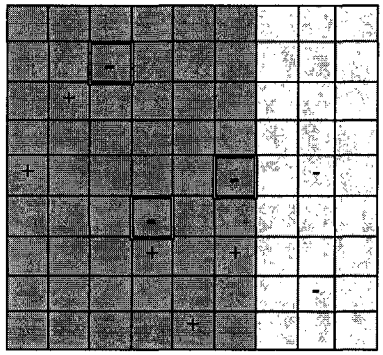
Iteration 1

■ Classified as +
▒ Classified as –
□ Misclassified

C4.5 Pruned Tree (horizontal is x, vertical is y)

```
x <= 2: + (2.0)
x > 2
|  y <= 6: - (4.0)
|  y > 6: + (4.0/1.0)
```

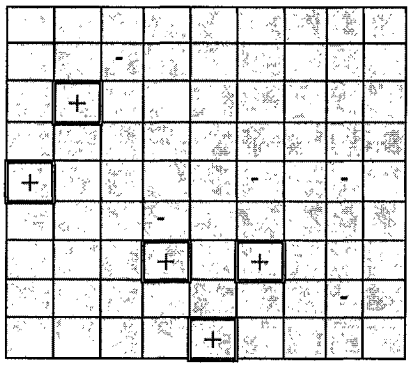
Fig.8. The 1st iteration



Iteration 2
- Classified as +
- Classified as –
- Misclassified

C4.5 Pruned Tree (horizontal is x, vertical is y)
x <= 6: + (4.44/1.67)
x > 6: - (5.56)

Fig.9. The 2nd iteration

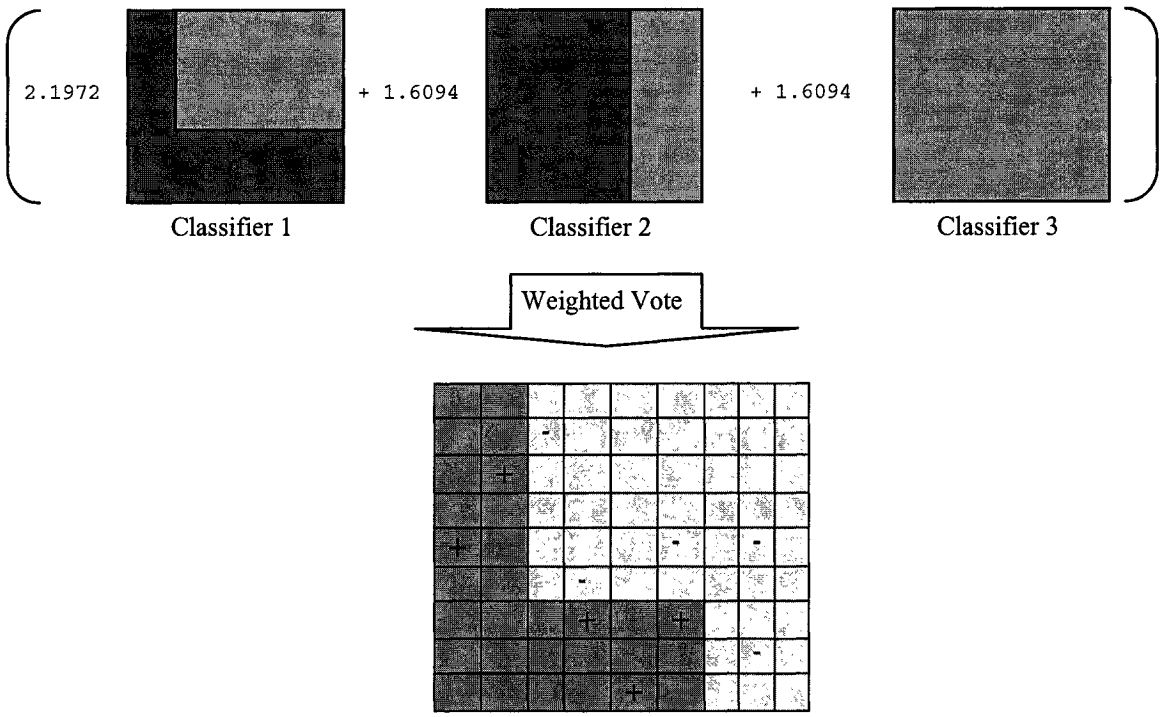


Iteration 3
- Classified as +
- Classified as –
- Misclassified

C4.5 Pruned Tree (horizontal is x, vertical is y)
: - (10.0/1.67)

Fig.10. The 3rd iteration

The AdaBoost.M1 algorithm combines the final model of the above procedure as follows. Each component classifier has a good hypothesis and a corresponding weight, which is based on its training error rate. In this way, the first, second, and third classifiers obtained the weights of 2.1972, 1.6094, and 1.6094, respectively, calculated as $log(\frac{1}{\beta_t})$. Using a weighted vote of these three component classifiers, the final model of the AdaBoost.M1 method is able to correctly classify all ten examples, achieving a 100% accurate rate against the training data set.



The Final Hypothesis achieved by the Boosting

This section provided an overview and an illustrative example of the AdaBoost approach. In the next section, a further analysis of this approach is presented.

2.3.2 Analysis of the AdaBoost Method

The success of the AdaBoost method is due to its effectiveness in increasing the margins of the training examples [17].

The margin of an example is defined as the difference between the weight assigned to the correct label and the maximal weight assigned to any single incorrect label [17]. Here, we refer to the sum of the weights of the base hypotheses that predict a particular

label as the weight of that label. In this way, the margin is a number between [-1, 1], and it indicates how close the decision boundary is to the data points on each side [17]. A higher margin on the training set should yield a lower generalization error. An example is classified correctly if and only if its margin is positive, and a large positive margin can be interpreted as a "confident" correct classification [17]. In term of a margin, examples with margins close to zero will be difficult to be correctly classified, and thus tend to be the hard examples.

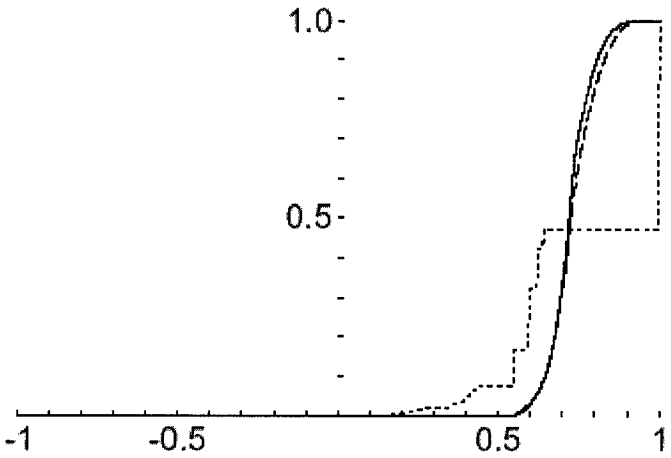


Fig. 11. Cumulative distribution of margins for the training sample after 5, 100, and 1,000 iterations [17]

Using the graph as shown in Figure 11, Schapire et al. showed the effectiveness of boosting the margin of the AdaBoost method [17]. Figure 11 depicts the margin distribution graphs of the AdaBoost method using the C4.5 decision tree as base classifiers on the "letter" data set from the UCI data repository [17, 26]. The figure shows the margin distributions for the AdaBoost method after 5, 100, and 1000 iterations, indicated by short-dashed, long-dashed and solid curves, respectively. Figure 11 shows that, after 100 iterations, all examples have a margin larger than 0.5. In comparison, after iteration 5, about 7.7% of the examples have margins below 0.5.

According to Schapire et al., "achieving a large margin on the training set results in an improved bound on the generalization error" [17]. The idea that maximizing the margin can improve the generalization error of a classifier was also suggested by many other researchers [17, 27, 28].

In terms of the margins, the AdaBoost method tends to increase the margins associated with examples. The AdaBoost method converges to a margin distribution in which most examples have large margins in order to increase the predictive performance of the classifiers. Shapire et al. also demonstrated that the AdaBoost method is especially aggressive in its effect on examples whose initial margin is small. Even though the training error remains unchanged (at zero) after iterations, the AdaBoost algorithms can still changes the margin distribution of examples quite significantly [17]. In the AdaBoost method, later classifiers focus on increasing the margins for examples with poor current margins.

Recall that the AdaBoost method is aggressive at increasing the margins of the hard examples by 'willingly' suffering reductions in the margins of the examples that already have large margins since, in this case, those reductions do not adversely affect the generalization error [17]. According to Shapire et al., this can be explained, in an intuitive way, as follows. Consider a classifier using a voting mechanism. If an example is classified by a large margin (either positive or negative), then small changes to the weights in the majority vote are unlikely to change the label. Also, if most of the examples have a large margin, then the classification error of the original majority vote will be similar to a perturbed vote [17].

This section discussed the Boosting algorithms' effectiveness in increasing the margins of training examples. The next section presents a challenging issue confronting the AdaBoost approach, namely the problem of over-emphasizing hard examples.

2.3.3 Over-emphasizing Hard Examples

Recall from Section 2.3.2 that the AdaBoost approach can greatly improve the predictive performance of the weak learners by focusing on increasing the margins for examples with poor current margins. However, recent research has shown that the AdaBoost method may fail to perform in some cases [1, 7, 8, 9, 13, 14, 15, 17]. This is due to the fact that, in these cases, the AdaBoost algorithm over-emphasizes hard or noisy examples, thus overfitting the training data sets [7, 8, 9, 13, 15, 17].

A number of research groups have indicated that the AdaBoost algorithm may suffer from over-emphasizing hard examples [1, 7, 9, 29]. Dietterich demonstrated that the AdaBoost algorithm tends to place much more weights on the noisy data. The emphasis placed on the hard examples can thus become detrimental to, or even destroy, the performance of the AdaBoost algorithms [29]. The author showed that the AdaBoost method tends to assign the examples, to which noise were added, much higher weight than other examples. As a result, component classifiers generated in later iterations cause the combined model to overfit the noise [29]. Also, Freund and Shapire suggested that the sometimes poor performance of the AdaBoost method results from overfitting the training set, since later training sets may be over-emphasizing examples that are noise (thus creating extremely poor classifiers) [7, 9]. In the authors' studies, they observed two such types of hard examples [9]. The first type is examples which are very atypical or wrongly labeled. The second type, which tends to dominate on later iterations, consists of examples that are very similar to each other but have different classes [9]. Moreover, Optiz indicated that the above argument seems especially pertinent to AdaBoost algorithms for two reasons [1]. The first, and most obvious, reason is that the method for updating the probabilities may be over-emphasizing noisy examples. The second reason is that the classifiers are combined using weighted voting. In AdaBoost algorithms, the later classifiers focus on increasing the margins for the examples with low current margins. In a domain with noisy data, putting too much effort on classifying misclassified examples might in fact be misleading the overall ensemble classification [1].

In this section, over-emphasizing hard examples, which was found to be a serious disadvantage of the Boosting algorithm, was discussed. The next section provides a detail discussion of approaches used to assist AdaBoost methods to avoid this problem.

2.3.4 Approaches to Avoid Over-emphasizing

A number of variants of the AdaBoost method were developed to help the AdaBoost algorithm avoid over-emphasizing hard examples. Five popular variants of the AdaBoost approach are discussed next.

The Gentle AdaBoost Method

Friedman et al. suggested a conservative variant of the AdaBoost method, called "Gentle AdaBoost" [19]. This approach puts less emphasis on outliers by slowing down the fitting procedure of the final ensemble model, thus avoiding over-emphasizing the outliers.

The authors reported that the AdaBoost algorithms are fitting an additive logistic regression model [19]. They derived a variety of loss functions for estimating a function $F(x) = \sum_{j=1}^{p} f_j(x_j)$ for classification. Typically these models are fitted by minimizing a loss function averaged over the training data. The Gentle AdaBoost method uses a gradient step to optimize the loss criterion to slow down the fitting procedure rather than exact optimization at each step, thus reducing susceptibility to overfitting [19]. In this way, the algorithm allows the weighting functions to increase more slowly than exponential.

The BrownBoost Method

In recent work, Freund suggested another algorithm based on his Boost-by-Majority algorithm, called the BrownBoost method, which takes a more radical approach [32]. In this approach, the boosting procedure will stop when it seems clear that some examples are "too hard" to be classified correctly, thus prevent the AdaBoost method from over-emphasizing the hard examples.

In the BrownBoost approach, the 'total time' parameter, c, sets the total amount of time for which the algorithm is set to run. During each of the iterations of the AdaBoost algorithm, a quantity t is subtracted from c, and the algorithm terminates when the remaining time s reaches 0. By incorporating an extra parameter (the time parameter), the algorithm knows in advance for how long it is to be run. It will therefore not attempt to fit examples that are unlikely to be learnable in the remaining time, since these examples are likely to be noisy. Given a good estimate of the time parameter which roughly corresponds to the proportion of noise in the training data set, the BrownBoost method has been observed to be particularly robust from over-emphasizing hard examples [19, 31]. The BrownBoost method forces the boosting procedure to stop earlier to avoid overfitting the hard data. However, in practice, it is difficult to estimate the parameter c [31].

The BB Algorithm Method

Krieger et al. combined both the Bagging and AdaBoost methods to produce a new algorithm, called the BB algorithm, which is particularly designed to outperform the AdaBoost method in noisy problems, while performing as well or nearly as well in non-noisy settings [30].

The BB algorithm performs a bootstrap aggregation on the boosted base learner. Here, the Bagging method is used to smooth the already boosted classifier trained by

a sub-sampling training data. By applying the Bagging method after the AdaBoost approach, the BB algorithm can average out (i.e. smooth) the noise contained in the training set when different boosted classifiers are combined into an ensemble by the Bagging method.

The SmoothBoost Algorithm Method

Servedio used the SmoothBoost method to construct a family of component classifiers of the AdaBoost approach in the presence of malicious noise [65]. The SmoothBoost approach constructs smooth probability distributions over the training data set, which does not put too much weight on any single example. To this end, the SmoothBoost method never assigns a weight greater than a threshold value to any single example. AdaBoost algorithms can fail if the algorithms generate distributions which are much skewed from the uniform distribution of the training data set [65]. By using a smooth weighted updating mechanism, the SmoothBoost method assures that the weak learner will function successfully at each stage, so that the overall boosting process will work correctly in the presence of malicious noise.

The MadaBoost Algorithm Method

Domingo and Watanabe proposed a new AdaBoost algorithm, called the MadaBoost method, by modifying the weighting system of the AdaBoost approach [66]. The MadaBoost approach is very simple. It just bounds the weight assigned to each example by its initial probability. In this way, the weights of the examples cannot become arbitrarily large as it happens in the AdaBoost algorithm. The authors observed that the uncontrolled growth of the weights seems to be the root of the problems of the AdaBoost algorithm and aim to address this issue [66].

Summary

In this section, several variants of the AdaBoost methods, which focus on avoiding over-emphasizing hard examples, were presented. The BrownBoost method makes the boosting procedure stop earlier. The Gentle AdaBoost method slows down the fitting procedure of the final model. The BB algorithm performs the Bagging method on boosted classifiers to average out noisy data. The SmoothBoost and the MadaBoost methods avoid putting too much weight on any single examples. In this ways, these variants of the AdaBoost methods are able to avoid overfitting the noisy data and improve AdaBoost methods' predictive performance. However, none of these five methods addressed one of the most critical drawbacks of the AdaBoost algorithm, namely how to identify hard examples which are easy to be over-emphasized. In contrast, the DataBoost algorithm, as developed in this research and as will be described in Chapter 4, addresses this important issue.

2.4 Conclusion and Discussion

This chapter focused on ensembles of classifiers. Section 2.1 provided a summary of the ensemble approach. This was followed, in Section 2.2, with an overview of the popular ensemble methods, namely the Bagging, Stacking and Boosting approaches. Finally, Section 2.3 provided a detailed discussion of the popular Boosting ensemble algorithm. Also, Section 2.3 discussed one of the most important research issues confronting the Boosting approach, namely how to prevent it from over-emphasizing hard examples. The section also included a review of several approaches to assist the Boosting method to improve the predictive performance.

In conclusion, the predictive performance of the ensemble methods is highly dependent the classifier method. The Boosting method has the ability to improve the predictive performance of the weak classification algorithm. However, a classifier that is too weak cannot guarantee that the ensemble will perform adequately [24]. On

the other hand, a strong classifier may lead to overfitting, which may become more severe during later iterations of the boosting procedure [1, 7, 9, 14, 15, 25].

In addition, the following comments regarding the ensemble size are noteworthy. Research shows that most of the reduction in error for ensemble methods occurs with the first few additional classifiers [1, 9]. Hansen and Salamon suggested that ensembles with as few as ten members were adequate to sufficiently reduce test-set error [21]. In their studies with decision trees, Schapire et al. suggested that it is possible to further reduce test-set error even after ten members have been added to an ensemble in the Boosting and Bagging methods [17]. Recently, Opitz and Maclin found that most of the gain in an ensemble's performance is achieved already after the first few classifiers have been combined [1]. However, relatively large gains can be seen up to 25 classifiers when boosting decision trees. It was traditionally believed that small reductions in test-set error may continue indefinitely for the Boosting approach [9]. However, Grove and Schuurmans demonstrated that AdaBoost algorithms can indeed begin to overfit with every large ensemble sizes (10,000 or more members), which implies that the AdaBoost method may overfit the data in the later iteration [23].

In the next chapter, I will discuss the class imbalanced problem and its challenges to the Boosting approach.

CHAPTER THREE

CLASS IMBALANCE PROBLEM

CHAPTER THREE

CLASS IMBALANCE PROBLEM

The previous chapter provided an overview of ensembles of classifiers and a discussion of the most popular ensemble method, namely the Boosting algorithm. This chapter focuses on a review of the current literature pertaining to learning from imbalanced data sets, while focusing on Boosting-based approaches. Recall from Chapter 1 that, for two-class problems, the class imbalance problem corresponds to domains for which one class is represented by a large number of examples while the other is represented by only a few [33]. Research has shown that learning from imbalanced data sets presents an important challenge to the machine learning community. This is due to the fact that traditional machine learning algorithms tend to ignore small classes while concentrating on classifying the large ones accurately (the so-called learning bias), thus producing poor predictive accuracy over the minority class [33, 34, 36, 37, 64]. In particular, when learning from imbalanced data sets, Boosting algorithms may worsen the learning algorithms' learning bias towards the majority class.

