Imbalance Learning Using Heterogeneous Ensembles

Hossein Ghaderi Zefrehi

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
September 2018
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Assoc. Prof. Dr. Ali Hakan Ulusoy
Acting Director

I certify that this thesis satisfies the requirements of thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Hadi Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Hakan Altınçay
Supervisor

Examining Committee

1. Prof. Dr. Hakan Altınçay
2. Prof. Dr. Hasan Kömürcügil
3. Asst. Prof. Dr. Nazife Dimililer
4. Asst. Prof. Dr. Zafer Erenel
5. Asst. Prof. Dr. Ahmet Ünveren

ABSTRACT

In pattern classification, class-imbalance problem occurs when the number of samples in one of the classes is much larger than those in the others. In such cases, the performance of classifiers is generally poor on the minority class. Ensembles of classifiers are used to tackle this problem where each member is developed using a different balanced dataset. In this approach, one balancing strategy and a classifier prototype is generally used. In order to increase the diversity among the members, bagging and boosting are also considered. In this thesis, the use of heterogeneous ensembles utilizing multiple prototypes and multiple balancing schemes for imbalance learning is addressed. Experiments conducted on 66 datasets have shown that significant improvements can be achieved by employing multiple prototypes. It is also observed that multiple balancing schemes contribute to the performance scores, especially in simple and bagging-based ensembles.

Keywords: imbalance learning, classifier ensembles, bagging, boosting, heterogeneous ensembles, multi-balancing

ÖZ

Örüntü tanımada, bir sınıftaki örnek sayısı diğer sınıflarınkinden çok daha fazla olduğunda sınıf-denksizliği problem oluşmaktadır. Bu tür durumlarda, sınıflandırıcı başarımı kıüçük sınflarda düşük olmaktadır. Bu problemi aşmak için, her üyenin denkleştirilmiş bir veri kümesi ile eğitildiği çoğul sınıflandırıcılı sistemler kullanılmaktadır. Bu sistemler, genellikle bir denkleştirme ve bir sınıflandırıcı tipi ile geliştirilmektedir. Sınıflandırıcılar arasındaki farklılıkları artırmak için, torbalama ve artırma teknikleri de kullanılmaktadır. Bu tezde, birden fazla denkleştirme ve sınıflandırcı tipi kullanan heterojen çoğul sınıflandırıcı sistemlerin denksizlik öğrenmede kullanımı çalışılmıştır. 66 veri kümesinde yapılan deneysel çalışmalar, birden fazla sınıflandırıcı tipi kullanılarak başarımda belirgin iyileştirmeler sağlanabileceğini göstermiştir. Ayrıca, birden fazla denkleştirme algoritmasının kullanılmasının, özellikle basit ve torbalama-tabanlı sistemlerin başarımına katkıda bulunduğu gözlemlenmiştir.

Anahtar Kelimeler: Denksizlik öğrenme, çoğul sınıflandırıcılı sistemler, torbalama, artırma, heterojen çoklu sınıflandırıcılı sistemler, çoklu-denkleştirme

Dedicated to:

My dear daughters, my patient wife, and my kind parents who tolerated my absence for three years.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to my supervisor Prof. Dr. Hakan Altınçay for continuous support during my thesis study and research, for his patience, enthusiasm, and immense knowledge.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Introduction

Pattern classification plays a critical role in various domains including medicine, computer vision, text categorization for effective search of electronic documents, analysis of gene expression arrays, combinatorial chemistry, high-value customer prediction, and disease pre-diagnosis. Typical problems that are generally observed in solving machine learning tasks are noise in the data, large numbers of features and class-imbalance. Many researchers are actively working on these problems and numerous solutions are already proposed.

In the case of imbalanced datasets, the number of samples for the target class (also called the positive class) is generally small. Consider a two-class classification problem where the target class is the minority, having much smaller number of samples than the other class (majority or negative class). In such cases, almost all algorithms tend to classify the test samples as the majority class, making large classification errors on the minority samples. In order to deal with this problem, in general, the data is balanced before training a classifier. Undersampling the majority class and oversampling the minority are two widely used techniques for this purpose.

1.2 Significance

When a classifier is generated using an imbalanced dataset, since accuracy is the most widely-used measure for parameter tuning, the decision boundaries will be computed in such a way that the samples in the majority class will be classified with high accuracy [1]. Such classifiers are not useful in practice since the cost of misclassifying a minority sample is generally much more than misclassifying a majority sample. For example, consider a disease classification problem where the number of patients who has the disease is less than those who do not, which occurs naturally. When a conventional classifier is used, it is highly likely that majority of patients will be misclassified as healthy. In fact, true diagnosis of a positive cancer is more important than a false alarm due to misclassifying a healthy person. Therefore, building a classifier that provides expected level of performance on the target class when the data is imbalanced attracts the interest of many researchers.

1.3 Aim of Study

In majority of the studies conducted on imbalance learning, the ultimate goal is set as balancing the dataset before training a classifier. This is generally done by removing some instances from majority class and/or adding some new instances to the minority class. In random undersampling, some randomly selected samples are discarded from the majority class. On the other hand, in oversampling approach, the number of minority instances is increased by duplicating randomly selected minority instances until the desired number of samples is obtained. Experimental results conducted on imbalanced datasets originating from different domains have shown that the highest performance scores are generally achieved using an ensemble of classifiers, where each member of the ensemble is trained on a different randomly-generated dataset [2].

In this thesis, taking into account the fact that effectiveness of a balancing scheme depends on the number of training samples in different classes [4] and, the possibility of obtaining more diverse members especially in cases of small minority classes by employing multiple base learners, multi-balancing and multi-prototype ensembling for imbalance learning is addressed. To explore the effectiveness of utilizing multiple balancing schemes and multiple prototypes, different subsets of prototype and balancing schemes are evaluated in a systematic procedure. For each of the a priori selected prototype and balancing scheme, the performance scores achieved using simple, bagging- and boosting-based ensembles are firstly computed. For each balancing scheme, by studying different prototype subsets having different cardinalities, the effectiveness of using heterogeneous ensembles is then explored. Similarly, for each prototype, utilizing different subsets of data balancing techniques is addressed for comparative evaluation of employing single or multiple balancing techniques in generating powerful ensembles. Ensembling multi-balancing and multi-prototype members are then investigated. The area under the receiver operating characteristics curve (AUC) is used as the performance metric in all simulations. Experiments conducted on 66 datasets in KEEL repository [35] have demonstrated that heterogeneous ensembles achieve significantly better performance scores than their homogeneous counterparts. The experimental results also support using multiple balancing techniques.

The rest of the thesis is organized as follows. Chapter 2 presents a brief literature review. In Chapter 3, the proposed ensembling framework utilized for investigating heterogeneous and multi-balancing ensembles is explained. Chapter 4 presents the experiments conducted and the results obtained. The conclusions of this thesis and future directions of research to deal with the imbalance problem are presented in Chapter 5.

Chapter 2

A BRIEF LITERATURE REVIEW

2.1 Introduction

The class-imbalance problem arises when the number of samples belonging to one class is much more than that of another class [1, 5]. In two-class classification problems, the positive class is in general the minority, including smaller number of samples than the negative (majority) class. Since the classifiers are generally tuned to minimize the number of misclassified training samples, the classification accuracy is higher for the majority class. However, the performance on the minority class may be of primary concern. Because of this, the models obtained using imbalanced data are generally unacceptable in practice.

The class-imbalance problem is extensively studied and numerous techniques have been proposed [2, 3, 4, 6, 7, 8]. These efforts can be broadly categorized into three groups, namely algorithm level, data level and ensemble learning approaches [2, 9, 10, 11]. In the algorithm level approach which also includes cost-sensitive methods, the main aim is to develop algorithms that mainly focus on the minority class [12, 13, 14, 15]. In the data level approach, the data is balanced using a preprocessing technique [18]. Ensemble learning is the most widely used approach, aiming to utilize algorithm level or data level techniques together with resampling approaches such as bagging and boosting [1, 2, 9, 23].