This chapter is organized as follows. Section 3.1 provides an overview of the class imbalanced problem. This is followed, in Section 3.2, with a description of previous proposals by other researchers for coping with it. The section presents a discussion regarding the use of Boosting approaches against imbalanced data sets. Finally, Section 3.3 concludes the chapter.

3.1 The Class Imbalance Problem

When learning from data sets with imbalanced class distributions, machine learning algorithms tend to produce biased classifiers. These biased classifiers have high predictive accuracy over the majority class, but poor predictive accuracy over the minority class [36]. This is due to the fact that there are many more examples of the majority class than of the minority class. In these cases, learning from the naturally occurring class distribution produces classifiers that perform poorly on minority-class examples.

In their experiments, Weiss and Provost have shown that examples belonging to the minority class are misclassified more often than examples belonging to the majority class [37]. Also, classifiers tend to perform worse on the minority class than on the majority class. The authors reported that, in their opinion, this drawback is due to two reasons. The first one is that the class "priors" in the natural training distribution are biased strongly in favor of the majority class. Some learners, such as decision trees, explicitly factor in these class priors, which biases the learning process by causing the majority class to be predicted more often than it otherwise would have been. This results in improved performance on the majority-class test examples, but degraded performance on the minority-class test examples [37]. The examples the authors gave is that, if a decision tree simply adopts the strategy of labeling the leaf with the majority class, the performance of the classifier on the majority will improve, but at the expense of the minority class. The second reason is that a classifier is less likely to fully flesh out the margins of the "minority" concept in the concept space because

there are fewer examples of that class to learn from. Thus, some minority-class test examples may be classified as belonging to the majority class [37].

In the work of Japkowicz et al., the authors further investigated whether or not the class imbalances are truly responsible for the degradation of the standard classifiers when learning from imbalanced data sets [67]. The authors suggested that the significant losses of performance in standard classifiers are not directly caused by class imbalances, but caused by the small disjuncts which may be yielded by the class imbalances [67].

3.2 Previous Approaches against Imbalanced Class Distribution

There have been several proposals for coping with imbalanced data sets [33]. In the following section, several popular proposed methodologies will be discussed. In this section, rather than comparing specific methods, various kinds of methods, i.e. sampling techniques, non-sampling techniques, and Boosting approaches, are introduced. A description of various Boosting approaches against imbalanced data sets is presented in Section 3.2.3.

3.2.1 Sampling Techniques

A common practice for dealing with imbalanced data sets is altering the class distribution of a data set. Doing so has been called 'Over-Sampling', which replicating cases from the minority, and 'Under-Sampling', which ignoring examples from the majority class. These methods cause the class distribution to become less imbalanced [33].

Over-Sampling

Over-sampling replicates examples in the minority class. In their experiments, Ling and Li replicate the minority-class examples to match the number of majority examples to the number of minority examples [38].

Under-Sampling

Under-sampling eliminates examples in the majority class while leaving the examples from the minority class untouched. Kubat and Matwin developed an under-sampling technique that intelligently removes harmful majority examples belonging to the following four groups [40]. The first group includes examples that suffer from the class-label noise. The second group is borderline examples that are close to the boundary between the positive and negative regions. The third group contains examples that are redundant so that this part can be taken over by other examples. The final group includes safe examples, e.g. examples which are defined as worth being kept for future classification tasks. After selecting a representative subset of the majority class examples, the training set becomes more balanced, thus alleviates the learning bias towards the majority class.

Integration of Over-Sampling and Under-Sampling

Chawla et al. combine under-sampling and over-sampling methods [34]. Instead of over-sampling by replicating minority-class examples, the authors over-sample the minority by generating synthetic examples from several minority-class examples that lie close together, in the SMOTE method [34]. In this way, on one hand, the SMOTE method can avoid overfitting the minority class examples. On the other hand, the SMOTE approach can spread the decision boundaries for the minority class into the majority-class space.

Discussion of Sampling Techniques

Sampling techniques cause the class distribution to become less imbalanced. However, both methods, namely the over-sampling and under-sampling techniques, have known drawbacks. The former does not add information and make overfitting more likely to occur since over-sampling typically makes exact copies of minority class examples. The latter actually removes information by throwing out some examples [64].

Japkowicz evaluated three strategies, namely under-sampling, re-sampling and a recognition-based induction scheme, using artificial data [33]. The author considered two sampling approaches, namely the Random sampling and Focus sampling techniques. Random re-sampling and Random under-sampling involve sampling one class at random until it consisted of as many samples as another class. Focused re-sampling re-samples only those minority examples that occurred on the boundary between the majority and minority classes. Focused under-sampling consists of under-sampling the majority class samples that lie away from the boundary. The author indicated that both the under-sampling and re-sampling methods were effective. Also, the author noted that using sophisticated sampling techniques did not give any clear advantage in the domains considered [33].

In the next section, some non-sampling techniques against imbalanced data sets are discussed.

3.2.2 Non-sampling Approaches

Changing the class distribution is not the only way to improve the predictive performance of a classifier when learning from imbalanced data sets.

Chan and Stolfo experimented to determine the best class distribution before training and subsequently generated multiple training sets with this class distribution [42]. When generating training sets with the best distribution for training, the authors usually included all of the minority-class examples and some of the majority-class examples in each training set. Also each majority-class example was presented in at least one of the training sets. The final model is a combination of all classifiers trained by each training set.

Japkowicz presented an approach, named learning by recognition, to deal with imbalanced class frequencies problems [68]. In this approach, first of all, an autoassociator, which was introduced by Japkowicz et al. in 1995 [69], is trained. After training, the autoassociator is used to classify novel examples, relying on the following idea. If being generalized correctly, the instance must be of the class it was trained on. Otherwise, the example must be of the other class. When applying the learning by recognition method, the threshold for discrimination between recognized and non-recognized examples is set by comparing the accuracy obtained with different threshold values and retraining the one yielding optimal classification performance [68].

Another non-sampling method is that of Cardie et al., who weighted examples in an effort to bias the learning toward the minority class [70]. In this approach, weights are carefully chosen to force the learning algorithms to put more efforts on the minority class examples.

Drummond and Holte argued that, using non-sampling techniques, all of the data available can be used to train the classifier, thus not throwing away information. Also, the learning speed is not degraded due to duplicate instances [41].

In sections 3.2.1 and 3.2.2, sampling and some non-sampling techniques against imbalanced class distribution were described. The next section provides a detailed discussion of Boosting approaches against imbalanced data sets.

3.2.3 Boosting Approaches against Imbalanced Data Sets

The following sections provide a description of some popular Boosting approaches against imbalanced data sets, i.e. the SMOTEBoost [46], AdaCost [48], CBS [47], RareBoost [44, 45], and ThetaBoost [49] methods.

The SMOTEBoost Approach

In the SMOTEBoost approach, Chawla et al. combined the Boosting method with synthetic data to improve the prediction of the minority class [46]. They used synthetic examples, as generated from several minority-class examples that lie close together, to change the class distributions, thus compensating for skew class distribution. By altering the class distribution in each round of the boosting procedure, the SMOTEBoost method can reduce the learning bias towards the majority class examples.

- Given: Set S $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ $x_i \in X$, with labels $y_i \in Y = \{1, \ldots, C\}$, where C_m, ($C_m < C$) corresponds to a minority class.
- Let $B = \{(i, y): i = 1,\ldots,m, y \neq y_i\}$
- Initialize the distribution D_1 over the examples, such that $D_1(i) = 1/m$.
- For $t = 1, 2, 3, 4, \ldots T$
  1. Modify distribution D_t by creating N synthetic examples from minority class C_m using the SMOTE algorithm
  2. Train a weak learner using distribution D_t
  3. Compute weak hypothesis h_t: $X \times Y \rightarrow [0, 1]$
  4. Compute the pseudo-loss of hypothesis h_t:
     $\varepsilon_t = \sum_{(i,y) \in B} D_t(i,y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$
  5. Set $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$ and $w_t = (1/2)\cdot(1-h_t(x_i,y)+h_t(x_i,y_i))$
  6. Update D_t: $D_{t+1}(i, y) = (D_t(i, y) / Z_t) \cdot \beta_t^{w_t}$
     where Z_t is a normalization constant chosen such that D_{t+1} is a distribution.
- Output the final hypothesis: $h_{fin} = \arg\max_{y \in Y} \sum_{t=1}^{T} (\log \frac{1}{\beta_t}) \cdot h_t(x, y)$

Fig. 12. Pseudo-code of the SMOTEBoost algorithm [46]

The SMOTEBoost approach creates synthetic examples from the minority class in each iteration of the boosting procedure. In this way, a new training data set which contains both the original data and synthetic data are used for training the component classifiers of the Boosting algorithms. The pseudo-code of the SMOTEBoost algorithm is shown in Figure 12.

As shown in Line 4 of Figure 12, the SMOTEBoost method modifies the framework of the AdaBoost algorithm. The main difference between the SMOTEBoost approach and the AdaBoost algorithm, as shown in Line 1 of Figure 12, is that the SMOTEBoost approach modifies the training distribution by creating a set of synthetic examples using the SMOTE algorithm [34].

In the SMOTE algorithm, new synthetic examples for the minority class are created [46]. The SMOTE method considers two types of attribute values, e.g. *continuous* and *nominal* attribute values. When considering *continuous* features, first of all, the SMOTE method takes the difference between a feature vector (minority class sample) and one of its k nearest neighbors (minority class samples). Secondly, this difference is multiplied by a random number between 0 and 1. This difference is added to the feature value of the original feature vector to form a new feature vector for the new synthetic minority instance. When considering *nominal* features, firstly, the SMOTE method determines the majority vote between the feature vector under consideration and its k nearest neighbors. In the case of a tie, a feature is chosen at random. After voting, the SMOTE method assigns this value to the new synthetic minority class sample [46].

This section described the SMOTEBoost method, which changes the class distribution of the Boosting algorithm to achieve a better performance against imbalanced data sets.

In addition, there has been significant interest in the recent literature for embedding cost-sensitivities in the Boosting algorithm when learning from imbalanced data sets. Four popular methods are discussed next.

The AdaCost Algorithm

In the AdaCost approach, the cost of misclassifications is used to update the training distribution on successive boosting iterations in order to reduce the cumulative misclassification cost [48]. As shown in Figure 13, the AdaCost method follows the framework of the standard AdaBoost algorithm. The main difference between the AdaCost method and the AdaBoost algorithm is the weight updating mechanism. In the AdaCost approach, a misclassification cost adjustment function, as shown in Line 4 of the Figure 13, is introduced in the weight updating mechanism. In this cost adjustment function $\beta(i) = \beta(sign(y_i h_t(x_i)), c_i)$, the c_i is the cost of instance i in the training data set and the *sign* is interpreted as the predicted label, which belongs to the label space $y = \{-1, +1\}$. It means that each training example i can have different cost c_i, and when being updated weight in the boosting procedure, each example is multiplied by different factor $\beta(i)$, which corresponding to the cost c_i with respect to the specific example, *i.e.* $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_t h_t(x_i)\beta(i)}{Z_t}$. Here D_{t+1} is the weight distribution of the round $t+1$, and Z_t is the normalization factor.

Using this cost adjustment function, the weights for expensive or rare class examples are higher. Inversely, the weights for inexpensive examples are comparatively lower. In this way, each weak classifier tries to correctly predict more expensive examples. In an imbalanced domain, the minority class examples will be assigned more weight in each iteration of the boosting procedures. Unlike the AdaBoost algorithm, in which misclassification cost is not used and more weights are equally given to wrongly classified training instances, the AdaCost algorithm

increases the weights of costly wrong classifications more aggressively. On the other hand, it decreases the weights of costly correct classifications more conservatively.

- Given: $\mathcal{S} = \{(x_1, c_1, y_1), \ldots, (x_m, c_m, y_m)\}$; $x_i \in \mathcal{X}, c_i \in \mathbb{R}^+, y_i \in \{-1, +1\}$.
- Initialize $D_1(i)$ (such as $D_1(i) = c_i / \sum_j^m c_j$).
- For $t = 1, \ldots, T$:
  1. Train weak learner using distribution D_t.
  2. Compute weak hypothesis $h_t : \mathcal{X} \rightarrow \mathbb{R}$.
  3. Choose $\alpha_t \in \mathbb{R}$ and $\beta(i) \in \mathbb{R}^+$.
  4. Update

$$D_{t+1}(i) = \frac{D_t(i)\exp\Big(- \alpha_t y_i h_t(x_i) \boxed{\beta(i)} \Big)}{Z_t}$$

  where $\beta(i) = \beta\Big(\text{sign}(y_i h_t(x_i)), c_i\Big)$ is a cost-adjustment function. Z_t is a normalization factor chosen so that D_{t+1} will be a distribution.
- Output the final hypothesis:

$$H(x) = \text{sign}(f(x)) \text{ where } f(x) = \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

Fig. 13. Pseudo-code of the AdaCost algorithm [48]

The CSB Algorithm

Another cost-sensitive adaptation of the AdaBoost approach, namely the CSB method, was introduced by Ting [47]. This approach is similar to the AdaCost method. In this approach, the author directly modifies the weight updating function in the original AdaBoost algorithms to convert the error-based AdaBoost models to the cost-sensitive Boosting algorithms. In the CSB approach, a cost factor is introduced into the weight-update rule to indicate that some instances are more

expensive than others, in terms of misclassification. The CSB method has three variants, denoted CSB0, CSB1, and CSB2, respectively. The three CSB methods are obtained by modifying the equation to calculate the value of D_{t+1} in the AdaBoost algorithm. The calculation of the weight distribution D_{t+1} in the CSB methods considers the cost factor of each instance. The CSB algorithms aim to reduce high cost errors in order to minimize the total misclassification cost. In an imbalanced data set, high cost factor is usually given to rare classes in order to enable the CSB algorithm to focus on correctly classifying minority class examples.

The RareBoost Algorithm

The AdaCost and CSB methods update the weights of examples according to the misclassification costs in order to force Boosting algorithms to put more efforts on the rare classes. Another approach, called the RareBoost method [44, 45], updates the weights of examples differently for *TP* (examples with positive labels are predicted as positive), *FN* (examples with positive labels are predicted as negative), *FP* (examples with negative labels are predicted as positive), and *TN* (examples with negative labels are predicted as negative) examples, when predicting rare class examples.

The RareBoost method has two variants, denoted RareBoost-1 and RareBoost-2, respectively. These two variants differ in the way that they update the weights of examples for *TP, FN, FP*, and *TN*. This discussion focus on the RareBoost-2 method. As shown in lines iii of Figure 14, the RareBoost-2 method updates the weights of four types of examples, i.e. *TP, FP, FN* and *FN* differently. This is done since it is expected that the RareBoost algorithm will achieve better *recall* and *precision* when compared to the AdaBoost method and other cost-sensitive Boosting approaches. Note that the *precision* and *recall* are calculated as *TP / (TP + FP)* and *TP / (TP + FN),* respectively. A classifier can achieve both high *recall* by better distinguishing *FN* from *TN*, and high *precision* by better distinguishing *FP* from *TP*.

Usually it is hard for learning algorithms to achieve high *recall* and high *precision* (*precision* and *recall* will be further discussed in Chapter 5.). Unlike the CSB and AdaCost algorithms, which focus on *FN* by varying cost factor and may result in high *recall* at a loss of *precision* or may result in high *precision* with poor *recall*, the RareBoost method specially targets the weak learner to model either *TP* vs. *FP*, or *TN* vs. *FN*. In this way, it aims to achieve better performance in both the *recall* and the *precision*, and obtain better *recall-precision* balance as well [44, 45].

Given: T, M.
Initialize weights $D_1(i) = 1/N$.
for $t = 1 \cdots M$

i. Learn weak model for $y_i = +1$ (class C) examples, $h_t^{+1} : x_i \rightarrow \{+1, 0\}$, using D_t.

ii. Learn weak model for $y_i = -1$ (class NC) examples, $h_t^{-1} : x_i \rightarrow \{-1, 0\}$, using D_t.

iii. Evaluate C's model and NC's model:

$$TP_t = \sum_{i:y_i>0, h_t^{+1}(x_i)>0} D_t(i);$$

$$FP_t = \sum_{i:y_i<0, h_t^{+1}(x_i)>0} D_t(i)$$

$$TN_t = \sum_{i:y_i<0, h_t^{-1}(x_i)<0} D_t(i);$$

$$FN_t = \sum_{i:y_i>0, h_t^{-1}(x_i)<0} D_t(i)$$

iv. Choose Model and Compute importance weight, α_t:
if($(1-(TP_t-FP_t)^2) < (1-(TN_t-FN_t)^2)$) then,
Choose C's Model, by setting $h_t = h_t^{+1}$:

$$\alpha_t = 0.5\, ln\, (TP_t/FP_t) \quad (11)$$

else Choose NC's Model by setting $h_t = h_t^{-1}$:

$$\alpha_t = -0.5\, ln\, (TN_t/FN_t) \quad (12)$$

endif

v. Update Weights:

$$D_{t+1}(i) = D_t(i) exp(-\alpha_t y_i)/Z_t; h_t(x_i) \neq 0, \quad (13)$$

where Z_t is chosen such that $\sum D_{t+1}(i) = 1$.

endfor
Final Model:

$$H(x) = sign\left(\sum_{t:x \text{ satisfies } h_t} \alpha_t\right)$$

Fig. 14. Pseudo-code of the RareBoost-2 algorithm [44]

The ThetaBoost Algorithm

Another approach, called the ThetaBoost algorithm, is similar to the CSB, RareBoost, and AdaCost methods [49]. The ThetaBoost approach suggests ways in which a hyper-plane should be optimized for imbalanced data sets where the loss associated with misclassifying the less prevalent class is higher [49]. The ThetaBoost algorithm modifies the AdaBoost algorithm by taking into account unequal loss functions and adjusting the margin for different classes in imbalanced data sets. The probability of each class for each example is estimated, and the examples are optimally re-labeled with respect to the misclassification costs. In this way, the error-based classifiers are made cost-sensitive. By re-labeling the examples, the algorithm can create new samples, from which the component classifiers may construct accurate models.

Discussion

When learning from data sets with imbalanced class frequencies, the main challenge is how to alleviate the Boosting methods' learning bias toward the major class examples. Two major methods have been investigated to assist Boosting algorithms to put more efforts on the minority class examples, i.e. re-sampling techniques and cost-sensitive methods. The SMOTEBoost algorithm changes the class distribution by creating new examples, thus alleviates the learning bias towards majority class examples. The CBS, AdaCost, ThetaBoost and RareBoost methods put more weights on the minority class examples during each of the iterations of the boosting procedure, in order to achieve better predictive performance. In the ThetaBoost method, examples are optimally relabeled with respect to the misclassification costs. Table 6 shows the summary of the most important characteristics of the approaches discussed above.

Table 6. Summary of Boosting approaches against imbalanced data sets

	Use misclassification cost or put more weights on minority examples	Use over-sampling techniques
AdaCost	Yes	No
CSB	Yes	No
RareBoost	Yes	No
ThetaBoost	Yes/re-label the examples	No
SMOTEBoost	No	Yes

In conclusion, as shown in Table 6, the AdaCost, CSB, RareBoost, and ThetaBoost approaches use misclassification cost to assist Boosting approaches to put much more effort on the minority class. The ThetaBoost approach also intelligently re-labels the examples based on the misclassification cost. In the SMOTEBoost approach, sampling technique is applied to alleviate the learning bias toward majority class. Approaches for integrating misclassification cost and sampling techniques, such as the DataBoost-IM algorithm, which will be discussed in Chapter 5, address both these issues.

3.3 Conclusion

This chapter discussed the class imbalance problem, where the training examples are not approximately equally represented, and its challenge to the Boosting approach. In Section 3.1, an overview of the class imbalance problem was provided. Section 3.2 presented a description of previous proposals by other research groups and included a discussion regarding the use of Boosting approaches against imbalanced data sets. In summary, the Boosting algorithms tend to ignore small classes while concentrating on classifying the large ones accurately, thus producing high predictive accuracy against the majority class, but poor predictive accuracy over the minority class. In this

section, several boosting-based approaches against imbalanced data sets were also described. A method to assist Boosting methods to be able to focus not only on hard examples, but also on rare examples, thus improving the predictive accuracies of both the majority and minority classes without forgoing one of the two classes will be discussed in Chapter 5.

Chapters 4 and 5 introduce the algorithms developed in this study, together with their experimental evaluations. The next chapter presents the DataBoost algorithm, which extends Boosting algorithms' predictive performance against domains with hard to learning examples through generating synthetic data. The chapter also includes an experimental evaluation. In chapter 5, the DataBoost-IM approach, which uses synthetic examples together with class re-balancing to extend Boosting algorithms' predictive performance against domains with imbalanced class frequencies, is presented.

CHAPTER FOUR

BOOSTING WITH DATA GENERATION

CHAPTER FOUR

BOOSTING WITH DATA GENERATION

Chapter 2 introduced the Boosting method and its shortcomings when learning from domains with hard to learn examples. Recall that hard examples are examples which are hard to be correctly classified and on which the Boosting algorithm needs to concentrate to achieve a good predictive performance. In this chapter, the DataBoost algorithm, which uses data generation to assist Boosting algorithms to avoid over-emphasizing hard examples, is described. In the DataBoost algorithm, hard examples are identified during each of the iterations of the Boosting algorithm. Subsequently, the hard examples are used to generate synthetic training data. These synthetic examples are added to the original training set and are used for further training. This chapter also includes an experimental evaluation of the DataBoost algorithm against ten data sets.

This chapter is organized as follows. In Section 4.1, the DataBoost algorithm is described. This is followed, in Section 4.2, with a comparative evaluation of the DataBoost algorithm against ten data sets. Finally, Section 4.3 concludes the chapter.

4.1 The DataBoost Algorithm

The DataBoost algorithm uses synthetic data to prevent Boosting algorithms from over-emphasizing hard examples. The synthetic examples thus provide complementary knowledge about hard examples on which the next iteration of the Boosting algorithm needs to concentrate.

The DataBoost algorithm, as shown in Figure 15, consists of the following four stages.

- During stage *A* (the initialization step of Figure 15), each example of the original training set is assigned an equal weight. The original training set is used to train the first classifier of the DataBoost ensemble.

- Secondly, during stage *B* (steps 1 and 2 of Figure 15), for each new classifier to be added to the DataBoost ensemble, a new training set is formed. This new training set is formed by identifying hard examples based on both the weights of the training examples and the accuracy of the current trained classifier being added to the DataBoost ensembles. For each of these hard examples, a set of synthetic examples is generated.

- During the third Stage *C* of the algorithm (steps 3 and 4 of Figure 15), the synthetic examples are re-weighted and then added to the original training set. Finally, the weights of the new training data set are renormalized.

- During the fourth Stage *D* of the algorithm (steps 5 to 9), the new training data set is used to train the component classifier. Similar to the AdaBoost.M1 algorithm, as discussed in Chapter 2, the original training data is re-weighted after training.

Following the AdaBoost.M1 algorithm, the second, third and fourth stages of the DataBoost algorithm are re-executed until *either* reaching the number of iterations as

specified or the current component classifier's error rate is worse than a threshold value. Similar to the AdaBoost.M1 ensemble method, this threshold is set to 0.5 in order to ensure each of the component classifiers is able to perform better than random guessing, as shown in Line 7 of Figure 15 [9].

ALGORITHM DataBoost

Input: Sequence of m examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ with labels $y_i \in Y = \{1, \ldots, k\}$

Weak learning algorithm **WeakLearn**

Integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i.

Do for $t = 1, 2, \ldots, T$

1. Identify the hard examples from the original training data set
2. Generate synthetic data based on the hard examples identified
3. Re-weight the synthetic examples generated
4. Add synthetic data to the original data set to form a new training data set
5. Call **WeakLearn**, providing it with the new training set with synthetic data
6. Get back a hypothesis $h_t : X \to Y$.
7. Calculate the error of $h_t : \varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$. If $\varepsilon_t > 0.5$, then set $T = t - 1$ and abort loop.
8. Set $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$.
9. Update distribution $D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t \leftarrow & \text{if } h_t(x_i) = y_i \\ 1 \leftarrow & \text{otherwise} \end{cases}$, where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis: $h_{fin}(x) = \underset{y \in Y}{\operatorname{argmax}} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$

Fig. 15. Pseudo-code of the DataBoost algorithm

The detailed DataBoost algorithm is described next. Throughout this discussion, the Hepatitis data set, which was obtained from the UCI data repository, is used as an illustrative example [26]. The Hepatitis data set contains 155 examples of Hepatitis patients, described by 19 continuous and discrete attributes. Of these cases, 123 corresponds to the patients who survived treatment (class 'Live') and 32 examples of mortalities (class 'Die').

4.1.1 Identifying Hard Examples

During the first stage of the algorithm, the difficult-to-learn training examples are identified. Here, the DataBoost algorithm identifies a set of hard examples and uses them to generate synthetic data.

The hard examples, namely the *seed* examples, are identified as follows. These values were found, by inspection, to produce data set sizes which augment the original training knowledge well.

- Firstly, the number of hard examples N_s is calculated. N_s is calculated as the number of the original training examples (E_{train}) multiplied by the error rate of the current trained classifier being added to the DataBoost ensemble.

- Secondly, the top N_s highest weighted examples are chosen as the seed examples (E_{seed}), which will be used as seeds to generate synthetic examples. Note that the weight of each seed examples will be kept with the seed example and will be used for assigning proper weights to the synthetic examples being generated.

That is, after each trained classifier is added to the DataBoost ensembles, the weights of the 'easy' training examples being correctly classified by the current trained classifier are left unchanged. For the hard examples, the weights are

increased by a multiplication factor. In this way, 'easy' examples that are correctly classified by many of the previous trained classifiers get lower weights, and hard examples, which tend often to be misclassified, get higher weights. The weights of the examples reflect the examples' probabilities of being misclassified by the previous classifiers.

Examples with higher weights are thus those which tend to be over-emphasized and need to be concentrated on by the DataBoost algorithm. Also, the accuracy of the current trained component classifier, which shows how many examples were misclassified by the current training, is used in the calculation of the number of seed examples.

*For the illustrative Hepatitis data set, assume that, in the fifth iteration of the Boosting algorithm, the current trained classifier's error rate is 16%. The hard examples set N_{seed} will consist of the 24 examples, which is calculated as (155*16%), with the highest weights as selected from the sorted E_{train}. Note that E_{seed} may contain hard examples from both the 'Live' and 'Die' classes.*

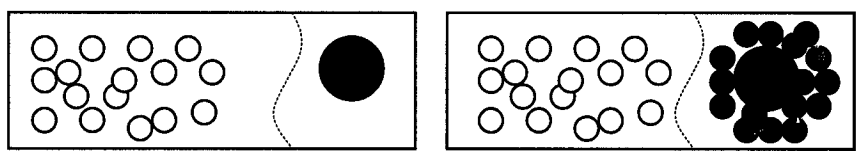
4.1.2 Generating Synthetic Data

Generating synthetic examples is an important step in the DataBoost algorithm. The data generation method used in the DataBoost algorithm is an extension of the method used in the Cooperative Inductive Learning Team (CILT) system, which was introduced by Viktor [11].

The data generation process generates a set of new examples that are based on each of the hard examples identified from the training set. These synthetic examples are added to the original training set to produce a new training set that is used to improve the current classifier's knowledge.

Such a new training set contains bias knowledge towards those particular examples and thus enhances the training process. Even though the new training set is biased towards the hard examples, care is taken to ensure that the relevant knowledge as embedded in the original training set is maintained. This is addressed in three ways. Firstly, the new training set contains the original training set together with the newly generated training examples. Secondly, the data generation process is constrained by generating new data without changing the class distribution of the original training examples. Finally, the original distribution of attribute values in the original training set are used when constructing the new training set [11].

Figure 16 shows results of a hypothetical run of the data generation process. Assume that the large bold circle presents the hard example, the empty circle is the easy instance, and the small bold circle presents the synthetic instance. Figure 16 (a) shows a hard example that has been identified, depicted by the large bold circle. This hard example is subsequently used as the seed to generate synthetic data. Figure 16 (b) shows how a number of synthetic data which are related to the hard example have been generated, as indicated by the small bold circles. Also, Figure 16 (b) depicts the new training data set, which contains a number of additional synthetic examples. This combined data set will be used as training set for the component classifiers used in the next iteration of the Boosting approach.



(a) Original training set **(b)** New training set

Fig. 16. Generating synthetic examples based on the hard examples to form new training data which contains synthetic examples with bias information and the original training examples

In detail, the data generation process proceeds as follows [11, 57]. Consider a data set that contains n attributes $A_1 \ldots A_n$, where $n \geq 1$. The data set contains m classes $C \ldots C_m$, where $m \geq 1$. Each class C_k in the original training set has a total of E_k examples, where $k=1 \ldots m$. Assume that the hard example set contains a hard example R_j, which describes class C_k. It is the task of the data generation process to generate a new set of examples that reflects the knowledge contained in R_j. The class distribution of this set of examples reflects the original class distribution of $E_1 \ldots E_m$ examples.

The first step of the data generation involves the generation of attribute values for each attribute A_i used in hard example R_j. Next, the data are formed into a new training set. The generation of values for each attribute A_i, as contained in R_j, is described next [11, 57]. Note that each of the attributes is considered separately.

- Suppose A_i denotes a **nominal** attribute. Consider R_j contains an attribute-value test $(A_i = x_i)$, where x_i is an element of a finite set of possible values V. For class C_k, the data generation process produces a total of E_k attribute values x_i, where $x_i \in V$ is randomly chosen to represent the distribution of values contained in the original training set with respect to class C_k. For each complementary class C_q, the data generation process produces a total of E_q attribute values x_i, where $x_i \in V$ is randomly chosen to represent the distribution of values contained in the original training set with respect to class C_q.

  For example, the attribute 'GENDER' in the Hepatitis data set has a value of either 'MALE' or 'FEMALE'. Assume that for the class 'Live', the number of occurrences of 'MALE' is 16 and 'FEMALE' is 107. The data generation creates 16 occurrences of 'MALE' and 107 occurrences of 'FEMALE'. These 123 values are randomly assigned to the 123 examples for class 'Live' created during data generation. In this data generation procedure, a set of 22 examples with class 'Die' are generated as well, following the same method.

- Assume A_i denotes a **continuous** attributes. Consider the attribute-value test ($A_i = x_i$). The data generation process proceeds to form, for each class C_k in the original data set a total of E_k attribute values that fall in the range [$minC_k \leq A_i \leq maxC_k$]. These values are randomly generated. Here, $minC_k$ denotes the minimum value of attribute A_i with respect to class C_k and $maxC_k$ refers to the maximum value of A_i for class C_k, as found in the original training set. The mean value and variance of A_i with respect to class C_k is determined and the distribution of the values of A_i is constrained by generating values without changing the original mean value or the variance of the attribute.

  For example, assume that, for the 'ALBUMIN' attribute in the Hepatitis data set, the 123 values for class 'Live' lies between 2.1 and 6.4, and the mean and deviation values are 3.817 and 0.652. The data generation randomly generates a total of 123 values between 2.1 and 6.4, following a mean value of 3.817 and a deviation value of 0.652. Again, the 123 values are randomly assigned to the 123 examples, with class 'Live', generated. In this data generation procedure, a set of 22 examples with class 'Die' are generated, following the same method.

The results of the first step are, for each attribute contained in example R_j , a total of $E_1 \ldots E_m$ attribute values that correspond to each of the classes $C_1 \ldots C_m$. The next step of the data generation process involves the merging of the attribute values into a set of training examples. This is achieved by, for each class C_k, combining the r^{th} value of attributes $A_1 \ldots A_n$, where $1 \leq r \leq k$, into one example [11, 57].

For the Hepatitis example, recall from Section 4.3.1 that E_{seed} contains 24 examples and E_{train} contains 155 examples. Followed the above-mentioned approach, the data generation process generates 24 sets of examples, each set contains 155 synthetic examples for each one of the seeds in E_{seed}. A total set consisting of 3720 synthetic

examples is thus newly generated. These instances are added to the original training set, leading to a final training set containing 3875 instances.

Interested readers are referred to [11, 57] for a detailed description of this process. Note that, in Viktor's earlier research reported in [11, 57], data were generated based on high-quality *rules*, as extracted by the current base classifier. Here, high-quality rules were defined as those rules with accuracy higher than a pre-set threshold value. In contrast, in the DataBoost approach, the new examples are generated based on the *data itself*, i.e. on the seed examples as identified during Section 4.1.1 Also, each of the seeds is re-weighted before training the next classifier, as discussed next.

4.1.3 Re-weighting the Synthetic data

Recall from Section 4.1.1 that, prior to the data generation process, a set of hard examples with weights that reflect the probabilities of being misclassified by the previous classifiers, are identified and used as seeds to generate new data. For each seed, the data generation process generates a set of synthetic examples. Each of synthetic examples will be assigned an initial weight corresponding to the weight of its seed used.

The initial weight of each example is calculated by dividing the weight of the seed example by the number of instances generated from it. Assume that we have a seed example E_s with weight W_s, and a total of N_d examples have been generated based on the seed example E_s. For each example of the N_d synthetic examples, a weight of (W_s / N_d) will be assigned. In this way, the very high weights associated with the hard examples are balanced out. Note that, prior to training, the weights of the new training set will be renormalized, following the AdaBoost method, so that their sum equals 1[9, 18]. The weight values of each synthetic examples were found, by inspection, to balance out the dominate weights of hard examples well.

For the Hepatitis example, assume that seed example x in E_{seed} has a weight of 0.986 and seed example y has a weight of 0.962. This implies that each of the 155 synthetic

examples generated based x is assigned a weight of 0.986/155 and those based on y are assigned weights of 0.962/155.

4.1.4 Motivation for Generation Synthetic Data

In the DataBoost algorithm, adding the synthetic training examples to the original data set has the following purposes. First of all, the synthetic training examples are able to balance out the dominating weights of the hard examples. Secondly, new knowledge with bias information is introduced. In this way, the component classifiers of the Boosting approach do not emphasize the hard examples, but rather a wider region of the training data space. This training data space contains, not only hard examples, but also examples with weights larger than the easy examples, but smaller than the hard examples.

An Illustrative Example

The aim of this example is to show how the DataBoost algorithm does not overemphasis the hard examples, but rather focus on a wider region of the training examples after synthetic data are added to the original data set. The following example also demonstrates how new knowledge is introduced into the learning process.

Assume that the illustrative data set has attributes Att1 and Att2. Att1 is nominal, and Att2 is continuous. This data set has two classes denoted by + and - and each of the instances has a certain weight after a particular iteration of the boosting procedure, as indicated in Table 7.

Table 7. Training instances of the illustrative example

Instance number	Att1	Att2	Class	Weight
1	A	9	-	0.05
2	A	3	+	0.05
3	B	10	-	0.2
4	B	6	-	0.2
5	C	7	+	**0.5**

Assume that the original training data is used as is. During the next iteration of the Boosting algorithm, too much emphasis will be put on example number five, since it has a dominating weight of 0.5. The task of the DataBoost algorithm is to force the Boosting approach to switch its focus from instance number five to instances number three, four *and* five. The process is illustrated in the following paragraph.

In the DataBoost algorithm, instance number five is identified as a hard example, and five synthetic examples with weight of 0.1 are created by the data generation procedure. The synthetic data have the following properties.

- The same mean (7.0) and variance (2.73) as and preserve the max and min values of the original data set for attribute Att1.
- The same number of occurrence of the nominal values as the original data set for the attribute Att2.
- The class frequencies of the original data is maintained, i.e. the number of instances belongs to the class + and class -.

The new training examples with their new weights are shown in Table 8, where the synthetic examples are highlighted in italics and the original examples are highlighted in bold.

Table 8. The new training instances of the illustrative example

Instance number	Att1	Att2	Class	Old Weight	New Weight after being normalized
1	**A**	**9**	**-**	**0.05**	**0.033333**
2	**A**	**3**	**+**	**0.05**	**0.033333**
3	**B**	**10**	**-**	**0.2**	**0.133333**
4	**B**	**6**	**-**	**0.2**	**0.133333**
5	**C**	**7**	**+**	**0.5**	**0.333333**
6	*A*	*7.2*	*-*	*0.1*	*0.066667*
7	*A*	*9*	*+*	*0.1*	*0.066667*
8	*B*	*10*	*-*	*0.1*	*0.066667*
9	*B*	*5.7*	*-*	*0.1*	*0.066667*
10	*C*	*3.1*	*+*	*0.1*	*0.066667*

The above data set shows that new knowledge with information biased towards instances number three and four are introduced by the synthetic instances number eight and nine, respectively. As such, the next classifier will put more emphasis on instances number five, four and nine, and number three and eight, instead of putting its focus on instance number five only. This effect is presented in the following table.