2.2 Algorithm Level Approaches

The algorithms in this group are based on adapting current classifiers so that they focus more on the minority class [1]. In other words, by making some modifications in the algorithm such as assigning a large penalty/cost for misclassification of the minority class, the algorithm is enforced to correctly classify the positive samples. For example, Hellinger Distance Decision Tree (HDDT) takes into account the size of the classes [9]. On the other hand, Class Confidence Proportion Decision Tree (CCPDT) is insensitive to the class distribution [13]. Similarly, large penalties can be assigned to misclassification of the minority class in SVM classifier [17]. Cost-sensitive decision trees [16] and cost-sensitive neural networks [18] also assign a different cost to each class. Another well-known approach is the cost-sensitive variant of AdaBoost, namely AdaCost [24].

2.3 Data Level Approaches

This group includes the most widely-used techniques, namely undersampling and oversampling [2]. In random undersampling approach, the number of negative samples is reduced by selecting a random subset of samples. Selection can be done with replacement or without replacement. When replacement is considered, a negative sample can appear in the target dataset more than once. When replacement is not allowed, any sample can appear only once in the final dataset.

In random oversampling, minority samples are duplicated until the desired number of samples is obtained. Duplications are done by replacement. In some cases the original minority sample will be kept and desired number of samples are randomly selected from original minority class and added to the dataset.

As more intelligent oversampling schemes, SMOTE [19] and its variants such as DBSMOTE [20], ANS [21] and ADASYN [22] are also proposed where, additional samples are generated by interpolating randomly selected pairs of the samples in the minority class. The main idea of SMOTE is to balance the dataset by creating new minority instances. For each sample in the minority class, SMOTE finds K nearest neighbors with the same label. Then, two of these neighbors are randomly selected. A point that is lying on the segment joining these samples is randomly selected as a new minority instance and it is be added to the dataset. The algorithm is repeated until the desired numbers of minority instances are obtained. In DBSMOTE [20], for generating a new positive instance, the segment between a randomly selected minority sample and the center of the minority class is considered. ANS [21] automatically determines the number of neighbors and adapts it for different regions of minority samples. In this algorithm, the minority instances that have no neighbor around themselves are labeled as outcast and they are not utilized in sample creation process. ADASYN [22] employs a weighting strategy to assign a weight to each minority sample by considering its difficulty in learning process. Consequently, for hard (difficult to learn) minority samples, more new instances will be created.

2.4 Ensemble Learning Approaches

In cost-sensitive ensembles, multiple cost-sensitive classifiers are utilized where each classifier is developed using a resampled dataset [24, 25]. On the other hand, in SMOTEBagging [2, 26], UnderBagging [2, 26] and OverBagging [26, 5], both bagging and balancing are considered in forming the training sets of ensemble members. In boosting based ensembles such as SMOTEBoost [27], RUSBoost [28] and DataBoost-IM [23], after balancing the data, the weights of samples are utilized in computing the training set of the following member. It should be noted that, in bagging-based ensembles, the training data of each member is independent of the previous members whereas, in boosting-based ones, each member is trained on the samples which are generally misclassified by the previous members. More than one balancing scheme may also be used. For instance, SMOTEBagging technique uses both SMOTE and random oversampling. As an alternative approach, simple ensembles are also studied [9]. In this technique, resampling is omitted where each member is generated using a randomly balanced dataset. In other words, either the minority class is oversampled or the majority class is undersampled. In almost all ensemble-based methods, a single classifier prototype is used. In fact, the use of so-called homogeneous ensembles highly dominates the efforts in learning imbalanced datasets [4].

2.5 Classifier Prototypes

SVM, decision trees, neural networks, logistic regression and KNN are among the most frequently used classifiers to generate ensemble members [4]. In this section, the classifiers utilized in the thesis are briefly described.

2.5.1 RIMARC

RIMARC [37] is a supervised classification method which ranks instances by assigning higher probabilities to samples in the positive class. It is originally developed for binary classification. The main constraint of RIMARC is that all features should be categorical. When this is not the case, RIMARC applies MAD2C algorithm to discretize the features. During the discretization process, the main objective of MAD2C is to keep the AUC for that feature as large as possible. After converting all features to categorical form, RIMARC is used to compute the scores for every interval of all features. This score is calculated as the ratio of positive samples to total number of instances that are in the same interval as:

$$S_i = \frac{P_i}{(P_i+N_i)}, i = 1,2,\ldots,C \quad (C \text{ is number of intervals}) \tag{1}$$

where P_i and N_i are the number of positive and negative lables in ith interval respectively. Then, for each feature a weight will be generated. This weight is based on the AUC of the corresponding feature that is computed after discretization. The weight is calculated as:

$$W_i = 2 * (AUC_i - 0.5) \tag{2}$$

Finally, for each test sample, depending on the intervals of the feature values, the overall rank score is computed as:

$$R = \frac{\sum_{i=1}^{d}(S_i * W_i)}{\sum_{i=1}^{d} W_i}) \tag{3}$$

where d denotes the total number of features.

2.5.2 Logistic Regression (LR)

The binary logistic regression model computes the probability of a binary target variable as a function of one or multiple features, x_i. The relationship between the probability of a particular class and the features is represented as:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_3 x_d.) \tag{4}$$

where p is the probability that the label of the sample is “positive”. The expression on the left-hand side of the equation given above is called logit, whereas $(\frac{p}{1-p})$ is named as odds. After some manipulations, p can be expressed as:

$$p = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_d x_d}}{1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\cdots+\beta_d x_d}}. \tag{5}$$

In practice, the coefficients of the features denoted by β_i are generally computed using maximum likelihood estimation.

2.5.3 Support Vector Machine (SVM)

SVM builds a hyperplane in the instance space which maximizes the margin that is defined as the gap between the samples of different classes. By utilizing a kernel function, the feature vectors can also be mapped into a higher dimensional space where a linear classifier developed in that space will correspond to a nonlinear one in the original space. Second order polynomial and radial basis function are two widely-used kernels in pattern classification. In this thesis, we employed radial basis function as the mapping kernel.

2.5.4 KNN

KNN is one of the simplest classification algorithms. For any test sample, K nearest neighbors are found where K is generally selected as an odd number. In computing the neighbors, by taking into account a predefined measure, the distances between the test sample and all samples in the training set are firstly calculated. Then, K samples having smallest distance scores are selected as neighbors. By voting over class labels of these neighbors, the label of the tested sample is determined.

2.5.5 J48

Decision trees are widely used classification schemes due to their simplicity and interpretability of the models generated. The models are in the form of rule-sets. They can handle various types of features at the same time. Each rule corresponds to a different route from the root to a leaf of the tree. Various decision trees are studied, each having a different feature selection or pruning strategy. J48 is a widely used tree that is available in Weka platform which implements the C4.5 algorithm. C4.5 uses normalized information gain to choose the best feature for each node. It applies post-pruning to eliminate the redundant branches.

Chapter 3

METHODOLOGY

3.1 Introduction

It is well known that a key factor in developing a good ensemble is the diversity of the member classifiers where, two classifiers are defined to be diverse if their errors do not overlap [29, 30]. In order to build a diverse ensemble, as described Chapter 2, a different training set is generally used for each member by using bagging- or boosting-based resampling together with random balancing. Since the performance on minority class is the primary concern, diversity originating from differences in the minority samples is highly crucial. However, only a few minority samples might be available in some classification problems. Increasing the number of minority samples by using an oversampling-based technique may not help to generate different diverse training sets in these domains. For example, all sets of five hundred samples obtained using random oversampling of only ten samples is expected to have around fifty duplicates of each sample. In such cases, the majority class has a higher potential to be the main source of diversity. However, the members trained on undersampled majority class may still have overlapping errors on the minority class [31]. Experiments conducted on imbalance learning have shown that SMOTE is a more effective approach when the minority class has small number of samples [4]. Moreover, the relative performance of different balancing schemes is observed to be dependent on the total number of training samples and imbalance ratio as well.

As an alternative source of diversity to random balancing, using multiple learners (prototypes) in generating member classifiers should also be taken into consideration in imbalanced learning. Although multi-prototype (heterogeneous) ensembles have been verified to perform consistently better than its members in various domains [32, 33, 34], the aforementioned schemes developed for imbalance learning are based on a single prototype that is in general a decision tree [4, 5]. To the best of our knowledge, the effectiveness of heterogeneous ensembles in imbalance learning is not fully investigated.

Let R denote a set of balancing techniques and T represent a set of classifier prototypes. In simple, bagging- and boosting-based ensemble approaches, all members are generally built by utilizing one a priori selected 2-tuple ($r_i \in$ R, $t_i \in$T) that is a member of the Cartesian product, R × T. In these ensembles, the main sources of the diversity between different members are the randomness in the balancing scheme, r_i and the differences in the training sets. However, as a more general approach, multiple 2-tuples can be considered in forming the ensemble. Depending on the 2-tuples selected, the members may be generated using (1) a single balancing scheme but multiple prototypes (2) multiple balancing schemes but a single prototype and (3) multiple balancing schemes and multiple prototypes. When compared with the classifiers obtained using a single 2-tuple, it is expected to achieve a more diverse set of ensemble members when multiple 2-tuples are employed. Table 1 illustrates 2-tuples for |R| = |T| = 5. When five balancing techniques, i.e. R = {r_1, r_2, r_3, r_4, r_5} and five prototypes, T = {t_1, t_2, t_3, t_4, t_5} are considered, 25 different ensembles can be built, one for each 2-tuple. Table 1 presents examplar setting. For instance in part (a), four 2-tuples are selected where, three balancing schemes and three prototypes are considered. Assume that K members will be generated for each 2-tuple. In such a case, using the 2-tuples (r_2, t_2), (r_2, t_4), (r_3, t_3) and (r_4, t_2), an ensemble of 4×K members will be computed. In part (b), 10 2-tuples in the Cartesian product of R and $\{t_2, t_4\}$ (i.e. $R \times \{t_2, t_4\}$) are considered, leading to 10×K members. Similarly, 10×K members are generated using all 2-tuples in $\{r_2, r_3\}\times T$ in part (c). In all three settings, diversities among the members is expected to increase due to utilizing more than one classifier prototype and balancing technique.

Table 1: The multi-prototype and multi-balancing ensemble architectures evaluated in this thesis.

	t_1	t_2	t_3	t_4	t_5
r_1					
r_2		*		*	
r_3			*		
r_4		*			
r_5					

(a)

	t_1	t_2	t_3	t_4	t_5
r_1		*		*	
r_2		*		*	
r_3		*		*	
r_4		*		*	
r_5		*		*	

(b)

	t_1	t_2	t_3	t_4	t_5
r_1					
r_2	*	*	*	*	*
r_3	*	*	*	*	*
r_4					
r_5					

(c)

In this thesis, to investigate the effectiveness of utilizing multiple prototypes and multiple balancing schemes, simple, bagging- and boosting-based ensembles of multiple 2-tuples is extensively studied. The experiments are organized in two major parts. In the first part, the relative performances of different 2-tuple sets is investigated. For each a priori selected 2-tuple, K members are generated in parallel. The members corresponding to each 2-tuple are generated independently from the members of the other 2-tuples. Then, members from different 2-tuples are combined using averaging. In this part of experiments, various subsets of 2-tuples are also evaluated. In the following context, this type of ensembles is referred as parallel (P). In the second part of the experiments, random selection of a 2-tuple for each member by using the weights assigned to each prototype in T is addressed. Weights of better classifiers are assigned as larger than worse ones to have them more frequently selected. Consequently, the number of members for each 2-tuple will not be the same. In this approach, since member generation is iteratively done, this ensemble architecture will be referred as serial (S) in the following context. As in the case of parallel approach, the members are combined using averaging. The architectures of multi-balancing and multi-prototype ensembles are described in more detail in the following subsections.

3.2 Generation of Parallel Ensembles

Let U denote the original imbalanced dataset. Assume that a single balancing scheme denoted by r_i and a classifier prototype, t_k is utilized. Consider a simple ensemble of K members where the kth member is generated by employing a different balanced dataset, B_k. In other words, r_i is applied K times to generate K balanced training sets. Then, the members are combined using averaging. Part (a) in Figure 1 shows the flowchart of this ensemble that is referred as r_i*t_k*Simple in the following context. It should be noted that the symbol '*' does not denote any operator. It is used to separate the design components of ensembles in their naming.

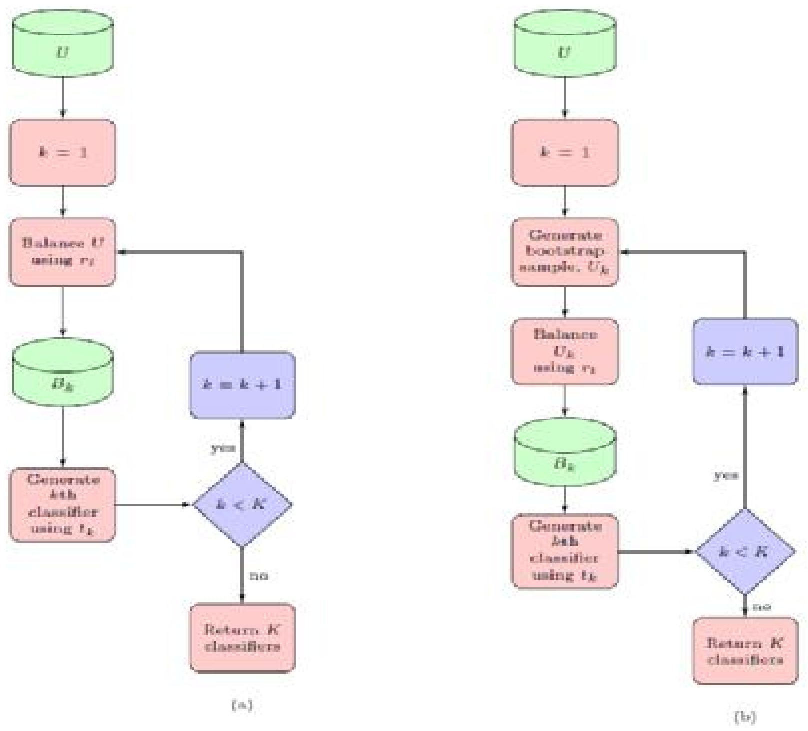
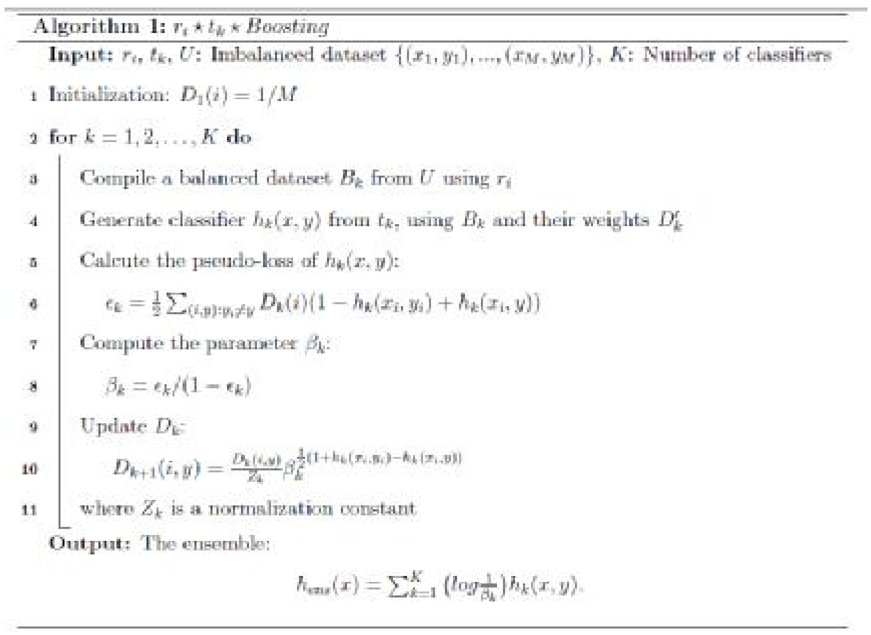