Table 9. The training focuses of the illustrative example

Instance number	Att1	Att2	Class	Weight	*Virtual Weight for the training process*
1	**A**	**9**	-	**0.033333**	*0.033333*
2	**A**	**3**	+	**0.033333**	*0.033333*
6	*A*	*7.2*	-	*0.066667*	*0.066667*
7	*A*	*9*	+	*0.066667*	*0.066667*
10	*C*	*3.1*	+	*0.066667*	*0.066667*
3	**B**	**10**	-	**0.133333**	*0.2*
8	*B*	*10*	-	*0.066667*	
4	**B**	**6**	-	**0.133333**	*0.2*
9	*B*	*5.7*	-	*0.066667*	
5	**C**	**7**	+	**0.333333**	*0.333333*

4.2 Experimental Design

This section describes the experiments to evaluate the DataBoost algorithm. The experimental evaluation presents the results of evaluating the prediction accuracy of the DataBoost algorithm, in comparison with the AdaBoost.M1 method, using both decision trees and neural networks as base classifiers.

4.2.1 Data Sets

To evaluate the performance of the AdaBoost.M1 and DataBoost methods, a number of data sets from the UCI data repository were obtained [26]. These data sets were

carefully selected to ensure that they (a) are based on real-world problems, (b) are varied in characteristics, and (c) were deemed useful by previous researchers. Table 10 gives the characteristics of the data sets used for the experiments. Shown are the type of the features in the data set (i.e., nominal, continuous, or a mix of the two), the number of output classes and the number of cases in the data set.

Table 10. Summary of the data sets used in this experiment. Shown are the number of examples in the data set; the number of output classes; the number of continous and discrete input features.

Data set	Case	Class	Feature	
			Continuous	Discrete
Balance-scale	625	3	0	4
Breast-cancer-data	286	2	0	9
Contact-lenses	24	3	0	4
Congress-voting-1984	435	2	0	16
Hepatitis-domain	155	2	6	13
Horse-colic-data	368	2	7	15
Monk2	169	2	0	6
Monk3	122	2	0	6
Post-operative	90	3	0	8
Tic-tac-toe	958	2	0	9

4.2.2 Methodology and Experimental Results

The experiments were implemented using Weka [3], a Java-based knowledge learning and analysis environment developed at the University of Waikato in New Zealand.

For the Monk2 and Monk3 data sets [58], the entire population of 432 instances was used as the test data set, averaged over executing the algorithm fifty times. Results for other data sets are averaged over five standard ten-fold cross validation experiments. For each ten-fold cross validation the data set was first partitioned into ten equal sized sets. Next, each set was, in turn, used as the test set while the classifier trained on the other nine sets. For each fold an ensemble of ten classifiers was created. Cross validation folds were performed independently for each algorithm.

Table 11. Test set accuracy rates for the data sets using (1) AdaBoost.M1 ensembles of decision trees; (2) DataBoost ensembles of decision trees (3) Improvement of DataBoost Approach (4) AdaBoost.M1 ensembles of Neural Network; (5) DataBoost ensembles of Neural Network; (6) Improvement of DataBoost Approach

Data set	C4.5			Neural Network		
	Ada Boost.M1	Data Boost	Imp. over AdaBoost.M1	Ada Boost.M1	Data Boost	Imp. over AdaBoost.M1
Balance-scale	77.9	76.2	-1.7	98.7	99.4	+0.7
Breast-cancer-data	65.4	67.8	+2.4	71.3	72.4	+1.1
Contact-lenses	66.7	83.3	**+16.6**	66.7	66.7	0
Congress-voting-1984	96.3	96.6	+0.3	95.2	95.2	0
Hepatitis-domain	81.3	84.4	+3.1	78.7	80.0	+1.3
Horse-colic-data	79.9	83.3	+3.4	78.5	81.0	+2.5
Monk2	69.7	74.6	+4.9	100	100	0
Monk3	92.6	93.5	+0.9	93.5	93.5	0
Post-operative	54.4	61.1	**+6.7**	56.7	61.1	**+4.4**
Tic-tac-toe	96.6	97.8	+1.2	97.0	97.0	0

C4.5 decision trees were used as base classifiers [20]. For the neural networks standard back-propagation learning was used. The parameter settings for the neural networks include a learning rate of 0.3, a momentum term of 0.2, and the number of hidden layers was calculated by (attributes +classes)/2 [3]. These values were set by inspection.

Table 11 shows the test set accuracy rates for the data sets described in Table 10, for both the two decision tree and two neural network ensembles. Also shown are the results of the comparison of the DataBoost and AdaBoost.M1 algorithms. The table shows that the DataBoost algorithm, in many cases, improved on the AdaBoost.M1 algorithm in terms of predictive accuracy. In particular, the DataBoost method produces equal or higher accuracies against all data sets, except a slight decrease of 1.7 against the balance-set data set when using C4.5 decision trees as component classifiers. In several cases, the experimental result shows that the DataBoost algorithm can obtain a good improvement over the AdaBoost.M1 approach in term of the overall accuracy. For example, in the Post-Operative data set, the improvements of DataBoost algorithm over the AdaBoost.M1 approach are 6.7 and 4.4, respectively, while boosting the C4.5 decision trees and Neural Networks. Similarly, for the Horse-colic data set, the improvements are 3.4 and 2.5. When boosting the C4.5 decision trees, the Databoost algorithm yielded higher accuracies than the AdaBoost.M1 approach against the Contact-lenses, Breast-Cancer-data, Hepatitis and Monk2 data sets with improvements of 16.6, 2.4, 3.1, and 4.9, respectively. When boosting the Neural Networks, the DataBoost algorithm achieved some improvements against the Balance-scale, Breast-cancer, Hepatitis, Horse-colic and Post-operative data sets, with improvements of 0.7, 1.1, 1.3, 2.5, and 4.4. The two algorithms produced equal accuracies against the five other data sets, i.e. Contact-lenses, Congress-voting-1984, Monk2, Monk3, and Tic-tac-toe.

One conclusion drawn from the results is that the DataBoost method appears to consistently reduce the error rate for almost all of the data set, and in many cases this reduction is large. The DataBoost algorithm consistently produces good results across both neural network and decision tree ensembles. Furthermore, analysis of results suggests that the largest reductions in error are produced for small data sets. The Post-operative data with 90 examples and the Contact-lenses data with 24 instances obtained the largest error reductions in the experiments, notably 16.6 and 6.7 respectively when using decision tree component classifiers. Such large reductions make sense, since for small data sets it is much easier to over-emphasize the hard

examples. Also, small data sets tend to not have sufficient knowledge from the hard examples for the learning processing, thus prevent Boosting algorithms from effectively concentrating on the hard examples.

Moreover, the experiments demonstrated that the DataBoost approach can obtain larger performance gains when applied to domains where the learning accuracies are low. For examples, the accuracies of the AdaBoost.M1 approach against the Post-operative data set are 54.4% when boosting C4.5 decision trees and 56.7% for Neural Networks. For such weak learners, the DataBoost algorithm achieved improvements of 6.7 and 4.4, respectively. Also, when considering the Contact-lenses and Monk2 data sets, the accuracies of the AdaBoost.M1 ensemble consisting of C4.5 decision trees are 66.7% and 69.7%. In these two data sets, the improvements obtained by the DataBoost algorithm are 16.6 and 4.9, respectively. These results indicate that the DataBoost algorithm has application in domains where the number of examples is few.

Paired t-test and Significance Analysis

When comparing two algorithms, statistical significance is used to determine whether there is a significant difference between the mean accuracies of the two algorithms. In other words, significance is used to state the degree of confidence that the obtained difference between the means of the two algorithms is too great to be a chance event [2]. The significance in difference in the above experiments is determined by averaging five ten-fold cross-validation. It would seem that a paired t-test could be applied to determine if the algorithms differ significantly in their mean accuracies [3]. The paired t-test is also purported to produce tighter confidence intervals on the mean, statistically. However, the following observation is noteworthy. The use of the paired t-test to determine the significance of results has been widely debated. For example, Dietterich evaluated several statistical tests, pointed out that, given a small data set, the k-fold cross-validated paired t-test will produce an artificially low probability value, e.g. high degree of confidence and should be interpreted cautiously

[71]. The problem with the paired t-test is now further elaborated. A paired t-test makes the assumption that individual measures are independent. The measures in a paired t-test are the differences in the k accuracy estimates of the two algorithms. K is the number of the training folds of the paired t-test. Unfortunately, cross-validation introduces a dependency among the measures, and this biases a paired t-test toward producing low probability value, namely producing wrong confidence on the level of difference. Note that, in a k-fold cross-validation the test sets are independent, but the training sets are overlap. In this way, the measures, i.e. the accuracies of the k models are dependent, which violates the assumptions underlying the t-test [71].

In the experiments presented in this thesis, all data sets are small, thus the k-fold cross-validation paired t- test is not ideal. Furthermore, in the DataBoost algorithm, a set of synthetic examples are created and added to the original training data set. This may cause the training sets to further overlap in the cross-validation paired t-test, thus worsening the violation of the independency assumption of the paired t-test.

Margin Analysis

Recall from Chapter 2 that the success of the AdaBoost method is due to its effectiveness in increasing the margins of the training examples. To further analyze the DataBoost approach's capability of increasing the margins of examples, the margins and their cumulative distribution of the previously illustrative Hepatitis data set are presented. These margins, as shown in Table 12, are shown for the C4.5 decision tree, together with AdaBoost.M1 and DataBoost ensembles, both consisting of C4.5 decision trees, produced over ten iterations. Recall, from Section 2.3.2, that the AdaBoost method is aggressive at increasing the margins of the hard examples by 'willingly' suffering reductions in the margins of those examples that already have large margins [17].

Table 12. Margins and their cumulative distribution of Hepatitis data set using (1) A C4.5 decision tree; (2) AdaBoost.M1 ensembles of C4.5 decision trees with ten iterations; (3) DataBoost ensembles of C4.5 decision trees with ten iterations.

C4.5 Margin Val.	Cum. Mar. Dist.%	AdaBoost.M1 Margin Val.	Cum. Mar. Dist.%	DataBoost Margin Val.	Cum. Mar. Dist.%
-0.24	**21.29**	**-0.008**	**18.71**	**-0.006**	**15.431**
0.0 Threshold					
0.344	21.935	0.008	20	0.029	15.882
0.924	92.903	0.8	66.452	0.816	71.613
0.932	93.548	0.812	69.677	0.82	74.194
0.936	94.194	0.816	71.613	0.832	77.419
0.948	94.839	0.844	73.548	0.844	78.065
0.984	97.419	0.896	74.839	0.86	80
1	100	1	100	1	100

Recall from Section 2.3.2 that the examples with margins less than zero tend to be misclassified. The data, as shown in the gray cells in the first line of Table 12, shows that the AdaBoost.M1 algorithm produces 18.71% examples with margins less than the threshold of 0.0. Similarly, the C4.5 decision tree produces 21.29% examples with margins which fall below the threshold of 0.0. In contrast, the execution of the DataBoost algorithm results in only 15.43% of examples with margins less than the threshold value. The result thus indicates that the DataBoost approach outperforms both the C4.5 decision tree and the AdaBoost.M1 algorithms in term of effectively increasing margins of examples whose margins were poor. This implies that the DataBoost algorithm is capable, for this illustrative example, of extending the AdaBoost.M1 algorithm's capability of increasing margins of poor performing examples.

4.2.3 Adding New Classifiers to the Ensemble

A further investigation of how DataBoost algorithm performs when new classifiers are added to the ensembles, is presented.

Recall that the Hepatitis data set contains 155 examples of Hepatitis patients, described by 19 continuous and discrete attributes. Of these cases, 123 corresponds to the patients who survived treatment (class 'Live') and 32 examples of mortalities (class 'Die').

The above experiment on the Hepatitis data set is repeated with 1, 2, 3, 5, 10, 50, 100, and 1000 component classifiers using the C4.5 decision trees as the base classifiers. The prediction accuracies are shown in Figure 17. Shown are the accuracy rates with 1, 2, 3, 5, 10, 50, 100, and 1000 classifiers, using C4.5 decision tree, AdaBoost.M1 and DataBoost ensembles consisting of C4.5 decision trees.

Figure 17 shows that most of the performance improvement of the Boosting algorithms (both the AdaBoost.M1 and the DataBoost algorithms) appears during iterations 3 and 5. The figure also shows that the improvement consistently increases when the number of component classifiers of the ensemble increases.

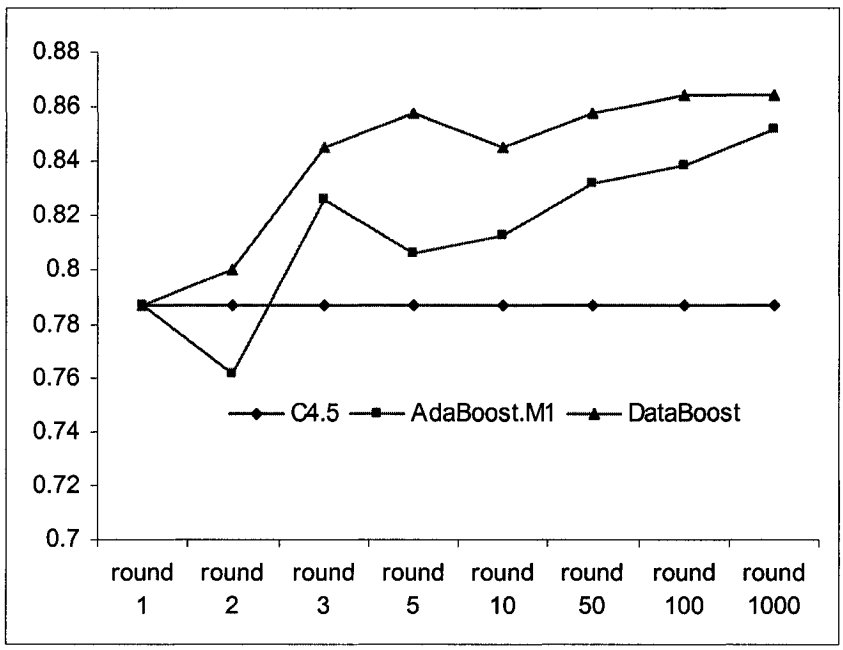


Fig. 17. Test accuracy rates on Hepatitis data set with 1, 3, 5, 10, 50, 100, and 1000 component classifiers using (1) C4.5 decision tree; (2) AdaBoost ensembles of C4.5 decision trees; and (3) DataBoost ensembles of C4.5 decision trees

These results indicate that, for this data set, the DataBoost approach consistently outperforms the AdaBoost.M1 algorithm when new classifiers are adding to the ensemble. The DataBoost methods achieved improvement over the AdaBoost.M1 algorithms during iterations 1, 2, 3, 5, 10, 50, 100, and 1000, as shown in Figure 15.

A further analysis of this figure indicates that in the second iteration of the Boosting approach, the AdaBoost.M1 algorithm did not obtain any improvement, but decreased the performance of the base classifier. However, even in this iteration, the DataBoost approach can achieve a significant improvement over the C4.5 decision tree base classifier. This result indicates that the DataBoost approach still performs well, even in cases where the AdaBoost.M1 algorithm fails to perform, against this data set.

4.3 Conclusion

This chapter introduced an algorithm in which synthetic data, which were generated based on hard to learn examples as identified during training, were combined with the Boosting approach. The DataBoost algorithm was illustrated by means of a number of data sets from the UCI data repository [26]. The results obtained indicate that the data generation approach extends the predictive performance of Boosting algorithms through augmenting the set of data used for training. The chapter also included an analysis of the margins of the DataBoost algorithm and the effect of adding new classifiers to an ensemble.

The results suggest two reasons for the accuracy improvement that is achieved by the DataBoost algorithm. The first is that the additional synthetic data with bias knowledge towards the hard examples prevent Boosting methods from over-emphasizing the hard examples. The second effect is that, through generating synthetic data in the proposed solution, complementary knowledge about the hard examples is provided for the learning process to focus on.

In the next chapter, namely Chapter 5, a detailed discussion of the DataBoost-IM approach for learning from domains with imbalanced class frequencies, is presented.

CHAPTER FIVE

BOOSTING IMBALANCED DATA SETS

CHAPTER FIVE

BOOSTING IMBALANCED DATA SETS

In the previous chapter, the DataBoost approach, which is used to extend the predictive performance of the Boosting algorithm against domains with hard-to-learn examples, was described. This chapter presents the DataBoost-IM approach for learning from domains with imbalanced class frequencies.

This chapter is organized as follows. Section 5.1 presented the DataBoost-IM algorithm. This is followed, in Section 5.2, with a comparative evaluation and a detailed discussion of the DataBoost-IM algorithm. The section also provides a discussion of performance measures for learning from imbalanced data sets. Finally, Section 5.3 concludes the chapter.

5.1 The DataBoost-IM Algorithm

The DataBoost-IM algorithm was developed to produce high predictive accuracy against both the majority and minority classes when learning from imbalanced data

sets. In the DataBoost-IM approach, the class frequencies and the total weights against different classes within the ensemble's training set are rebalanced by adding new synthetic data, during all iterations of the Boosting algorithm. The DataBoost-IM algorithm is shown in Figure 18.

ALGORITHM DataBoost-IM

Input: Sequence of m examples $\langle (x_1, y_1), ..., (x_m, y_m) \rangle$ with labels $y_i \in Y = \{1, ..., k\}$

Weak learning algorithm **WeakLearn**

Integer T specifying number of iterations

Initialize $D_1(i) = 1/m$ for all i.

Do for $t = 1, 2, ..., T$

1. Identify hard examples from the original data set for different classes
2. Generate synthetic data to balance the training knowledge of different classes
3. Add synthetic data to the original training set to form a new training data set
4. Update and balance the total weights of the different classes in the new training data set
5. Call **WeakLearn**, providing it with the new training set with synthetic data and rebalanced weights
6. Get back a hypothesis $h_t : X \to Y$.
7. Calculate the error of $h_t : \varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$ If $\varepsilon_t > 1/2$, then set $T = t - 1$ and abort loop.
8. Set $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$.
9. Update distribution $D_t : D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t \leftarrow & if \quad h_t(x_i) = y_i \\ 1 \leftarrow & otherwise \end{cases}$, where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis: $h_{fin}(x) = \underset{y \in Y}{argmax} \sum_{t:h_t(x)=y} log \frac{1}{\beta_t}$

Fig. 18. Pseudo-code of the DataBoost-IM algorithm

The DataBoost-IM approach extends the DataBoost algorithm, as presented in Chapter 4, as follows. Firstly, the DataBoost-IM method *separately* identifies hard examples from different classes. Secondly, the DataBoost-IM independently generates synthetic examples for different classes. Thirdly, the class distribution and the total weights of different classes are *rebalanced* by the DataBoost-IM method to alleviate the learning algorithms' bias toward the majority class.