Figure 1: Generation of the ensembles (a) r_i*t_k*Simple and (b) r_i*t_k*Bagging that are based on using a single balancing scheme, r_i and a single prototype, t_k.

Part (b) in Figure 1 illustrates the flowchart of bagging-based ensemble that is referred as r_i*t_k*Bagging in the following context. In this approach, the training set of each member is computed by applying the balancing scheme to a bootstrap sample of the original dataset. As in the case of simple ensembles, the members are combined using averaging.

The algorithm used for developing boosting-based ensembles denoted by r_i*t_k*Boosting for an a priori selected balancing scheme r_i and prototype t_k is presented in Algorithm 1. The algorithm is obtained by adding data balancing (in Line 3) to Adaboost.M2 [36]. In fact, when r_i is selected as SMOTE, the algorithm corresponds to SMOTEBoost. Similarly, when r_i is undersampling, it is equivalent to RUSBoost.



```
Algorithm 1: $r_i \star t_k \star Boosting$
  Input: $r_i$, $t_k$, $U$: Imbalanced dataset $\{(x_1, y_1), ..., (x_M, y_M)\}$, $K$: Number of classifiers
  Initialization: $D_1(i) = 1/M$
  for $k = 1, 2, \ldots, K$ do
      Compile a balanced dataset $B_k$ from $U$ using $r_i$
      Generate classifier $h_k(x, y)$ from $t_k$, using $B_k$ and their weights $D_k'$
      Calcute the pseudo-loss of $h_k(x, y)$:
          $\epsilon_k = \frac{1}{2}\sum_{(i,y):y_i \neq y} D_k(i)(1 - h_k(x_i, y_i) + h_k(x_i, y))$
      Compute the parameter $\beta_k$:
          $\beta_k = \epsilon_k/(1 - \epsilon_k)$
      Update $D_k$:
          $D_{k+1}(i, y) = \frac{D_k(i,y)}{Z_k}\beta_k^{\frac{1}{2}(1+h_k(x_i,y_i)-h_k(x_i,y))}$
      where $Z_k$ is a normalization constant
  Output: The ensemble:
          $h_{ens}(x) = \sum_{k=1}^{K} (log\frac{1}{\beta_k})h_k(x, y)$.
```

In this thesis, the main goal is to investigate combination of classifiers generated using multiple balancing schemes and prototypes. This can be achieved using r_i*t_k*Simple, r_i*t_k*Bagging and r_i*t_k*Boosting and multiple 2-tuples. For this purpose, employing either the whole set of R or T is firstly addressed. Specifically, r_i×T*Simple*P corresponds to combination of 5×K members, where r_i is used in all members as shown in Figure 2.

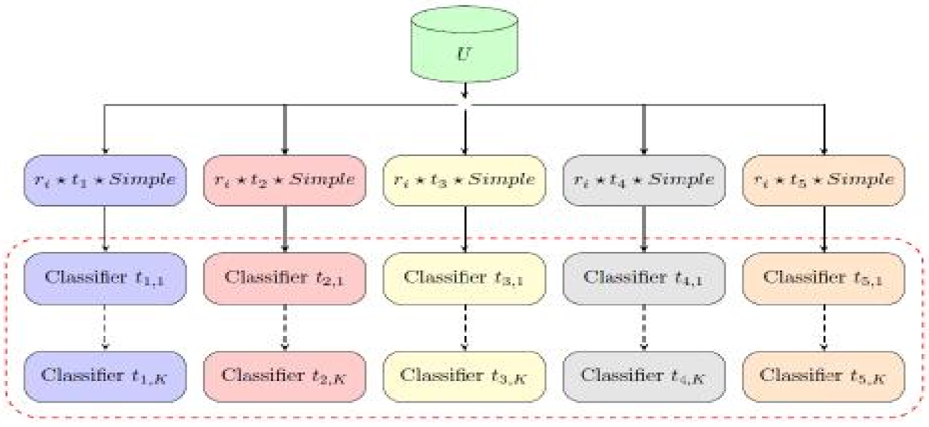
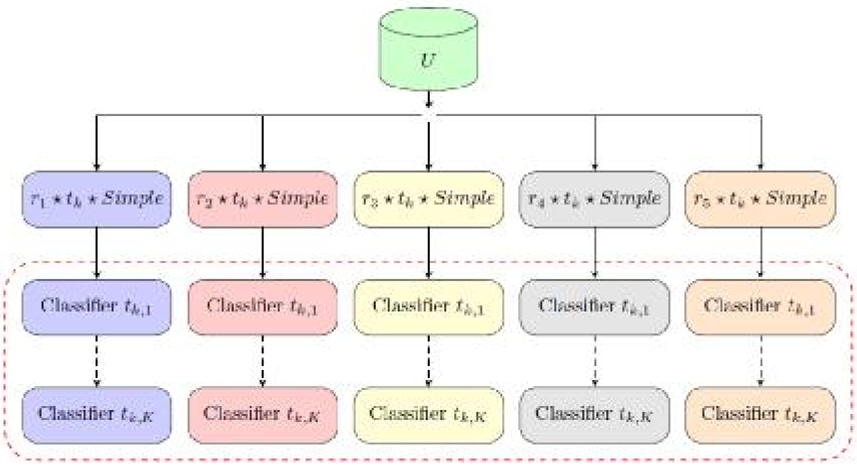


Figure 2: Generation of r_i×T*Simple*P that is based on using a single balancing scheme, r_i and the whole set of all prototypes. The ensemble corresponds to the classifiers in the dashed-line box.

In this setting, K members are generated for each prototype. The label “P” denotes parallel generation of members for different prototypes. Similarly, the ensembles r_i×T*Bagging*P and r_i×T*Boosting*P can be computed by replacing r_i*t_k*Simple with r_i*t_k*Bagging and r_i*t_k*Boosting, respectively.

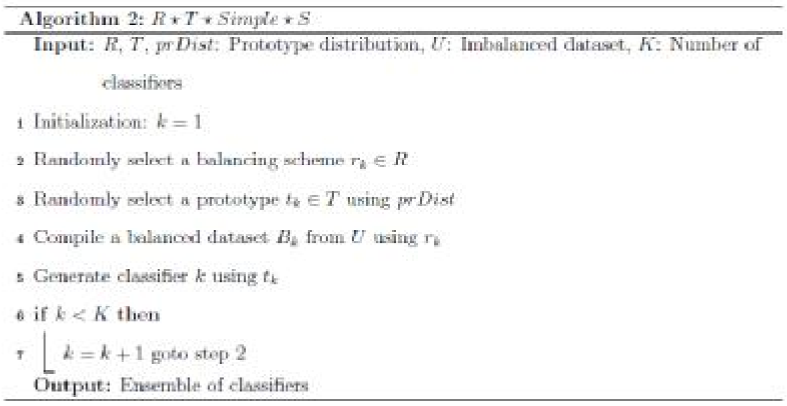
Using a single prototype but multiple balancing schemes is also addressed. R×t_k*Simple*P, R×t_k*Bagging*P and R×t_k*Boosting*P correspond to ensembles of 5×K members, where t_k is the prototype used and, K members are generated for each balancing scheme in R. The ensemble corresponding to R×t_k*Simple*P is shown in Figure 3. Averaging is used for the combination of the member outputs. By studying r_i×t_k*Simple, r_i×T*Simple and R×t_k*Simple*P , it is aimed to identify the relative importance of using single or multiple schemes for balancing and model development in the case of simple ensembles. This is also done for bagging- and boosting-based ensembles.