Recall that Boosting methods involve the creation of a series of dependent classifiers which aim to correctly classify hard to learn examples during each of the iterations, through focusing on these hard examples during training. Following the DataBoost algorithm, the DataBoost-IM algorithm, as shown in Figure 18, consists of the following four stages. First of all, each example of the original training set is assigned an equal weight, e.g. each instance of the original training set has weight *1/m*. The original training set is used to train the first classifier of the DataBoost-IM ensembles. Secondly, the hard examples (so-called seed examples) are identified. Thirdly, for each of these seed examples, a set of synthetic examples, which contain the same class as the seed example, is generated. During the four step of the DataBoost-IM algorithm, the synthetic examples are added to the original training set and the class distribution, namely the number of examples from different classes, and the total weights of different classes, are rebalanced. Similar to the Databoost approach, the second, third and four stages of the DataBoost-IM algorithm are re-executed until *either* reaching a user-specified number of iterations or the current component classifier's error rate is worse than a threshold value. Following the DataBoost ensemble method, this threshold is set to 0.5.

The following sections provide a detailed discussions of the seed selection, data generation and re-balancing process of the DataBoost-IM algorithm. Throughout this discussion, the previously introduced Hepatitis data set is used as an illustrative example [26].

5.1.1 Identify Seed Examples

The first key step of the DataBoost algorithm is to identify hard examples, namely the seed examples. The aim of the seed selection process is to identify hard examples for both the majority and minority classes. These examples are used as input for the data generation process as discussed in Section 5.1.2.

The seed examples are identified and selected as follows. Firstly, the examples in the training set (E_{train}) are sorted in descending order, based on their weights. The original training set E_{train} contains N_{maj} examples from the majority class and N_{min} examples from the minority class. The number of examples that is considered to be hard (denoted by N_s) is calculated as (E_{train} *x Err*), where *Err* is the error rate of the currently trained classifier. Next, the set E_s, which contains the N_s examples with the highest weights in E_{train}, is created. The set E_s consists of two subset of examples E_{smin} and E_{smaj}, i.e. examples from the minority and majority classes, respectively. Here, E_{smin} and E_{smaj} contain N_{smin} and N_{smaj} examples, where $N_{smin} < N_{min}$ and $N_{smaj} < N_{maj}$. A number of seed examples of the majority class in E_{smaj} is selected by calculating M_L, which is equal to *min* (N_{maj}/N_{min}, N_{smaj}). Correspondingly, a subset M_S of the minority class examples in E_{smin}, is selected as seeds, where M_S is calculated as *min* ((N_{maj} *x* M_L) / N_{min}, N_{smin}). These values of M_l and M_s were found, by inspection, to produce data generation set sizes which augment the original training set well. The final sets of seed examples are placed in sets E_{maj} and E_{min}. Note that, when considering an imbalanced data set, the experimental results against seventeen data sets indicate that a very high percentage of minority class examples are hard examples with high weights. Due to this fact, experimental results show that for the seed examples, the number of higher weighted examples from the minority class is usually more.

Table 13. Seed examples and their weights of the Hepatitis Data set

Seed examples and their weights from the majority class (stored in E_{maj}):
2,female,y,y,y,n,n,y,y,n,y,n,n,1,59,249,3.7,54,n,LIVE (.986)
41,female,y,y,y,n,n,y,y,n,n,n,n,0.9,81,60,3.9,52,n,LIVE (.962)
Seed examples and their weights from the minrity class (stored in E_{minj}):
44,female,n,n,y,y,n,y,n,y,n,n,y,0.9,135,55,?,41,y,DIE {.436}
43,female,y,n,y,n,n,y,n,y,y,y,n,1.2,100,19,3.1,42,y,DIE {.436}
31,female,n,n,y,y,y,y,n,y,n,n,n,8,?,101,2.2,?,y,DIE {.348}
38,female,n,n,n,n,n,y,y,n,n,n,n,0.4,243,49,3.8,90,y,DIE {.323}
46,female,y,n,y,y,y,y,n,n,y,y,y,7.6,?,242,3.3,50,y,DIE {.323}
33,female,n,n,y,y,n,y,n,n,n,y,n,0.7,63,80,3,31,y,DIE {.235}
37,female,y,n,y,n,n,y,n,n,y,n,n,0.6,67,28,4.2,?,n,DIE {.235}
34,female,n,n,y,y,n,n,y,n,y,n,n,2.8,127,182,?,?,n,DIE {.208}

For the illustrative Hepatitis data set, assume that, in the fifth iteration of the boosting, the current trained classifier's error rate is 18%. The set E_s will consist of the 27 examples with the highest weights as selected from the sorted E_{train}. Assume that, of these 27 hard examples, 2 correspond to the class 'Live', and 25 examples are of the class 'Die'. Note that this high occurrence of examples from the minority class is due to the fact that, for imbalanced data sets, the minority class is harder to learn. M_L is equal to 3 and E_{maj} will thus contain both hard examples of the majority class 'Live'. M_s is equal to 8 and the set E_{min} will consist of the 8 highest weighted examples of class 'Die'. The output of this step is shown in the Table 13.

5.1.2 Generate Synthetic Data and Balance Class Frequencies

Following the DataBoost approach, the aim of the data generation process is to generate additional synthesis instances to *add to* the original training set E_{train}. The data generation process extends the DataBoost algorithm, as presented in the Chapter

4, by generating data for the majority and minority classes *separately*. That is, the data generation process generates two sets of data. Firstly, a total of M_L sets of new majority class examples, based on each seed instance in E_{maj}, are generated. For each attribute included in the synthetic example, a new value is generated based on the following constraints, which is similar to the approach used in the DataBoost algorithm, as described in Chapter 4.

- For a ***Nominal*** attribute, the data generation produces a total of N_{maj} attribute values for each seed in E_{maj}. The values are chosen to reflect the distribution of values contained in the original training attribute with respect to the particular class. This is achieved by considering, for each class, the number of occurrences of different attribute values in the original data set.

  Following the description in Section 4.1.2, for the illustrative Hepatitis data set, when considering the 'GENDER' attribute, the data generation process generates 16 occurrences of 'MALE' and 107 occurrences of 'FEMALE', thus reflecting the original training set distribution for the class 'Live'. Recall that these 123 values are randomly assigned to the 123 new instances being created. In contrast with the data generation method of the DataBoost approach, no examples of the class 'Die' are generated.

- For a ***Continuous*** attribute, the data generation produces a total of N_{maj} attribute values. The values are chosen by considering the range [*min, max*] of the original attribute values with respect to the seed class. Also, following the approach as discussed in Section 4.1.2, the distribution of original attribute values, in terms of the deviation and the mean, is used during data generation.

  For the illustrative Hepatitis data set, this procedure is similar to the method described in Section 4.1.2. For example, recall from Section 4.1.2 that for the 'ALBUMIN' attribute, the 123 values for class 'Live' lies between 2.1 and 6.4,

and the mean and deviation values are 3.817 and 0.652, respectively. The data generation process of the DataBoost-IM approach randomly creates a total of 123 values, which is assigned to each example. The data generator of the DataBoost-IM algorithm does not produce any instances of the class 'Die'.

Similarly, M_s different sets of new minority class examples, each based on a seed instance in E_{min}, are constructed. These sets of instances are added to the original training set.

For the illustrative Hepatitis data set example, recall from Table 13 that E_{maj} contains 2 examples for the class 'Live' and E_{min} contains 8 instances for the class 'Die'. According to the above-mentioned method, the data generation process creates two sets of examples for the class 'Live', each set contains 123 synthetic examples for each one of the seeds in $E_{maj,}$. Eight sets of examples containing a total of 32 synthetic examples of the class 'Die', based on the 8 seed examples in $E_{min,}$ will also be created. A total set consisting of 246 synthetic examples of 'Live' and 256 instances of 'Die' is thus newly generated, as shown in Table 14. These instances are added to the original training set, leading to a final training set containing 369 instances of the class 'Live' and 288 instances of the class 'Die'.

Table 14. The number of synthetic examples generated and the number of their seeds

	Total cases	Synthetic cases generated	Original cases	Seeds
Majority Class	369	246	123	2
Minority Class	288	256	32	8

5.1.3 Balance the Training Weights of Separate Classes

Recall that the data generation process generates sets of synthetic examples based on seed examples E_{maj} and E_{min} corresponding to the majority and minority classes. Before the generated data are added to the original data set, each of the synthetic examples is assigned an initial weight. The initial weight of each example is calculated by dividing the weight of the seed example by the number of instances generated from it. In this way, the very high weights associated with the hard examples are balanced out. Rebalancing ensures that the DataBoost-IM algorithm focuses on hard as well as minority class examples.

When the new training set is formed, the total weights of the majority class examples (denoted by W_{maj}) and the minority class examples (denote by W_{min}) in the new training data are rebalanced as follows. If $W_{maj} > W_{min}$, the weight of each instance in the *minority* class is multiplied by W_{maj} / W_{min}. Otherwise, the weight of each instance in the *majority* class is multiplied by W_{min} / W_{maj}. In this way, the total weight of the majority and minority classes will be balanced. The weight values of each synthetic examples were found, by inspection, to balance out the dominate weights of hard examples well. Note that, prior to training, the weights of the new training set will be renormalized, following the AdaBoost method, so that their sum equals 1 [9, 18, 59].

For the Hepatitis example, assume that seed example x in E_{maj} has a weight of 0.986 and seed example y has a weight of 0.962. This implies that each of the 123 synthetic examples generated based on x is assigned a weight of 0.986/123 and those based on y are assigned weights of 0.962/123. Similarly, an initial weight are assigned to each of the synthetic examples generated based on the seed examples from E_{min}. After adding the synthetic data to the original data set, the new training data set contains 369 examples of class 'Live' and 288 cases of the class 'Live'. Assume that W_{maj} is equal to 12.251 and W_{min} equals 6.983. Since $W_{maj} > W_{min}$, each of the 288 examples describing the minority class is multiplied by a constant equal to 12.251/6.983. As a result, the total weights of the majority and minority classes in the new training set are equal to 12.251, thus equally distributing the balance for the 2 classes.

5.2 Experimental Design

This section describes the results of evaluating the performance of the DataBoost-IM algorithm, in comparison with the C4.5 decision tree [20], AdaBoost.M1 [9, 18], DataBoost, AdaCost [48], CSB2 [47] and SMOTEBoost [46] boosting algorithms. The C4.5 algorithm, which has become a *de facto* standard against which new algorithms are being judged, is again used as base classifier [61].

When learning from imbalanced data sets, the traditional measure of evaluating the performance of a classifier, i.e. the overall error rate against test cases, is often not sufficient. Therefore, in the first part of this section, evaluation measures for learning from imbalanced data sets are discussed.

5.2.1 Performance Measures

The confusion matrix, as shown in Table 15, represents the typical metrics for evaluating the performance of machine learning algorithms. In Table 15, *TN* is the number of *True Negatives, FP* is the number of *False Positives*, *FN* is the number of *False Negatives*, and *TP* is the number of *True Positives.* The *TP Rate* and *FP Rate* are calculated as *TP / (FN+TP)* and *FP / (FP+TN).*

Table 15. Confusion Matrix

	Predicted Negative	Predicted Positive
Actual Negative	*TN*	*FP*
Actual Positive	*FN*	*TP*

Error Rate

Traditionally, the performance of a classifier is evaluated by considering the overall Error rate against test cases [51]. Error rate is defined as $1 - (TP+TN) / (TP + FP + TN + FN)$. It expresses the percentage of testing examples incorrectly recognized by the system, and it is the most common evaluation metric in machine-learning and data mining research. In classification, errors are either systematic, namely bias towards a learning algorithm, or due to the nature of the particular data set, namely variance [16].

However, when learning from imbalanced data sets, this measure is often not sufficient [51]. A classifier can achieve a high overall accuracy by simply predicting the majority class if the minority class examples are rare. For example, as shown in Lewis and Catlett's studies, if only 0.2% instances are positive, a classifier that simply labels every instance with the majority class will achieve an overall accuracy of 99.8% [35]. Even a system with accuracy close to 100% can thus be useless and the benefit of classifiers in similar domains must therefore be assessed by more appropriate criteria [40].

Metrics such as *recall, precision*, *F-Measures* [56], Receiver Operating Characteristic (*ROC*) curve [51], area under *ROC* curve (*AUC*) [51], and *G-mean* [40] have been used to better understand the learning performance on imbalanced data sets. These metrics are applicable to the class imbalanced problem since they take the class frequencies into account [52]. These metrics indicate the classifier's performance, not only on the majority class, but also on the rare examples, as explained next.

Precision and Recall

The *precision* and *recall* are calculated as $TP / (TP + FP)$ and $TP / (TP + FN)$. A classifier can achieve high *recall* by better distinguishing *FN* from *TN*, and high *precision* by better distinguishing *FP* from *TP*.

In classification systems, the *recall* measures the 'completeness' of the classification, i.e. it is the probability that an instance belonging to a class is classified into this class. The *precision* measures the 'purity' of the classification, it is the probability that an instance predicted to be in a class truly belongs to this class. *Recall* and *precision* goals are often conflicting, especially in imbalanced data sets.

F-Measure

The *recall* and *precision* goals are often conflicting and improving them simultaneously may not work well, especially when examples from one class are rare [46]. The *F-measure* incorporates the *recall* and *precision* into a single number, and aims to seek a balance between the two given no "a priori" knowledge of costs associated with *recall* and *precision* [45]. This metric can measure exact performance of machine learning algorithms on the interesting class, especially on rare examples. The *F*-measure is defined as

$$((1+\beta^2)\times Recall\times Precision)/\ (\beta^2\times Recall+Precision) \qquad (1)$$

where β corresponds to the relative importance of *precision* versus the *recall* and it is usually set to 1. The *F-measure* is high if both the *recall* and *precision* are high, and is dominated by the smaller of the two.

ROC Curve

The *ROC* curve [54] is a technique for summarizing a classifier's performance over a range, by considering the tradeoffs between *TP Rate* and *FP Rate* [55].

On an *ROC* curve, as shown in Figure 19, the X-axis represents the *FP* rate, calculated by *FP/ (TN+FP),* and the Y-axis represents the *TP* rate, calculated by *TP / (TP + FN)*. The *ROC* curve can measure a classifier's performance over the full range of possible class frequencies. *ROC* curves are produced by varying the threshold on a classification model's numeric output. One of the most attractive

properties of *ROC* curves is that they are insensitive to changes in class distribution. In other words, if the proportion of positive to negative instances changes in a test set, the *ROC* curve will not change.

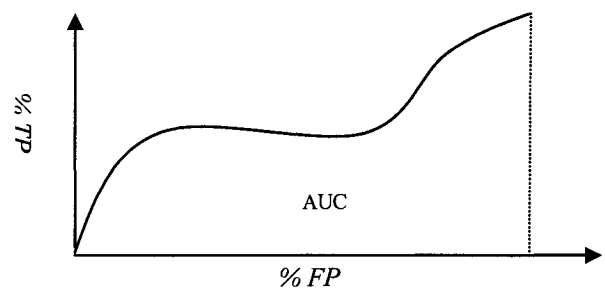


Fig. 19. ROC Curve

The area under the *ROC* curve (*AUC*), the total area that falls under the *ROC* curve, has recently been used to reduce the *ROC* curve to a single scalar value. The *AUC* effectively factors in the performance of the classifier over all costs and distributions [37]. Potentially, the larger *AUC* value, the better performance a classifier possesses. However, the *AUC* is sometimes not very interesting because it only shows how well a classifier predicts averaged over all possible threshold values.

G-mean

Another criteria used to evaluate a classifier's performance is the *G-mean,* which was introduced by Kubat et al [40, 53]. The *G-mean* is defined as

$$\sqrt{Positive\_Accuracy \times Negative\_Accuracy} \tag{2}$$

where *Positive Accuracy* is calculated as *TP/(FN+TP)*, and *Negative Accuracy* is calculated as *TN/(TN+FP)*. This measure relates to a point on the *ROC* curve. The idea of this measure is to maximize the accuracy on each of the two classes while

keeping these accuracies balanced. For example, a high *Positive accuracy* multiplied by a low *Negative accuracy* will result in poor *G-Mean* [40].

This section provides an overview of the appropriate measures to evaluate the performance of machine learning algorithms against imbalanced data sets. The next section describes the data sets used to evaluate the DataBoost-IM algorithm.

5.2.2 Data Sets

To evaluate the performance of the DataBoost-IM algorithm, sixteen data sets from the UCI data repository [26] as well as the Oil Spill data set [40, 53] were used. Following earlier experiments, these data sets were carefully selected to ensure that they (a) are based on real-world problems, (b) varied in feature characteristics, (c) vary extensively in size, and (d) varied in class distribution. Table 16 presents the characteristics of the data sets used for the experiments. Shown are the number of cases, the number of the majority and minority classes, the class distribution, and the type of the features. For the Glass, Vowel, Vehicle, Satimage, and Primary-tumor data sets, the degree of skew were increased by converting all but the smallest class into a single class. For the Sick data sets, the 'TBG' attribute was removed due to the high number of missing values. For the Abalone and Yeast data sets, this experiment respectively learned the classes '9' versus '18' in the Abalone, and the classes 'CYT' versus 'POX' in the Yeast in order to present the DataBoost-IM algorithm with highly imbalanced problems.

Table 16: Summary of the data sets used in this experiment. Shown are the number of examples in the data set; the number of minority class; the number of majority class; the class distribution; the number of continous, and the number of discrete input features.

Data set	Case	Min. Class	Maj. Class	Class Dist.	Feature Cont.	Feature Disc.
SONAR	208	97	111	0.47:0.53	60	0
MONK2	169	64	105	0.37:0.63	0	6
IONOSPHERE	351	126	225	0.35:0.65	34	0
BREAST-W	699	241	458	0.34:0.66	9	0
BREAST-CANCER	286	85	201	0.30:0.70	0	9
PHONEME	5484	1586	3818	0.29:0.71	5	0
VEHICLE	846	199	647	0.23:0.77	18	0
HEPATITIS	155	32	123	0.20:0.80	6	13
SEGMENT	2310	330	1980	0.14:0.86	19	0
GLASS	214	29	185	0.13:0.87	9	0
SATIMAGE	6435	626	5809	0.09:0.91	33	0
VOWEL	990	90	900	0.09:0.91	10	3
SICK	3772	231	3541	0.06:0.94	6	23
ABALONE	731	42	689	0.06:0.94	7	1
YEAST	483	20	463	0.04:0.96	8	0
PRIMARY-TUMOR	339	14	325	0.04:0.96	0	17
OIL	937	41	896	0.04:0.96	49	0

5.2.3 Methodology and Experimental Results

Following the experiment of the DataBoost algorithm, the experiments were implemented using Weka [3], a Java-based knowledge learning and analysis environment developed at the University of Waikato in New Zealand.

Results for the above data sets, as shown in Table 17 and Table 18, were averaged over five standard ten-fold cross validation experiments. For each ten-fold cross

validation the data set was first partitioned into ten equal sized sets and each set was then, in turn, used as the test set while the classifier trains on the other nine sets. A stratified sampling technique was applied to ensure that each of the sets had the same proportion of different classes. For each fold, an ensemble of *ten* component classifiers was created. In the experiments, the C4.5 decision trees were pruned [20].

The experimental results for the eigth most highly imbalanced data sets, as described in Table 16, are presented in Table 17. The results for the nine moderately imbalanced data sets are displayed in Table 18. For each data set, the results achieved when using the C4.5, AdaBoost.M1, DataBoost, AdaCost [48], CSB2 [47], SMOTEBoost [46] and DataBoost-IM methods are presented. For each algorithm, the tables present the results in terms of the *G-mean*, *overall accuracy rates, TP rates* and *F-measures* against the majority and minority classes, respectively. Following the work of Fan et al. [48], Ting [47], and Chawla et al. [46], the cost-adjustment functions from the AdaCost and CSB2 algorithms were chosen as follows: $\beta_- = 0.5 * C + 0.5$ and $\beta_+ = 0.5 * C + 0.5$, where β_+ and β_- are the functions for correctly and mislabeled labeled examples, respectively. Also, the three cost factors used in these two algorithms were 2, 5, and 9 [47, 48]. The parameter N, which specifies the amount of synthetically generated examples in the SMOTEBoost was set to 100, 300 and 500, respectively [46].

The results, as shown in Table 17 indicate that the DataBoost-IM algorithm performed well against highly imbalanced data sets, in terms of the *F-measures* of both the minority and majority classes. In many cases, the results were comparable or slightly higher than that produced by the other algorithms. The DataBoost-IM approach also yielded good results in terms of the *G-mean* and *overall accuracy*, when compared to the other approaches.

Table 17. Results against eight highly imbalanced data sets: *F-measure* of minority class, *F-measure* of majority class, *overall accuracy* , *G-mean*, *true positive rate* of minority class, and *true positive rate* of majority class for the data sets using (1) the C4.5 classifier (2) AdaBoost.M1, (3) DataBoost, (4) AdaCost, (5) CSB2, (6) SMOTEBoost, and (7) DataBoost-IM ensembles.

Data Set Name	Methods		F-measure of min. class	F-measure of maj. class	Overall Accuracy	G-Mean	TP rate of min. class	TP rate of Maj. Class
GLASS	C4.5		78.5	96.7	94.3	85.9	75.8	97.2
	AdaBoostM1		81.3	97.0	94.8	89.4	82.7	96.7
	DataBoost		86.2	97.8	96.3	84.3	86.2	97.8
	AdaCost	(Cost Factor: 2)	84.4	97.3	95.3	94.3	93.1	95.7
		(Cost Factor: 5)	77.6	95.8	92.9	91.5	89.7	93.5
		(Cost Factor: 9)	73.5	95.0	91.5	89.2	86.2	92.4
	CSB2	(Cost Factor: 2)	76.9	95.9	55.6	90.9	88.2	93.7
		(Cost Factor: 5)	59.3	89.2	50.0	87.4	94.1	81.3
		(Cost Factor: 9)	36.5	65.6	38.3	68.3	**95.0**	49.2
	SMOTEBoost	(N=100)	84.0	97.4	95.6	91.1	85.5	97.1
		(N=300)	82.5	97.1	95.0	91.1	86.2	96.4
		(N=500)	81.4	96.8	94.5	91.5	87.5	95.6
	DataBoost-IM		**89.2**	**98.3**	**97.1**	**92.3**	86.2	**98.9**
SATIMAGE	C4.5		56.4	95.4	91.7	72.7	55.2	95.6
	AdaBoostM1		66.7	**96.7**	**94.1**	77.0	60.7	**97.7**
	DataBoost		66.3	**96.7**	93.9	77.3	61.3	97.5
	AdaCost	(Cost Factor: 2)	64.9	95.8	92.4	82.3	71.6	94.7
		(Cost Factor: 5)	58.6	92.9	87.8	**88.1**	88.5	87.8
		(Cost Factor: 9)	51.4	89.6	82.9	87.2	93.0	81.8
	CSB2	(Cost Factor: 2)	64.7	95.4	91.8	84.5	76.4	93.6
		(Cost Factor: 5)	47.7	87.5	79.7	86.0	94.7	78.2
		(Cost Factor: 9)	30.5	68.2	56.3	71.4	**98.6**	51.8
	SMOTEBoost	(N=100)	64.5	96.5	93.7	75.6	58.6	97.5
		(N=300)	65.3	96.6	93.8	76.0	59.3	97.6
		(N=500)	65.2	96.6	93.8	76.1	59.4	97.5
	DataBoost-IM		**68.8**	**96.7**	**94.1**	80.4	66.6	97.1
VOWEL	C4.5		93.7	99.3	98.8	95.8	92.2	99.5
	AdaBoostM1		97.1	99.7	99.4	97.6	95.5	**99.8**
	DataBoost		95.5	99.6	99.2	94.1	94.4	99.7
	AdaCost	(Cost Factor: 2)	96.1	99.6	99.2	98.1	96.7	99.6
		(Cost Factor: 5)	96.7	99.7	99.3	98.7	97.8	99.6
		(Cost Factor: 9)	88.8	98.8	97.7	97.3	96.7	97.9
	CSB2	(Cost Factor: 2)	96.6	99.7	79.4	**99.6**	**100**	99.3
		(Cost Factor: 5)	78.8	97.3	95.1	96.8	98.9	94.8
		(Cost Factor: 9)	41.3	83.5	66.7	84.6	**100**	71.6
	SMOTEBoost	(N=100)	97.3	99.7	99.5	98.7	97.7	99.6
		(N=300)	96.1	99.6	99.2	97.8	96.2	99.5
		(N=500)	96.3	99.6	99.3	97.9	96.2	99.6
	DataBoost-IM		**98.8**	**99.8**	**99.7**	99.3	98.8	**99.8**

Data Set Name	Methods	F-measure of min. class	F-measure of maj. class	Overall Accuracy	G-Mean	TP rate of min. class	TP rate of Maj. Class
	C4.5	89.1	99.3	98.7	93.0	87.0	99.4
	AdaBoostM1	91.1	99.4	**98.9**	94.2	89.1	**99.5**
	DataBoost	90.3	99.3	98.8	96.1	93.1	99.2
	AdaCost (Cost Factor: 2)	**91.8**	**99.5**	**98.9**	95.9	92.6	99.4
SICK	(Cost Factor: 5)	90.8	99.4	98.8	97.3	95.7	99.0
	(Cost Factor: 9)	86.7	99.0	98.1	**97.6**	97.0	98.2
	CSB2 (Cost Factor: 2)	91.1	99.4	98.8	97.3	95.7	99.1
	(Cost Factor: 5)	73.3	97.6	95.5	97.1	**99.1**	95.3
	(Cost Factor: 9)	27.1	79.0	67.4	80.4	**99.1**	65.3
	SMOTEBoost (N=100)	89.6	99.3	98.6	95.5	92.1	99.1
	(N=300)	88.5	99.2	98.5	95.0	91.2	99.0
	(N=500)	87.9	99.1	98.4	94.9	91.0	98.9
	DataBoost-IM	**91.8**	99.4	**98.9**	95.9	92.6	99.4
	C4.5	36.0	**97.2**	**94.6**	50.8	26.1	**98.8**
	AdaBoostM1	41.0	96.9	94.1	59.0	35.7	97.6
	DataBoost	39.7	96.9	94.2	32.6	33.3	97.9
	AdaCost (Cost Factor: 2)	29.8	95.2	90.9	56.1	33.3	94.5
ABALONE	(Cost Factor: 5)	28.8	93.3	87.8	62.3	42.9	90.6
	(Cost Factor: 9)	26.3	90.9	83.8	65.5	50.0	85.9
	CSB2 (Cost Factor: 2)	39.6	95.8	92.0	65.4	45.2	94.9
	(Cost Factor: 5)	23.4	81.7	70.4	**74.1**	78.6	70.0
	(Cost Factor: 9)	15.7	60.1	45.8	61.7	**88.1**	43.3
	SMOTEBoost (N=100)	37.6	96.6	93.6	56.9	33.3	97.3
	(N=300)	39.0	96.7	93.7	58.1	34.7	97.3
	(N=500)	37.4	96.6	93.5	56.9	33.3	97.2
	DataBoost-IM	**45.0**	97.1	**94.6**	61.1	38.0	98.1
	C4.5	9.5	97.9	96.0	22.3	5.0	**100**
	AdaBoostM1	51.4	98.1	96.4	66.6	45.0	98.7
	DataBoost	54.1	98.2	96.5	70.2	50.0	98.5
	AdaCost (Cost Factor: 2)	40.9	97.2	94.6	66.0	45.0	96.8
YEAST	(Cost Factor: 5)	47.1	97.0	94.4	75.8	60.0	95.9
	(Cost Factor: 9)	43.8	96.0	92.5	**80.9**	70.0	93.5
	CSB2 (Cost Factor: 2)	55.6	98.3	96.6	70.2	50.0	98.7
	(Cost Factor: 5)	25.5	90.4	83.0	76.5	70.0	83.6
	(Cost Factor: 9)	14.5	74.6	60.8	69.2	**80.0**	60.0
	SMOTEBoost (N=100)	57.6	98.5	97.1	67.5	46.0	99.3
	(N=300)	53.0	98.2	96.5	68.0	47.0	98.7
	(N=500)	46.7	97.6	95.5	67.7	47.0	97.6
	DataBoost-IM	**58.0**	**98.6**	**97.3**	66.9	45.0	99.5
	C4.5	0.00	**97.8**	**95.8**	0.00	0.00	**100**
	AdaBoostM1	19.0	97.4	94.9	37.5	14.2	98.4
	DataBoost	17.3	97.1	94.4	14.0	14.3	97.8
	AdaCost (Cost Factor: 2)	12.0	93.0	87.0	43.8	21.4	89.8
PRIMARY-	(Cost Factor: 5)	11.0	89.3	80.8	48.7	28.6	83.1
TUMOR	(Cost Factor: 9)	14.6	88.3	79.3	58.9	42.9	80.9
	CSB2 (Cost Factor: 2)	16.3	93.5	87.9	50.8	28.6	90.5
	(Cost Factor: 5)	13.8	66.9	52.2	**68.4**	92.9	50.5

Data Set Name	Methods		F-measure of min. class	F-measure of maj. class	Overall Accuracy	G-Mean	TP rate of min. class	TP rate of Maj. Class
OIL		(Cost Factor: 9)	8.1	3.6	5.8	13.4	**100**	1.8
	SMOTEBoost	(N=100)	16.4	96.8	93.9	37.3	14.2	97.4
		(N=300)	21.4	97.0	94.3	42.3	18.5	97.6
		(N=500)	24.3	97.1	94.5	45.7	21.4	97.6
	DataBoost-IM		**28.5**	96.9	94.1	52.6	28.5	96.9
	C4.5		37.6	97.6	95.4	55.8	31.7	98.3
	AdaBoostM1		38.8	97.7	95.6	55.8	31.7	98.5
	DataBoost		45.5	98.0	96.2	36.2	36.6	98.9
	AdaCost	(Cost Factor: 2)	42.2	97.1	94.4	66.9	46.3	96.7
		(Cost Factor: 5)	35.8	95.5	91.5	70.7	53.7	93.3
		(Cost Factor: 9)	32.1	93.8	88.6	74.0	61.0	90.0
	CSB2	(Cost Factor: 2)	50.6	97.6	95.4	72.2	53.7	97.3
		(Cost Factor: 5)	33.3	91.9	85.4	**84.2**	82.9	85.6
		(Cost Factor: 9)	20.1	81.1	69.3	77.5	**87.8**	68.5
	SMOTEBoost	(N=100)	53.7	98.1	96.5	67.5	46.3	98.8
		(N=300)	49.4	98.0	96.2	63.9	41.4	98.7
		(N=500)	52.7	98.1	96.4	66.8	45.3	98.7
	DataBoost-IM		**55.0**	**98.2**	**96.6**	67.7	46.3	**98.9**

The DataBoost-IM approach achieved promising results when considering the minority class *F-measures*. In some cases, such as the Yeast and Sick data sets, the value obtained was the same as, or slightly higher than, the best value produced by the other algorithms. However, for data sets such as the Abalone, Primary-Tumor, Glass, and Satimage the minority class *F-measure* surpasses that of all the other techniques. Also, the majority class *F-measure* against five of the eight data sets as produced by the DataBoost-IM algorithm was the highest, or the same as, that obtained by the other algorithms. For the other three data sets, namely the Sick, Abalone and Primary-Tumor data sets, the values obtained were lower by only 0.1, 0.1 and 0.9, when compared to the best performing algorithm.

Table 18. Experimental results against nine moderately imbalanced data sets: *F-measure* of minority class, *F-measure* of majority class, *overall accuracy* , *G-mean*, *true positive rate* of minority class, and *true positive rate* of majority class for the data sets using (1) C4.5, (2) AdaBoostM1 ensembles, (3) DataBoost, (4) AdaCost, (5) CSB2, (6) SMOTEBoost, and (7) DataBoost-IM ensembles.

Data Set Name	Methods	F-measure of min. class	F-measure of maj. class	Overall Accuracy	G-Mean	TP rate of min. class	TP rate of Maj. Class
	C4.5	71.6	73.4	72.5	72.6	74.2	71.1
	AdaBoostM1	81.2	83.9	82.6	82.5	80.4	84.6
	DataBoost	81.4	83.8	82.7	68.2	81.4	83.8
	AdaCost (Cost Factor: 2)	73.7	71.4	72.5	72.6	82.5	64.0
SONAR	(Cost Factor: 5)	76.5	67.1	72.5	70.8	95.9	52.3
	(Cost Factor: 9)	72.7	52.6	65.3	59.7	99.0	36.0
	CSB2 (Cost Factor: 2)	79.7	73.3	76.9	75.9	96.9	59.5
	(Cost Factor: 5)	67.1	25.2	54.3	37.9	**100**	14.4
	(Cost Factor: 9)	64.2	5.3	48.0	16.4	**100**	2.7
	SMOTEBoost (N=100)	78.6	79.9	79.3	79.3	81.6	77.2
	(N=300)	76.9	80.4	78.8	78.5	75.6	81.6
	(N=500)	75.8	79.2	77.6	77.4	75.2	79.8
	DataBoost-IM	**82.1**	**84..9**	**83.6**	**83.3**	80.4	**86.4**
	C4.5	29.2	**74.6**	62.7	42.4	20.3	**88.5**
	AdaBoostM1	45.9	69.4	60.9	55.9	43.7	71.4
	DataBoost	41.0	68.8	59.2	27.1	37.5	72.4
	AdaCost (Cost Factor: 2)	53.8	35.5	46.1	44.3	82.8	23.8
MONK2	(Cost Factor: 5)	54.7	12.7	40.3	25.8	95.1	7.0
	(Cost Factor: 9)	54.9	0	37.8	0.0	**100**	0
	CSB2 (Cost Factor: 2)	**56.2**	28.1	45.5	39.7	92.2	17.1
	(Cost Factor: 5)	54.9	0	37.8	0.0	**100**	0
	(Cost Factor: 9)	54.9	0	37.8	0.0	**100**	0
	SMOTEBoost (N=100)	46.8	62.4	55.9	54.9	51.2	58.8
	(N=300)	54.3	60.2	57.5	58.8	66.8	51.8
	(N=500)	53.8	57.6	55.8	57.3	68.1	48.3
	DataBoost-IM	52.7	70.8	**63.9**	**61.1**	53.1	70.4
	C4.5	83.3	91.3	88.6	86.2	79.3	93.7
	AdaBoostM1	87.2	93.5	91.4	88.9	81.7	**96.8**
	DataBoost	89.3	94.3	92.6	82.8	86.0	96.3
	AdaCost (Cost Factor: 2)	88.4	93.6	91.7	90.6	87.3	94.2
Ionosphere	(Cost Factor: 5)	84.7	90.2	88.0	88.8	92.1	85.8
	(Cost Factor: 9)	79.6	84.9	82.6	84.7	94.4	76.0
	CSB2 (Cost Factor: 2)	89.7	93.6	82.9	**93.0**	96.5	89.7
	(Cost Factor: 5)	68.9	67.1	61.2	70.8	99.1	50.7
	(Cost Factor: 9)	54.3	12.0	35.8	25.3	**100**	6.4
	SMOTEBoost (N=100)	**90.2**	**94.7**	**93.1**	92.0	88.4	95.8
	(N=300)	89.4	94.4	92.7	91.2	86.5	96.1