U
$r_1 \star t_k \star Simple$
$r_2 \star t_k \star Simple$
$r_3 \star t_k \star Simple$
$r_4 \star t_k \star Simple$
$r_5 \star t_k \star Simple$
Classifier $t_{k,1}$
Classifier $t_{k,K}$
Figure 3: Generation of R×t_k*Simple*P that is based on using a single prototype, t_k and the set of all balancing schemes. The ensemble corresponds to the classifiers in the dashed-line box.

The performances of different subsets of R and T are also evaluated. In particular, R_m×T*Simple*P corresponds to using a subset of m balancing schemes in R and the whole set of T in generating a simple ensemble. Consider the case for |R| = 5. We have ten different subsets of m = 2 balancing schemes. In this study, we evaluated all such ensembles and reported the average AUC scores. Similarly, utilizing subsets of T together with the whole set of R denoted by $R \times T_m$*Simple*P is explored. The experiments are repeated for bagging- and boosting-based ensembles. The ensembles obtained are named as $R_m \times T$*Bagging*P and $R \times T_m$*Bagging*P in the case of bagging. Similarly, the ensembles are named as $R_m \times T$*Boosting*P and $R \times T_m$*Boosting*P in the case of boosting.

3.3 Generation of Serial Ensembles

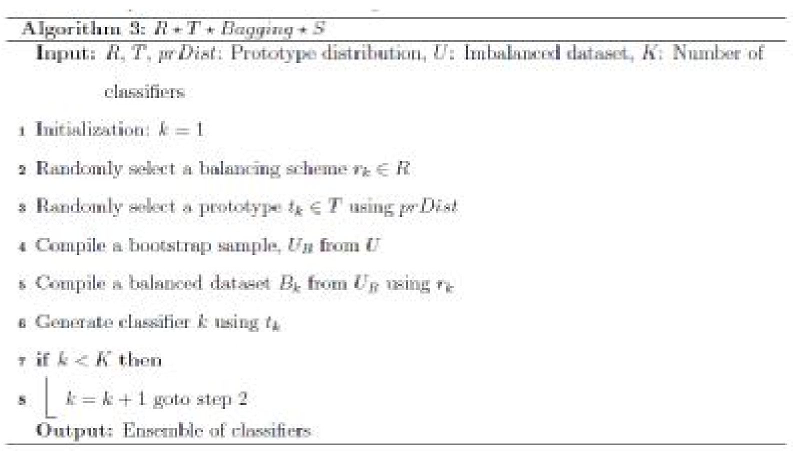
As mentioned above, all 2-tuples in $R \times T$ are used in generating serial ensembles. In particular, each member is developed using a randomly selected balancing scheme and a randomly selected prototype. Algorithm 2 presents R*T*Simple*S, which corresponds to a simple serial (S) ensemble. In this type of ensembles, weighted selection of the prototypes is addressed by taking into account various prototype distributions, prDist in Line 3. It should be noted that the distributions investigated include uniform distribution and others that are proportional to the individual performances of the prototypes.



Algorithm 2: $R * T * Simple * S$

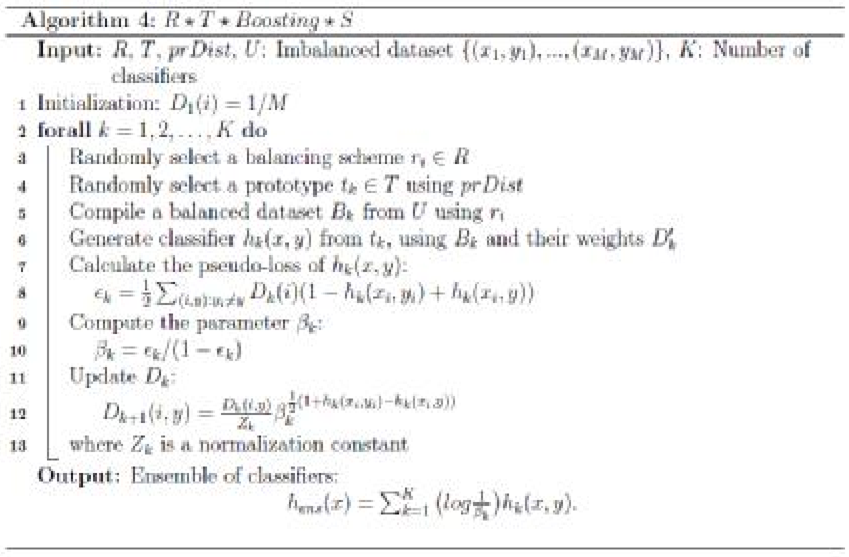
```
  Input: $R$, $T$, $prDist$: Prototype distribution, $U$: Imbalanced dataset, $K$: Number of
         classifiers
1 Initialization: $k = 1$
2 Randomly select a balancing scheme $r_k \in R$
3 Randomly select a prototype $t_k \in T$ using $prDist$
4 Compile a balanced dataset $B_k$ from $U$ using $r_k$
5 Generate classifier $k$ using $t_k$
6 if $k < K$ then
7     $k = k + 1$ goto step 2
  Output: Ensemble of classifiers
```

Algorithm 3 presents R*T*Bagging*S. This Algorithm is obtained by adding Line 4 into Algorithm 2. More specifically, balancing is performed on a bootstrap sample from the imbalanced dataset. This is expected to contribute to the diversity among the ensemble members. As in Algorithm 2, weighted selection of the prototypes is addressed by taking into account various prDist values in line 3 of Algorithm 3.



```
Algorithm 3: $R * T * Bagging * S$
  Input: $R$, $T$, $prDist$: Prototype distribution, $U$: Imbalanced dataset, $K$: Number of
         classifiers
1 Initialization: $k = 1$
2 Randomly select a balancing scheme $r_k \in R$
3 Randomly select a prototype $t_k \in T$ using $prDist$
4 Compile a bootstrap sample, $U_B$ from $U$
5 Compile a balanced dataset $B_k$ from $U_B$ using $r_k$
6 Generate classifier $k$ using $t_k$
7 if $k < K$ then
8     $k = k + 1$ goto step 2
  Output: Ensemble of classifiers
```

Algorithm 4 presents R*T*Boosting*S, which is based on Adaboost.M2 algorithm [36]. In each iteration of this algorithm, a balanced dataset denoted by B_k is computed using the randomly selected balancing scheme. The weights of the selected samples are normalized to compute D'_k. The next classifier is then generated using B_k and D'_k and the randomly selected prototype, t_k.



```
Algorithm 4: $R * T * Boosting * S$
Input: $R, T, prDist, U$: Imbalanced dataset $\{(x_1, y_1), ..., (x_M, y_M)\}$, $K$: Number of
        classifiers
Initialization: $D_1(i) = 1/M$
forall $k = 1, 2, \ldots, K$ do
    Randomly select a balancing scheme $r_i \in R$
    Randomly select a prototype $t_k \in T$ using $prDist$
    Compile a balanced dataset $B_k$ from $U$ using $r_i$
    Generate classifier $h_k(x, y)$ from $t_k$, using $B_k$ and their weights $D'_k$
    Calculate the pseudo-loss of $h_k(x, y)$:
        $\epsilon_k = \frac{1}{2} \sum_{(i,y):y_i \neq y} D_k(i)(1 - h_k(x_i, y_i) + h_k(x_i, y))$
    Compute the parameter $\beta_k$:
        $\beta_k = \epsilon_k/(1 - \epsilon_k)$
    Update $D_k$:
        $D_{k+1}(i, y) = \frac{D_k(i,y)}{Z_k} \beta_k^{\frac{1}{2}(1+h_k(x_i,y_i)-h_k(x_i,y))}$
    where $Z_k$ is a normalization constant
Output: Ensemble of classifiers:
        $h_{ens}(x) = \sum_{k=1}^{K} (log \frac{1}{\beta_k}) h_k(x, y)$.
```

3.4 Summary of the Ensembles Evaluated

Table 2 presents the ensembles developed to study the effectiveness of utilizing multiple balancing schemes and multiple prototypes in imbalance learning. The table also includes the number of classifiers in each ensemble which is based on the value of K, number of balancing methods and prototypes utilized in generating the ensemble. Due to the higher computational cost of RIMARC and SVM, the ensemble size is set to K = 50 for t_1, t_2 and t_3 as in [1], whereas K = 100 for t_4 and t_5. The ensembles implemented are also compared with six widely used techniques. These are SMOTEBagging, UnderBagging, OverBagging, SMOTEBoost, RUSBoost and DataBoost-IM. In fact, when r_i is selected as SMOTE, r_i*t_k*Boosting is identical to SMOTEBoost. Similarly, RUSboost is obtained when r_i is random undersampling. DataBoost-IM is the third ensemble from the boosting family of ensembles considered in this study. Since it has an additional step of focusing only on the hard samples in generating synthetic data, it is not identical to any of the ensembles listed in Table 2. In SMOTEBagging, a bootstrap sample of the majority class is used.

Table 2: The list of ensembles evaluated.

Ensemble name	Number of balancing methods	Number of prototypes	Ensemble size

r_i*t_k*Simple
r_i*t_k*Bagging
r_i*t_k*Boosting
	1	1	50 for {t_1, t_2, t_3}
			100 for {t_4, t_5}

r_i×T*Simple * P r_i×T*Bagging * P r_i×T*Boosting * P	1	5	20 for K = 4 50 for K = 10 100 for K = 20
R×t_k*Simple * P R×t_k*Bagging * P R×t_k*Boosting * P	5	1	20 for K = 4 50 for K = 10 100 for K = 20
R×T*Simple*P R×T*Bagging*P R×T*Boosting*P	5	5	100 for K = 4 250 for K =10 500 for K =20
R_m×T*Simple*P R_m×T*Bagging*P R_m×T*Boosting*P	m	5	5 × m ×K
R×T_m*Simple*P R×T_m*Bagging*P R×T_m* Boosting*P	5	m	5 × m ×K
R*T*Simple*S R*T*Bagging*S R*T*Boosting*S	5	5	100

The minority samples are generated using both oversampling and SMOTE [26]. In both UnderBagging and OverBagging, a bootstrap sample with replacement is taken separately from both minority and majority classes [5]. In the case of UnderBagging, the size of bootstrap samples in each class is equal to the original size of the minority class whereas, in OverBagging, it is equal to the original size of the majority class. As it can be seen in Figure 1, in our bagging family of ensembles, oversampling or undersampling is applied after a bootstrap sample is taken and class labels are not considered in selecting the samples. In all six reference ensembles, J48 was considered as the base learner and ensemble size is set as 100.

Chapter 4

EXPERIMENTAL RESULTS

The experiments are conducted on 66 datasets in KEEL collection which are originally from the UCI Repository [35].

Table 3: The datasets in KEEL collection and their imbalance ratios.

Dataset	Minority class size	Imbalance ratio	Dataset	Minority class size	Imbalance ratio
abalone19	32	129.44	ecoli-0-2-6-7 vs 3-5	22	9.18
yeast6	35	41.40	ecoli-0-1 vs 2-3-5	24	9.17
ecoli-0-1-3-7vs2-6	7	39.14	ecoli-0-4-6 vs 5	20	9.15
yeast5	44	32.73	yeast-0-2-5-7-9vs3-6-8	99	9.14
yeast-1-2-8-9 vs 7	30	30.57	yeast-0-2-5-6vs3-7-8-9	99	9.14
yeast4	51	28.10	yeast-0-3-5-9 vs 7-8	50	9.12
yeast-2 vs 8	20	23.10	glass-0-1-5 vs 2	17	9.12
glass5	9	22.78	ecoli-0-2-3-4 vs 5	20	9.10
yeast-1-4-5-8 vs 7	30	22.10	ecoli-0-6-7 vs 3-5	22	9.09
shuttle-c2-vs-c4	6	20.50	yeast-2 vs 4	51	9.08
glass-0-1-6 vs 5	9	19.44	ecoli-0-3-4 vs 5	20	9.00
abalone9-18	42	16.40	page-blocks0	559	8.79
page-blocks-13vs4	28	15.86	ecoli3	35	8.60
ecoli4	20	15.80	yeast3	163	8.10
glass4	13	15.46	glass6	29	6.38
yeast-1 vs 7	30	14.30	segment0	329	6.02
shuttle-c0-vs-c4	123	13.87	ecoli2	52	5.46
ecoli-0-1-4-6 vs 5	20	13.00	new-thyroid2	35	5.14
cleveland-0 vs 4	13	12.62	new-thyroid1	35	5.14
ecoli-0-1-4-7vs5-6	25	12.28	ecoli1	77	3.36
glass2	17	11.59	vehicle0	199	3.25
glass-0-1-4-6vs 2	17	11.06	glass-0-1-2-3 vs 4-5-6	51	3.20
ecoli-0-1 vs 5	20	11.00	vehicle3	212	2.99
glass-0-6 vs 5	9	11.00	vehicle1	217	2.90
led7digit-0-2-4-5- vs 1	37	10.97	vehicle2	218	2.88
ecoli-0-1-4-7vs2-3-	29	10.59	haberman	81	2.78
glass-0-1-6 vs 2	17	10.29	yeast1	429	2.46
ecoli-0-6-7 vs 5	20	10.00	glass0	70	2.06
vowel0	90	9.98	iris0	50	2.00
yeast-0-5-6-7-9vs 4	51	9.35	pima	268	1.87
ecoli-0-3-4-7vs5-6	25	9.28	wisconsin	239	1.86
ecoli-0-3-4-6vs 5	20	9.25	ecoli-0 vs 1	77	1.86
glass-0-4 vs 5	9	9.22	glass1	76	1.82

In this collection, some datasets are different partitions of one multi-class dataset. For instance, in “glass0”, class-0 of “glass” dataset is the minority class whereas the samples in all other classes form the majority class. The characteristics of these datasets, namely the number of samples in the minority class and the imbalance ratio that is defined as the ratio of the number of samples in majority class to that of the minority are presented in Table 3. As seen in the table, each dataset is highly different from many of the others. For instance, imbalance ratios are between 1.82 and 129.44. Table 4 presents the distribution of the datasets according to their minority class sizes (MS) and imbalance ratios (IR). The datasets having an imbalance ratio above 10.0 are categorized as high imbalance whereas the low imbalance datasets have ratios less than 5.5. The datasets are grouped as small minority if the number of samples in the minority class is at most 20 whereas large minority datasets include at least 90 samples.

Table 4: Categorization of datasets according to their imbalance ratios and size of the minority classes.