Data Set Name	Methods		F-measure of min. class	F-measure of maj. class	Overall Accuracy	G-Mean	TP rate of min. class	TP rate of Maj. Class
		(N=500)	88.6	94.0	92.1	90.5	85.3	96.0
	DataBoost-IM		91.2	95.4	94.0	92.3	87.3	97.7
	C4.5		92.4	95.9	94.7	94.3	93.3	95.4
	AdaBoostM1		93.9	96.8	95.8	95.4	94.1	96.7
	DataBoost		93.7	96.7	95.6	90.8	94.2	96.4
	AdaCost	(Cost Factor: 2)	93.2	96.2	95.1	95.4	96.7	94.3
BREAST-W		(Cost Factor: 5)	93.5	96.3	95.2	96.0	98.8	93.4
		(Cost Factor: 9)	92.2	95.5	94.2	95.2	98.8	91.9
	CSB2	(Cost Factor: 2)	93.6	96.4	95.4	95.7	96.7	94.8
		(Cost Factor: 5)	87.9	92.3	90.5	92.4	99.6	85.8
		(Cost Factor: 9)	68.3	67.6	67.9	71.4	**100**	51.1
	SMOTEBoost	(N=100)	94.1	96.8	95.9	95.8	95.6	96.1
		(N=300)	94.3	96.9	96.0	95.9	95.6	96.1
		(N=500)	94.5	97.0	96.1	96.2	96.3	96.0
	DataBoost-IM		**95.2**	**97.4**	**96.7**	**96.4**	95.4	**97.3**
	C4.5		39.3	**84.3**	**75.1**	50.8	27.0	**95.5**
	AdaBoostM1		44.0	74.9	65.4	58.1	45.8	73.6
	DataBoost		40.2	75.9	67.8	29.9	38.8	77.1
	AdaCost	(Cost Factor: 2)	46.5	56.1	51.7	55.6	70.6	43.8
BREAST-		(Cost Factor: 5)	**47.3**	12.0	34.0	24.7	93.0	6.6
CANCER		(Cost Factor: 9)	46.1	0	29.9	0.00	**100**	0
	CSB2	(Cost Factor: 2)	42.4	41.5	41.9	45.9	71.8	29.4
		(Cost Factor: 5)	45.8	0	29.7	0.00	**100**	0
		(Cost Factor: 9)	45.8	0	29.7	0.00	**100**	0
	SMOTEBoost	(N=100)	44.9	75.6	66.2	58.8	46.3	74.6
		(N=300)	45.9	73.9	64.8	59.7	50.3	70.9
		(N=500)	46.1	72.8	63.9	59.8	52.0	68.9
	DataBoost-IM		46.5	77.0	67.8	**60.0**	47.0	76.6
	C4.5		77.2	90.0	86.1	84.2	79.8	88.7
	AdaBoostM1		81.8	92.6	89.4	86.7	80.8	93.0
	DataBoost		81.8	92.7	89.6	74.9	80.1	**93.5**
	AdaCost	(Cost Factor: 2)	78.8	88.9	85.4	87.3	92.3	82.6
PHONEME		(Cost Factor: 5)	71.9	81.5	77.7	82.2	97.3	69.6
		(Cost Factor: 9)	67.2	75.3	71.7	77.3	98.4	60.8
	CSB2	(Cost Factor: 2)	80.4	89.8	86.6	88.4	93.4	83.8
		(Cost Factor: 5)	58.0	57.4	57.7	63.4	99.5	40.4
		(Cost Factor: 9)	46.2	6.0	31.5	17.6	**100**	3.1
	SMOTEBoost	(N=100)	82.4	92.4	89.4	87.8	84.3	91.5
		(N=300)	80.5	91.3	88.0	86.9	84.4	89.4
		(N=500)	79.8	90.9	87.5	86.4	84.0	89.0
	DataBoost-IM		**83.8**	**93.2**	**90.5**	**88.4**	83.7	93.3
	C4.5		87.9	96.1	94.2	92.5	89.4	95.6
	AdaBoostM1		92.5	97.6	96.4	95.5	93.9	97.2
	DataBoost		91.6	97.5	96.1	88.8	91.0	97.7
	AdaCost	(Cost Factor: 2)	88.3	96.2	94.2	93.8	93.0	94.6
VEHICLE		(Cost Factor: 5)	86.2	95.0	92.6	94.3	97.5	91.2
		(Cost Factor: 9)	85.1	94.4	91.8	94.1	99.0	89.6

Data Set Name	Methods	F-measure of min. class	F-measure of maj. class	Overall Accuracy	G-Mean	TP rate of min. class	TP rate of Maj. Class
	CSB2 (Cost Factor: 2)	90.5	96.7	95.1	**96.1**	98.0	94.3
	(Cost Factor: 5)	73.1	87.3	82.7	87.8	99.5	77.6
	(Cost Factor: 9)	45.0	39.7	42.4	49.7	**100**	24.7
	SMOTEBoost (N=100)	92.4	97.6	96.4	95.3	93.4	97.3
	(N=300)	92.1	97.5	96.2	95.5	94.3	96.8
	(N=500)	91.0	97.1	95.6	95.1	94.1	96.1
	DataBoost-IM	**93.7**	**98.0**	**97.0**	95.7	93.4	**98.1**
	C4.5	42.1	86.9	78.7	57.9	37.5	89.4
	AdaBoostM1	52.4	88.3	81.3	66.8	50.0	89.4
	DataBoost	58.3	89.2	**84.4**	50.0	54.7	**91.5**
	AdaCost (Cost Factor: 2)	59.5	87.3	80.6	75.8	68.8	83.7
HEPATITIS	(Cost Factor: 5)	50.9	73.0	65.1	72.0	87.5	59.3
	(Cost Factor: 9)	48.8	66.3	59.3	68.7	93.8	50.4
	CSB2 (Cost Factor: 2)	**63.4**	86.8	80.6	**80.9**	81.3	80.5
	(Cost Factor: 5)	42.3	51.2	47.0	57.3	93.8	35.0
	(Cost Factor: 9)	36.0	13.6	26.4	27.0	**100**	7.3
	SMOTEBoost (N=100)	58.9	89.3	83.0	72.6	59.3	89.2
	(N=300)	59.2	88.4	82.0	74.0	63.1	86.9
	(N=500)	58.4	87.8	81.1	74.1	64.3	85.5
	DataBoost-IM	62.6	**89.7**	83.8	76.2	65.6	88.6
	C4.5	87.2	97.8	96.3	92.2	86.9	97.9
	AdaBoostM1	93.5	98.9	98.1	95.9	93.0	99.0
	DataBoost	93.9	99.0	98.3	92.5	93.3	99.1
	AdaCost (Cost Factor: 2)	90.7	98.4	97.2	96.0	94.5	97.7
SEGMENT	(Cost Factor: 5)	86.4	97.4	95.6	96.1	96.7	95.5
	(Cost Factor: 9)	84.3	96.9	94.7	96.2	98.5	94.1
	CSB2 (Cost Factor: 2)	91.7	98.5	97.4	**97.4**	97.3	97.5
	(Cost Factor: 5)	75.2	94.2	90.6	94.0	99.1	89.2
	(Cost Factor: 9)	44.7	74.1	64.7	76.6	**100**	58.8
	SMOTEBoost (N=100)	94.2	99.0	98.3	96.2	93.4	99.1
	(N=300)	95.4	**99.2**	98.6	97.2	95.2	**99.2**
	(N=500)	94.1	99.0	98.3	96.4	94.0	99.0
	DataBoost-IM	**95.5**	**99.2**	**98.7**	97.3	95.4	**99.2**

The results, when comparing the DataBoost-IM method to the base-line C4.5 and AdaBoost.M1 methods, showed that DataBoost-IM again performed well in terms of both *F-Measures*. In some cases such as the Glass, Abalone, Yeast, Primary-Tumor, and Oil spill data sets, there were large improvements. For example, in the Oil spill data set, the C4.5 and AdaBoostM1 algorithms produced minority class *F-measures* of 37.6 and 38.8, respectively. The DataBoost-IM approach achieved a minority class *F-measure* of 55.0. In the Primary data set, the C4.5 and AdaBoost.M1

algorithms produced minority class *F-measures* of 0 and 19.0 respectively, whereas the DataBoost-IM approach achieved a minority class *F-measure* of 28.5. Also, the *G-mean* of the DataBoost-IM method were, for all eight data sets, the same or slightly higher than that of the other two approach. Similar results holds for the *overall accuracy*, where DataBoost-IM performs slightly lower in only one case.

When considering the DataBoost-IM and the original DataBoost approaches, the values shown in Table 17 confirm that the DataBoost-IM approach benefits from generating synthetic examples for different classes separately and rebalancing the class frequencies and the total weights from the different classes. The DataBoost-IM approach achieved higher *F-measures* against both classes, except for the Primary-Tumor data set, in which the value against the majority class is lower by only 0.2, and the Satimage data set, in which the values against the majority class were the same. In many cases the improvement for the minority class was quite significant. For example, in the Primary-Tumor data set, the DataBoost method produced a minority class *F-measure* of 17.3, while the DataBoost-IM approach achieved a value of 28.5. Also, for the highly imbalanced Oil spill and Abalone data sets, the improvements achieved by the DataBoost-IM method over DataBoost algorithm were promising, i.e. 9.5 and 5.3, respectively. For the *G-mean* and *overall accuracy*, the DataBoost-IM method consistently produced similar or higher results than that of the DataBoost algorithm.

When comparing the DataBoost-IM method with the AdaCost, CSB2, and SMOTEBoost algorithms, the results showed that the DataBoost-IM method produced results which compare well, in terms of the *G-mean*, *overall accuracy* and *F-measures*. The DataBoost-IM method achieved similar or slightly better minority and majority class *F-measures* against the eight data sets. Importantly, the results indicate that the approach does not sacrifice one class, but produces high predictions against both. For example, for the Primary-Tumor data set, where the majority class *F-measure* values were lower by 0.2, when compared to the best values obtained by the other algorithms, the improvement of minority class *F-measure* were 4.2.

Similarly, for the Abalone and Glass data sets, the improvements in minority class *F-measures* were 5.3 and 4.8, respectively.

Further analysis of the moderately imbalanced data sets, as shown in Table 18, shows that the DataBoost-IM approach obtained the highest values against six of the nine data sets in terms of the minority class *F-measures*. Also, against seven of the nine data sets the majority class *F-measure* results were slightly higher. However, the improvements in term of *F-measures* were less significant than those against the highly imbalanced data sets. For the Monk2 and Breast-Cancer data sets, the majority class *F-measures* decreased by more than 40.0, when compared to the values obtained by the DataBoost-IM algorithm. The same results hold for the *overall accuracy* and *G-mean*, where DataBoost-IM produces comparable and slightly higher results in six of the nine data sets.

In conclusion, the results, as shown in Table 17 and Table 18, indicate that the results obtained by the DataBoost-IM approach are comparable to that of the other techniques, when evaluated in terms of *overall accuracy*, *G-mean* and *F-measures*. In particular, the results against highly imbalanced data sets are promising. Importantly, for some highly imbalanced data sets, the DataBoost-IM approach produces the highest results in terms of both minority and majority class *F-measures*. Results indicate that the DataBoost-IM technique does not sacrifice the one class in favor of the other. Rather, it aims at producing an ensemble which produces high values against both.

ROC Analysis

To better understand the achievements of the DataBoost-IM method, the *ROC* analysis results of the Hepatitis data set are presented. This data set was chosen since it contains both continuous and discrete features, and has a moderately imbalance degree and instance size. The previous experiment against the Hepatitis data set is repeated, using the optimal parameter values for all algorithms as shown in Table 17

and Table 18. This experiment varied the decision thresholds of the C4.5 algorithm, by varying the proportion of instances at the leaf node of the decision tree for labeling a class, to obtain the *ROC* curve. An average *ROC* curve, as shown in Figure 20, is produced by averaging the *TP* and *FP* rates over ten runs [55]. Also, the *ROC* curves of the ten iterations of the DataBoost-IM algorithm, as shown in Figure 21, are drawn.

One conclusion drawn from the analysis of the *ROC* curves, as shown in Figure 20, is that the DataBoost-IM ensemble's *ROC* curve is of a high quality. This result indicates that the DataBoost-IM method achieved a high outcome, which compares well to that of the C4.5, AdaBoost.M1, DataBoost and SMOTEBoost algorithms over most of the threshold values of the *ROC* space. Further analysis of Figure 21 shows us an essential fact of the DataBoost-IM approach, namely that each component classifier in the DataBoost-IM ensembles was pursuing a point with both a high *TP rate* and a low *FP rate* in the *ROC* space. This implies that the DataBoost-IM was able to produce a series of high quality classifiers, and each of them should thus be better able to predict examples for which the previous classifier's performance is poor.

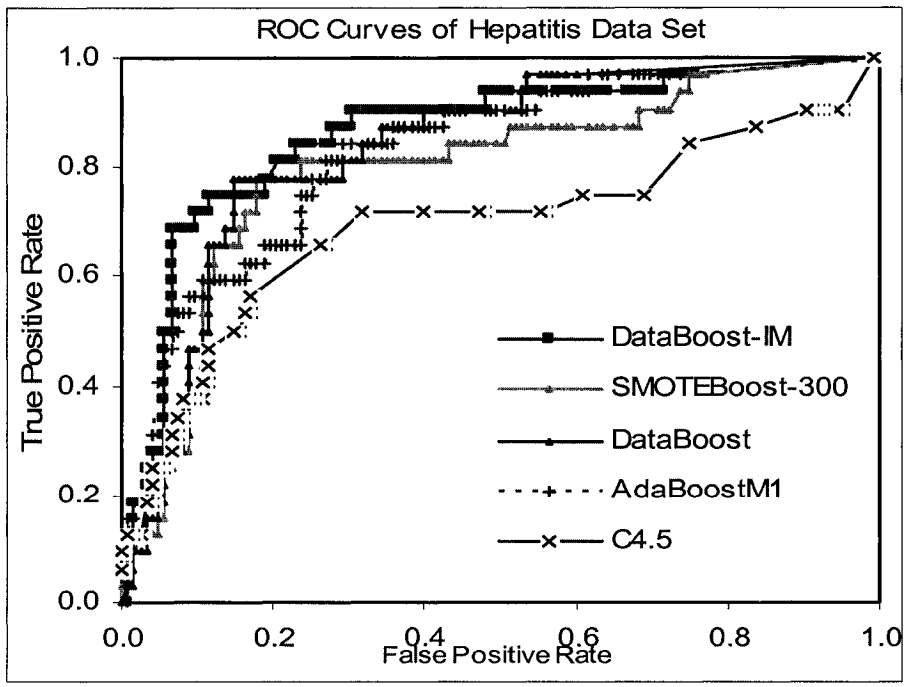


Fig. 20. ROC Curve of the Hepatitis Data Set

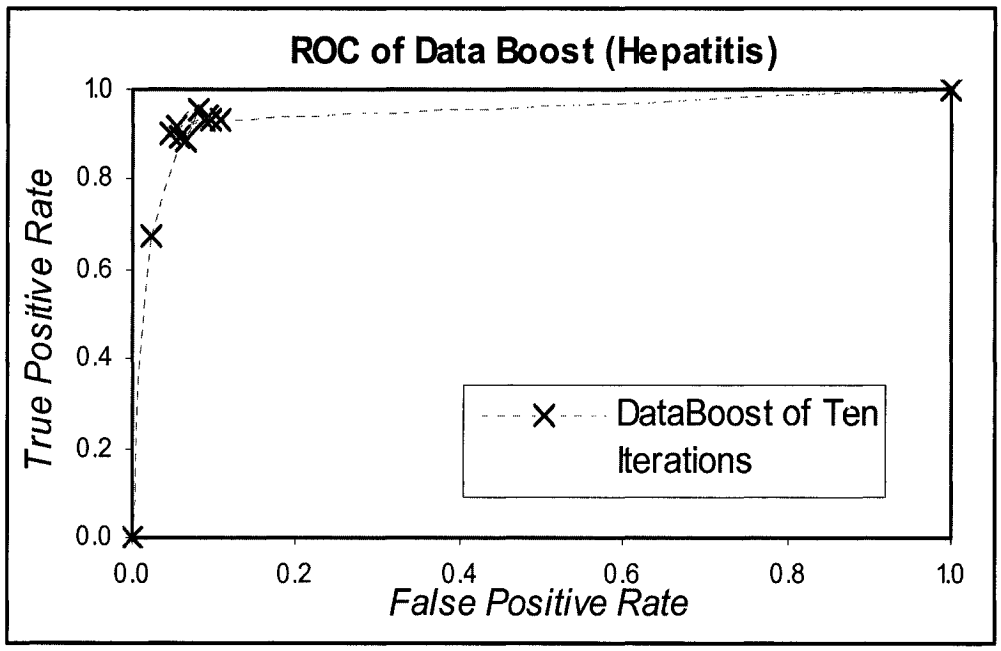


Fig. 21. ROC Curve of ten iteration of the DataBoost-IM algorithm

5.3 Summary

This chapter introduced the DataBoost-IM approach for learning from imbalanced data sets through combining the Boosting method and data generation. In the DataBoost-IM approach, the class frequencies and the total weights against different classes within the ensemble's training set, which consist of both the synthesis and the original training data, are rebalanced during all iterations of the Boosting algorithm.

The DataBoost-IM algorithm was illustrated by means of seventeen data sets with various features, degrees of imbalance and sizes. The results obtained indicate that the DataBoost-IM approach performs well against imbalanced data sets. In particular, the DataBoost-IM algorithm achieved comparable and slightly better predictions, in terms of the *G-mean* and *F-measures metrics*, against both the minority and majority classes, when compared with a component classifier as well

as four other Boosting algorithms. Importantly, the method does not sacrifice one class for the other, but produce high predictive accuracy against both the majority and the minority class.

In conclusion, these results indicate four reasons for the performance improvements achieved by the DataBoost-IM algorithm. The first is that the additional synthetic data provide complementary knowledge for the learning process. The second is that rebalancing the class frequencies alleviates the classifiers' learning bias toward the majority class. The third one is that rebalancing the total weight distribution of different classes forces the Boosting algorithm to focus on the hard examples as well as rare examples. The last one is that the synthetic data prevent Boosting method from over-emphasize the hard examples. This property is especially important when considering the minority class which contains few examples.

The next chapter provides a summary of the thesis and outlines the suggested directions for future research.

CHAPTER SIX

CONCLUSION AND FUTURE WORK

CHAPTER SIX

CONCLUSION AND FUTURE WORK

The underlying principle of the Boosting ensemble states that, by using a weak learning algorithm several times on a sequence of *carefully constructed* training examples, the weak learning algorithm can be converted into an algorithm with a predictive performance which surpasses the original weak algorithm. This study presented two new approaches to extend the Boosting ensemble's capability of *effectively constructing* the sequence of training examples, thus extending the predictive performance thereof.

6.1 Summary

In summary, this thesis began by exploring the nature of the Boosting approach. This is followed by a detailed discussion of the shortcomings of Boosting algorithms, when learning from domains with hard to learn examples and domains with imbalanced class frequencies, was presented. Two techniques, namely the DataBoost and DataBoost-IM algorithms were presented. Also, the thesis described

two sets of experiments, evaluating the proposal approaches using benchmark data sets.

In summary, the main contribution of this study was the introduction of the two above-mentioned approaches, for extending the predictive performance of the Boosting algorithm. The DataBoost approach, which combines data generation and the boosting procedure, was used to extend the predictive performance of Boosting algorithms against domains with hard to learn examples. The DataBoost-IM approach, which integrated data generation and re-balancing process into the boosting procedure, aided the Boosting algorithm to produce high predictions against both minority and majority classes, when learning from domains with imbalanced class frequencies.

Firstly, when applying to domains with hard to learn examples, the Boosting algorithms frequently suffer from over-emphasizing the hard examples, thus decreasing the predictive performance of Boosting algorithms. Also, the knowledge acquired from such hard examples, on which the Boosting algorithms need to concentrate in order to achieve better predictive performance, may be insufficient to improve the overall accuracy of the ensemble. In the DataBoost method, hard examples were identified during each of the iterations of the Boosting algorithm. Subsequently, the hard examples were used to generate synthetic training examples. The synthetic examples were added to the original training set and subsequently used for further training. The thesis presented the results of this approach against ten data sets, using both decision trees and neural networks as base classifiers. The experiments showed promising results when compared with the AdaBoost.M1 approach, in terms of the overall accuracy obtained.

Secondly, when learning from domains with imbalanced class distribution, where the number of examples of the majority class is much higher than that of the minority class, the traditional Boosting algorithms tend to produce high predictive accuracy over the majority class, but poor predictive accuracy over the minority

class. That is due to the fact that few examples from the minority class are much easier to be over-emphasized by the Boosting algorithms, because of the algorithms' extreme interest in hard examples. Also, knowledge from the minority class which contains few examples may not be sufficient to aid the classifiers' learning process. Moreover, the weight updating mechanism of the traditional Boosting algorithm favors the majority class. The DataBoost-IM approach was designed to assist traditional Boosting algorithms to improve their predictive performance on both majority and minority classes when learning from imbalanced data sets. In the DataBoost-IM method, the hard examples from both the majority and minority classes were identified during execution of the Boosting algorithm. Subsequently, the hard examples were used to separately generate synthetic examples for the majority and minority classes independently. The synthetic data were then added to the original training set, and the class distribution and the total weights of the different classes in the new training set were rebalanced. The DataBoost-IM method was evaluated, in terms of the *F-measures, G-mean and overall accuracy*, against seventeen highly and moderately imbalanced data sets using decision trees as base classifiers. The results showed that the DataBoost-IM method compared well in comparison with a base classifier, a standard benchmarking Boosting algorithm and three advanced boosting-based algorithms for imbalanced data set. Results indicated that the proposal approach does not sacrifice one class in favor of the other, but produced high predictions against both minority and majority classes.

6.2 Future work

Future research should address several issues to extend both the DataBoost and DataBoost-IM approaches.

For the DataBoost algorithm, it follows that the approach should be thoroughly tested against a large number of diverse data sets, using a variety of classifiers and ensemble schemes. Future work will thus include the application of this approach to both

artificial and real-world data repositories. The use of other base classifiers and ensemble algorithms and other data generation methods will be further investigated. Furthermore, further work needs to be done on selecting the parameters in the DataBoost algorithm. In particularly, more theoretical discussions are needed to support the choice of parameters. For examples, further investigation is needed to optimize the number of hard examples and the number of synthetic examples. How to choose optimal parameters is a crucial issue in the DataBoost algorithm. On the one hand, we have to ensure that the DataBoost algorithm can, firstly, correctly identify the hard examples which are easy to be over-emphasized and then, secondly, introduce more knowledge about these hard examples. On the other hand, we need to ensure that the DataBoost algorithm is not generating noisy training examples and increasing the computational cost exponentially.

Another issue which needs to address is the computational complexity. The experiments have shown that the DataBoost algorithm needs more time to build the model than the AdaBoost.M1 algorithm. This is due to the fact that the DataBoost algorithm introduces more training examples in each iteration of the boosting procedure. The experiments also indicated that the computation time of building the DataBoost ensemble is reasonably acceptable for small or moderate data sets. However, further investigation of the scalability of the DataBoost ensemble against large and very large data sets is needed.

Finally, work needs to be done to address the disadvantages and some sensitive issues of the DataBoost algorithm. For example, we need to investigate how the DataBoost algorithm performs when noisy data are presented, how the DataBoost approach is effected when class distribution are extremely skew in the original data sets, and how the core idea in the DataBoost algorithm may be applied to other domains in the machine learning and data mining fields.

To extend the DataBoost-IM approach, future research should address some issues, including further investigating the optimal number of new seed examples to generate,

experimenting with other component classifiers and considering the performance against noisy data. A detailed analysis of the margins of the DataBoost and DataBoost-IM algorithms, when compared to other Boosting-based approaches, should prove worthwhile. Also, other weight-assignment methods will be further investigated. The weigh-assignment methods and the number of examples generated in the data generation procedure are crucial to the DataBoost-IM algorithm. On one hand, we need to create synthetic data to compensate the class skew and balance out the dominate weights associating with the hard examples. On the other hard, we should carefully choose the appropriate parameters to ensure proper new knowledge is introduced into the learning process. Future work will also include studying the voting mechanism of the Boosting algorithm using different metrics such as the *F-measures*. Although the DataBoost-IM algorithm and the experiments addressed only two-class problems, it is the opinion of the author that a similar approach can be used within the framework of multi-class learning problems. This will be further investigated. Furthermore, the author is interested in integrating the *"hyper-ROC"* concept into the DataBoost-IM ensembles. In this way, a *ROC* curve may be constructed from a few classifiers. The results of these classifiers as subsequently combined by only considering their "area of expertise". Some classifiers distinguish better at the positive end, others are good at the negative end, and the others are better in the middle of the *ROC* spectrum. When learning from an imbalanced data set, this type of ensemble will be able to dynamically select the best area of expertise of each component classifier when classifying new instances.

REFERENCES

REFERENCES

[1] D. Opitz, R. Maclin (1999). Popular Ensemble Methods: An Empirical Study, Journal of Artificial Intelligence Research, 11, 169-198.

[2] T. Mitchell (1997). Machine Learning, McGraw Hill Companies, Inc.

[3] I. Witten, E. Frank (2000). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, San Francisco, CA, USA.

[4] D. Wolpert (1992). Stacked Generalization, Neural Networks, 5, 241-259.

[5] L. Breiman (1996). Bagging Predictors, Machine learning, 24(2), 123-140.

[6] R. Maclin, D. Opitz (1997). An Empirical Evaluation of Bagging and Boosting, In Proceedings of the 14th National Conference on Artificial Intelligence, Menlo Park, CA: AAAI Press, 546-551.

[7] Y. Freund, R. Schapire (1999). A Short Introduction to Boosting, Journal of Japanese Society for Artificial Intelligence, 14(5), 771-780.

[8] T. Dietterich (1997). Machine Learning Research: Four Current Directions, Artificial Intelligence, 18(4), 97-136.

[9] Y. Freund, R. Schapire (1996). Experiments with a New Boosting Algorithm, the Proceedings of the 13th International Conference on Machine Learning, Morgan Kaufman, 148-156.

[10] L. Breiman (1996). Stacked Regressions, Machine Learning, 24(1), 49-64.

[11] HL Viktor (1999). The CILT Multi-agent Learning System, South African Computer Journal (SACJ), 24, 171-181.

[12] S. Puuronen, V. Terziyan, A. Tsymbal (1999). A Dynamic Integration Algorithm for an Ensemble of Classifiers, Foundations of Intelligent Systems: 11th International Symposium ISMIS'99. Lecture Notes in Artificial Intelligence, Vol. 1609, Warsaw, Poland, Springer-Verlag, 592-600.

[13] J. R. Quinlan (1996). Bagging, Boosting, and C4.5, In Proceedings of the 13th National Conference on Artificial Intelligence, Menlo Park, CA: AAAI Press, 725-730.

[14] G. Ridgeway (1999). The State of Boosting, Computing Science and Statistics, Vol. 31, 172-181.

[15] R. Schapire (1999). A Brief Introduction to Boosting, In Thomas Dean, editor, 16th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1401-1406.

[16] L. Breiman (1996). Bias, Variance, and Arcing Classifiers, Technical Report 460, Statistics Department, UC-Berkeley, Berkeley, CA,USA.

[17] R. Schapire, Y.Freund, P. Bartlett, W. Lee (1997). Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods, In Proceedings of the 14th International Conference on Machine Learning, Morgan Kaufmann, 322-330.

[18] Y. Freund, R. Schapire (1997). A Decision-theoretic Generalization of On-line Learning and An Application to Boosting, Journal of Computer and System Sciences, 55(1), 119-139.

[19] J. Friedman, T. Hastie, R. Tibshirani (2000). Additive Logistic Regression: A Statistical View of Boosting, the Annals of Statistics, 28, 337 – 374.

[20] J.R. Quinlan (1994). C4.5: Programs for Machine Learning, Morgan Kaufmann, CA, USA.

[21] L. Hansen, P. Salamon (1990). Neural Network Ensembles, IEEE Transactions on Pattern Analysis and Machine Intelligence, 12, 993-1001.

[23] A. Grove, D. Schuurmans (1998). Boosting In the Limit: Maximizing the Margin of Learned Ensembles, In Proceedings of the 15th National Conference on Artificial.

[24] R. Schapire (1990). The Strength of Weak Learnability, Machine Learning, 5(2), 197-227.

[25] R. Meir, G. Ratsch (2002). An Introduction to Boosting and Leveraging, Advanced Lectures on Machine Learning, 118-183.

[26] C.L. Blake, C. J. Merz (1998). UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/~mlearn/MLRepository.html], Department of Information and Computer Science, University of California, Irvine, CA, USA.

[27] V.N. Vapnik (1982). Estimation of Dependences Based on Empirical Data, Springer-Verlag, New York.

[28] B.E. Boser, I. M. Guyon, V.N. Vapnik (1992). A Training Algorithm for Optimal Margin Classifiers, In Proceedings of the 15th Annual ACM Workshop on Computational Learning Theory, 144-152.

[29] T.G. Dietterich (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization, Machine Learning, 40, 139-157.

[30] A. Krieger, A. J. Wyner, C. Long (2001). Boosting Noisy Data. In C. E. Brodley and A. P. Danyluk (eds.), Proceedings of the 18th International Conference on Machine Learning (ICML-2001), Williamstown, MA, Morgan Kaufmann, 274-281.

[31] R.A. McDonald, D.J. Hand, I.A. Eckley (2003). An Empirical Comparison of Three Boosting Algorithms on Real Data Sets with Artificial Class Noise, In Proceedings of Multiple Classifier Systems Workshop 2003, Springer-Verlag.

[32] Y. Freund (1999). An Adaptive Version of the Boost by Majority Algorithm, In Proceedings of the 12th Annual Conference on Computational Learning Theory.

[33] N. Japkowicz (2000). Learning from Imbalanced Data Sets: A Comparison of Various Strategies, Learning from Imbalanced Data Sets: Papers from the AAAI Workshop, Menlo Park, CA, AAAI Press, Technical Report WS-00-05, 10-15.

[34] N. Chawla, K. Bowyer, L. Hall, W. Kegelmeyer (2002). SMOTE: Synthetic Minority Over-sampling Technique, Journal of Artificial Intelligence Research, 16, 321-357.

[35] D. Lewis, J. Catlett (1994). Heterogeneous Uncertainty Sampling for Supervised Learning, Proceedings of the 11th International Conference on Machine Learning, ICML'94, New Brunswick, New Jersey: USA, Morgan Kaufmann, 148-156.

[36] M.A. Maloof (2003). Learning when Data Sets are Imbalanced and when Costs are Unequal and Unknown, ICML-2003 Workshop on Learning from Imbalanced Data Sets II.

[37] G. Weiss, F. Provost (2001). The Effect of Class Distribution on Classifier Learning: An Empirical Study, Technical Report ML-TR 43, Department of Computer Science, Rutgers University.

[38] C. Ling, C. Li (1998). Data Mining for Direct Marketing: Problems and Solutions, Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD '98), Menlo Park, CA: AAAI Press, 73-79.

[39] N. Japkowicz (2000). The Class Imbalance Problem: Significance and Strategies, In the Proceedings of the 2000 International Conference on Artificial Intelligence (IC-AI'2000): Special Track on Inductive Learning.

[40] M. Kubat, S. Matwin (1997). Addressing the Curse of Imbalanced Training Sets: One-Sided Selection, Proceedings of the 14th International Conference on Machine Learning. San Francisco, CA: Morgan Kaufmann, 179-186.

[41] C. Drummond, R. Holte (2000). Exploiting the Cost (In)sensitivity of Decision Tree Splitting Criteria, In Proceedings of the 17th International Conference on Machine Learning, 239-246.

[42] P. Chan, S. Stolfo (1998). Toward Scalable Learning with Non-uniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection, In Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, Menlo Park, CA: AAAI Press, 164-168.

[44] M. Joshi, V. Kumar, R. Agarwal (2001). Evaluating Boosting Algorithms to Classify Rare Classes: Comparison and Improvements, First IEEE International Conference on Data Mining, San Jose, CA.

[45] M. Joshi, V. Kumar, R. Agarwal (2002). Predicting Rare Classes: Can Boosting Make Any Weak Learner Strong, Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2002), 297-306.

[46] N. Chawla, A. Lazarevic, L. Hall, K. Bowyer (2003). SMOTEBoost: Improving Prediction of the Minority Class in Boosting, 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Cavtat-Dubrovnik, Croatia , 107-119.

[47] K. Ting (2000). A Comparative Study of Cost-sensitive Boosting Algorithms, Proceedings of 17th International Conference on Machine Learning, Stanford, CA, 983-990.

[48] W. Fan, S. Stolfo, J.Zhang, P. Chan (1999). AdaCost: Misclassification Cost-Sensitive Boosting, Proceedings of 16th International Conference on Machine Learning, Slovenia.

[49] G. Karakoulas, J. Shawe-Taylor (1999). Optimizing Classifiers for Imbalanced Training Sets. In M. Kearns, S. Solla, and D. Cohn, editors. Advances in Neural Information Processing Systems 11, MIT Press.

[50] T.M. Ha, M.Zimmermann, H. Bunke (1998). Off-line Handwritten Numeral String Recognition by Combining Segmentation-based and Segmentation-free Methods, Pattern Recognition, Vol. 31, No. 3, 257-272.

[51] F. Provost, T. Fawcett (1997). Analysis and Visualization of Classifier Performance: Comparison Under Imprecise Class and Cost Distributions, In Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, Newport Beach, CA, USA, 43-48.

[52] C. Metz (1989). Some Practical Issues of Experimental Design and Data Analysis in Radiological ROC Studies, Investigative Radiology, 24, 234-245.

[53] M. Kubat, R. Holte, S. Matwin (1998). Machine Learning for the Detection of Oil Spills in Satellite Radar Images, Machine Learning, 30, 195–215.

[54] J. Swets (1988). Measuring the Accuracy of Diagnostic Systems, Science, 240, 1285–1293.

[55] F. Provost, T. Fawcett, R. Kohavi (1998). The Case against Accuracy Estimation for Comparing Induction Algorithms, Proceedings of the 15th International Conference on Machine Learning, San Francisco, CA: Morgan Kaufmann, 445-453.

[56] C. J. van Rijsbergen (1979). Information Retrieval. Butterworths, London.

[57] HL Viktor, I. Skrypnik (2001). Improving the Competency of Ensembles of Classifiers through Data Generation, ICANNGA'2001, Prague: Czech Republic, April 21-25, 59-62.

[58] SB Thrun et al (1991). The Monk's problems: A Performance Comparison of Different Learning Algorithms. Technical Report CMU-CS-91-17. Computer Science Department, Carnegie Mellon University, Pittsburgh: USA.

[59] H. Schwenk, Y. Bengio (1997). AdaBoosting Neural Networks: Application to On-line Character Recognition, In Proceedings ICANN'97, Volume 1327 of LNCS, Springer-Verlag, Berlin, 969-972.

[60] D. Skalak (1997). Combining Nearest Neighbor Classifiers. Ph.D. Thesis, Department of Computer Science, University of Massachusetts, Amherst, MA, USA.

[61] C. Drummond, R. Holte (2003). C4.5, Class Imbalance, and Cost Sensitivity: Why Under-sampling Beats Over-sampling, Workshop on Learning from Imbalanced Data Sets II Held in Conjunction With ICML'2003.

[62] K. Ting, Z. Zheng (1998). Boosting Cost-sensitive Trees, Proceedings of the 1st International Conference on Discovery Science, LNAI-1532, Springer-Verlag, Berlin, 244-255.

[63] L. Breiman (1997). Prediction Games and Arcing Algorithms, Technical Report 504, December 1997, Statistics Department, University of California, Berkeley: USA.

[64] F. Provost (2000). Learning with Imbalanced Data Sets 101. The AAAI'2000 Workshop on Imbalanced Data Sets.

[65] R. Servedio (2003). Smooth Boosting and Learning with Malicious Noise. Journal of Machine Learning Research 4, 633-648.

[66] C. Domingo, O. Watanabe (2000). Madaboost: A Modified Version of AdaBoost, In Proceedings of the 13th Annual Conference on Computational Learning Theory, 180-189.

[67] T. Jo, N. Japkowicz (2004). Class Imbalances versus Small Disjuncts, SIGKDD Explorations 6(1), 2004, 40-49.

[68] N. Japkowicz (2002). Learning from Imbalanced Data Sets, a Comparison of Various Strategies, In the Proceedings of Learning from Imbalanced Data Sets, Papers from the AAAI Workshop, Technical Report WS-00-05, 10-15.

[69] N. Japkowicz, C. Myers, M. Gluck (1995). A Novelty Detection Approach to Classification, In the Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), 518-523.

[70] C. Cardie, N. Howe (1997). Improving Minority Class Prediction Using Case-specific Feature Weights, Proceedings of the 14th International Conference on Machine Learning. San Francisco, CA, Morgan Kaufmann, 57-65.

[71] T.G. Dietterich (1988). Approximate statistical tests for comparing supervised classification learning algorithms. Neural Computation, 10(7):1895-1924.

APPENDIX A
PAPERS BASED ON THIS THESIS

Paper in Refereed Journals:

H. Guo and HL Viktor, Learning from Imbalanced Data Sets with Boosting and Data Generation: The DataBoost-IM Approach, ACM SIGKDD Explorations, 6(1), 2004, 30-39.

Papers in Refereed Conference Proceedings:

H. Guo and HL Viktor, Boosting with data generation: Improving the Classification of Hard to Learn Examples, the 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE), Ottawa, Canada, May 17-20, 2004. Lecture Notes in Artificial Intelligence (LNAI) 3029, Springer-Verlag, Berlin, 1082-1091.

HL Viktor and H. Guo, Improving the Prediction of Multiple Classifiers against Imbalanced Datasets through Adding Synthetic Examples, to be presented at the 10th International Workshop on Statistical Pattern Recognition, Lisbon, Portugal, August 18-20, 2004.