	Low imbalance ratio (IR<5.5)	Moderate imbalance ratio (5.5<=IR<=10)	High imbalance ratio (IR>10)
Small minority (MS<=20)	0	7	14
Moderate minority (20<MS<90)	10	9	12
Large minority (MS>=90)	7	6	1

Table 5 presents the balancing schemes and prototypes used in this thesis. In SMOTE, the number of neighbors used for generating synthetic minority samples is set as k = 5 as it generally used in [20]. ADASYN is a variant of SMOTE where more minority samples are generated for samples that are harder to classify. ANS is another variant where the number of neighbors, k is not fixed and a different value is dynamically assigned to each minority instance.

Table 5: The balancing schemes and prototypes used in this study.

Balancing method		Classifier prototype	
r_1:	Random undersampling	t_1:	RIMARC
r_2:	Random oversampling	t_2:	Logistic regression (LR)
r_3:	SMOTE	t_3:	SVM with Gaussian kernel
r_4:	ADASYN	t_4:	Nearest neighbor classifier
r_5:	ANS	t_5:	Decision tree (J48)

RIMARC is a recently proposed classification scheme that is based on discretizing continuous features [37]. The discretization thresholds of each feature are calculated in a way that maximizes the AUC that can be achieved by the feature [38]. After discretization, the feature values are computed as the percentage of positive samples in each interval. The overall score generated by the classifier is calculated as the weighted sum of the probabilities obtained from all features. The weight of each feature is proportional to its individual AUC score. It is experimentally shown that the discretization policy applied helps to achieve higher AUC scores when compared to other widely used classifiers [37]. It should be noted that, logistic regression classifier is also observed to achieve higher AUC scores when compared to many other widely used classifiers [37]. J48 was used since it is one of the most frequently used implementation of decision trees [1, 9]. The average AUC scores obtained using 5×2-fold cross validation are employed in comparing different ensembles [39].

Table 6 presents the average AUC scores obtained using a single balancing scheme and a single prototype, namely r_i*t_k*Simple and r_i*t_k*Bagging and r_i*t_k*Boosting, respectively in parts (a), (b) and (c). Last rows and columns present the column-wise and row-wise averages, respectively. The scores presented in boldface are the highest row or column averages. The average of all 25 AUC scores is presented in underlined form. The results show that undersampling provides the highest score when averaged over all prototypes in all three types of ensembles. SVM and LR provide the highest scores for r_i*t_k*Simple and r_i*t_k*Bagging, respectively whereas J48 provides the best score when used in r_i*t_k*Boosting. J48 is the second best in bagging-based ensembles whereas LR is the second best in simple ensembles. Although the highest AUC score is obtained by a boosting-based ensemble using J48 as the classifier prototype (i.e. r_3*t_5*Boosting or SMOTEBoost), when the col-AVG scores are compared, it can be seen that the performance of boosting based ensembles is higher than those of both simple and bagging-based ensembles only for J48. Therefore, selection of the classifier prototype is more critical in the case of boosting-based ensembles. Comparing the overall averages given in underlined form, it can be concluded that bagging-based ensembles generally achieve superior scores when compared to simple and boosting-based ensembles. Table 7 presents the average AUC scores obtained by using (a) $r_i \times T$*Simple*P, (b) $R \times t_k$*Simple*P and (c) $R \times T$*Simple*P for $K = 4$, 10 and 20. For the ensembles in (a) and (b), totally $5 \times K$ members are utilized, leading to 20, 50 and 100 classifiers for $K = 4$, 10 and 20, respectively. In the case of $R \times T$*Simple*P, by considering all 25 2-tuples (r_i, t_k), ensemble sizes are 100, 250 and 500, respectively for $K = 4$, 10 and 20. When $r_i \times T$*Simple*P and $R \times t_k$*Simple*P are compared, it can seen that the scores achieved using multiple prototypes given in part (a) are much higher than those obtained by using a single prototype that are presented in part (b). The scores are also superior to those obtained using individual 2-tuples presented in part (a) of Table 6. It can be concluded that it is highly crucial to utilize multiple prototypes in simple ensembles, regardless of the balancing scheme employed. $R \times T$*Simple*P surpasses the ensembles corresponding to $r_i \times T$*Simple*P and $R \times t_k$*Simple*P, which means that better ensembles are generated by using multiple balancing techniques and classifier prototypes.

Table 6: Average AUC scores achieved using (a) r_i*t_k*Simple, (b) r_i*t_k*Bagging and r_i*t_k*Boosting.

	t_1	t_2	t_3	t_4	t_5	row-AVG
r_1	0.8966	0.9075	0.8871	0.8939	**0.9083**	**0.8987**
r_2	0.8947	0.8938	0.9065	0.8014	0.8640	0.8736
r_3	0.8950	0.8945	0.9058	0.8386	0.8788	0.8844
r_4	0.8921	0.8929	0.9023	0.8397	0.8753	0.8817
r_5	0.8871	0.8930	0.8993	0.8182	0.8583	0.8728
col-AVG	0.8931	0.8963	**0.9002**	0.8384	0.8769	0.8810

(a)

	t_1	t_2	t_3	t_4	t_5	row-AVG
r_1	0.8966	**0.9151**	0.9002	0.9018	0.9104	**0.9048**
r_2	0.8940	0.9134	0.9085	0.8705	0.9094	0.8992
r_3	0.8948	0.9146	0.9084	0.8870	0.9132	0.9036
r_4	0.8927	0.9132	0.9053	0.8878	0.9123	0.9022
r_5	0.8890	0.9139	0.9054	0.8839	0.9092	0.9003
col-AVG	0.8934	**0.9140**	0.9055	0.8862	0.9109	0.9020

(b)

	t_1	t_2	t_3	t_4	t_5	row-AVG
r_1	0.8264	0.9020	0.9130	0.8979	0.9131	**0.8905**
r_2	0.8194	0.9013	0.9006	0.8696	0.9114	0.8805
r_3	0.8176	0.9039	0.9027	0.8893	**0.9153**	0.8858
r_4	0.8227	0.8990	0.8965	0.8844	0.9148	0.8835
r_5	0.8117	0.9011	0.8947	0.8762	0.9110	0.8784
col-AVG	0.8196	0.9015	0.9015	0.8835	**0.9131**	0.8837

(c)

However, the actual gain due to using multiple resampling schemes can be better evaluated when part (b) of Table 7 is compared with part (a) in Table 6. For instance, the score achieved using t_3 and R (0.9078) in Table 7 part (b) is higher than all the scores in the third column in part (a) of Table 6. This shows that, for t_3, utilizing multiple resampling schemes should be preferred.

Table 7: Average AUC scores achieved using (a) r_i×T*Simple*P, (b) R×t_k*Simple*P and (c) R×T*Simple*P.

	r_i ×T * Simple * P		
	K=4	K=10	K=20
r_1	0.9198	0.9237	0.9242
r_2	0.9277	0.9281	0.9283
r_3	0.9279	0.9284	0.9284
r_4	0.9261	0.9266	0.9268
r_5	0.9264	0.9267	0.9268

(a)

	R × t_k * Simple * P		
	K=4	K=10	K=20
t_1	0.8962	0.8963	0.8964
t_2	0.9040	0.9082	0.9097
t_3	0.9079	0.9075	0.9078
t_4	0.8868	0.8917	0.8942
t_5	0.8987	0.9049	0.9072

(b)

R × T * Simple * P		
K=4	K=10	K=20
0.9297	0.9301	0.9301

(c)

In fact, this is also true for t2 and t4. However, the performance gain is not significant when compared using only r_1. Table 8 presents the average AUC scores obtained by using, (a) r_i×T*Bagging*P, R×t_k*Bagging*P and (c) R×T*Bagging * P for K = 4, 10 and 20.

Table8: Average AUC scores achieved using (a) r_i×T*Bagging*P, (b) R×t_k*Bagging*P and (c) R×T*Bagging*P.

	r_i × T * Bagging * P		
	K=4	K=10	K=20
r_1	0.9179	0.9225	0.9250
r_2	0.9274	0.9303	0.9310
r_3	0.9281	0.9301	0.9305
r_4	0.9266	0.9288	0.9296
r_5	0.9268	0.9295	0.9300

(a)

	R × t_k * Bagging * P		
	K=4	K=10	K=20
t_1	0.8944	0.8954	0.8950
t_2	0.9120	0.9163	0.9176
t_3	0.9089	0.9096	0.9098
t_4	0.8935	0.8990	0.9019
t_5	0.9065	0.9134	0.9174

(b)

R × T * Bagging * P		
K=4	K=10	K=20
0.9300	0.9311	0.9314

(c)

The results are consistent with those obtained in simple ensembles. However, slightly better scores can be obtained in general. When the AUC scores presented in parts (a) and (c) are compared, it can be seen that using multiple balancing schemes and large K (i.e. 10 or 20) leads to better scores than all ensembles presented in part (a) which are based on a single balancing scheme. Comparing part (b) of Table 8 with part (b) in Table 6, it can be seen that utilizing multiple resampling schemes leads to better scores for all classifiers except for t_1. Therefore, as in the case of simple ensembles, instead of selecting a good balancing strategy or classifier prototype, using multiple schemes and prototypes should be considered. However, as in the case of simple ensembles, the gain due to utilizing multiple balancing schemes is not as remarkable as employing multiple prototypes. Table 9 presents the average AUC scores obtained by using, (a) ri×T*Boosting*P, R×t_k*Boosting*P and (c) R×T*Boosting*P for K = 4, 10 and 20.

Table 9: Average AUC scores achieved using (a) r_i×T*Boosting*P, (b) R×t_k*Boosting*P and (c) R×T*Boosting*P.

	r_i× T * Boosting * P		
	K=4	K=10	K=20
r_1	0.9160	0.9212	0.9251
r_2	0.9200	0.9186	0.9160
r_3	0.9220	0.9181	0.9172
r_4	0.9188	0.9047	0.9035
r_5	0.9191	0.9063	0.9154

(a)

	R × t_k * Boosting * P		
	K=4	K=10	K=20
t_1	0.8053	0.8155	0.8205
t_2	0.9095	0.9090	0.9105
t_3	0.9112	0.9111	0.9095
t_4	0.8947	0.8959	0.8953
t_5	0.9125	0.9170	0.9198

(b)

R × T * Boosting * P		
K=4	K=10	K=20
0.9253	0.9229	0.9210

(c)

As in the case of simple and bagging-based ensembles, totally 5×K members are utilized in both (a) and (b), leading to 20, 50 and 100 classifiers for K = 4, 10 and 20, respectively. Consider part (a) where the scores obtained using multiple prototypes and a single balancing scheme are presented. When compared with the scores presented in part (c) of Table 6, it can be seen that the performance may degrade when multiple prototypes are employed. More specifically, inferior scores than the best-fitting prototypes are obtained for r_4. When 100 members are employed, the AUC score achieved using {r_4, t_5} is 0.9148 (from Table 6) whereas 0.9035 is obtained using r_4 and multiple prototypes. On the other hand, the best score achieved by using r_1 is improved from 0.9131 to 0.9251, which is greater than the scores of all tuples. In fact, r_4 can be considered as an exception since multiple prototypes lead to better scores than the best individual for all other balancing schemes.

In boosting-based ensembles, the performance gain from multiple balancing schemes depends on the classifier prototype. As it can be seen in part (b) of Table 9, the scores achieved using t_1 are highly poor. These scores are also inferior than the score achieved using (r_1, t_1), 0.8264. This is also the case for t_3 and t_4. In other words, using multiple resampling schemes does not provide better ensembles for these base learners. However, using multiple balancing schemes, the highest score obtained using a single balancing scheme and prototype (i.e. (r_3, t_5), in Table 6) is improved from 0.9153 to 0.9198.

The AUC scores obtained by utilizing multiple balancing schemes and multiple classifier prototypes are presented in part (c) of Table 9. The scores are higher than the best scores achieved using a single prototype given in part (b). This verifies the importance of using multiple prototypes in boosting-based ensembles as well. However, the difference in the scores is not notable when compared to using only r_1 together with T as presented in the first row of part (a). In summary, as in simple and bagging-based ensembles, the highest scores are obtained using multiple prototypes. However, when multiple prototypes are used, a notable improvement is not observed by utilizing multiple resampling schemes when compared with r_1.

The experimental results presented in Tables 7, 8 and 9 clearly demonstrate the importance of using heterogeneous ensembles in imbalance learning. In order to investigate the contribution of subsets of T (denoted by T_m when m prototypes are used) to the ensemble performance when used together with the whole set of R (i.e. all 2-tuples in $R \times T_m$), further experiments are conducted. Similarly, the performances of all 2-tuples in $R_m \times T$ are evaluated. The average AUC values obtained for m = 1, 2, 3, 4, 5 are presented in Figure 4.

It should be noted that, when m = 1, the average AUCs obtained using five ensembles are reported whereas the scores of ten ensembles are averaged when m = 2. Each ensemble involves m × |T| × K members. For instance, 200 members are computed when m = 2 since |T| = 5 and K = 20, in both R_m×T*Simple*P and R×T_m*Simple*P. Similarly, the ensemble size is 300 when m = 3. The first row of the figures illustrates the gains obtaining by using increasing numbers of balancing methods. The figures in the second row show that increasing the number of prototypes leads to notable improvements in the performance scores for all three types of ensembles.

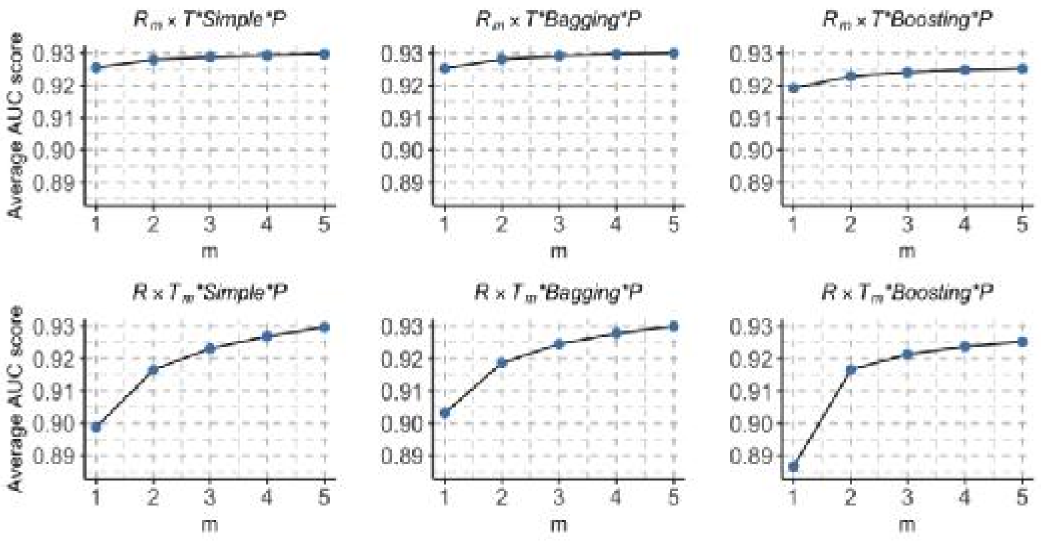


Figure 4: The average AUC scores achieved using R_m and T_m for m = 1, . . . ,5.

The figure on the left in Figure 5 presents the average AUC scores obtained using R×t_k*Simple*S for individual prototypes denoted by t_k, k = 1, . . . , 5 and the whole set of prototypes, T.

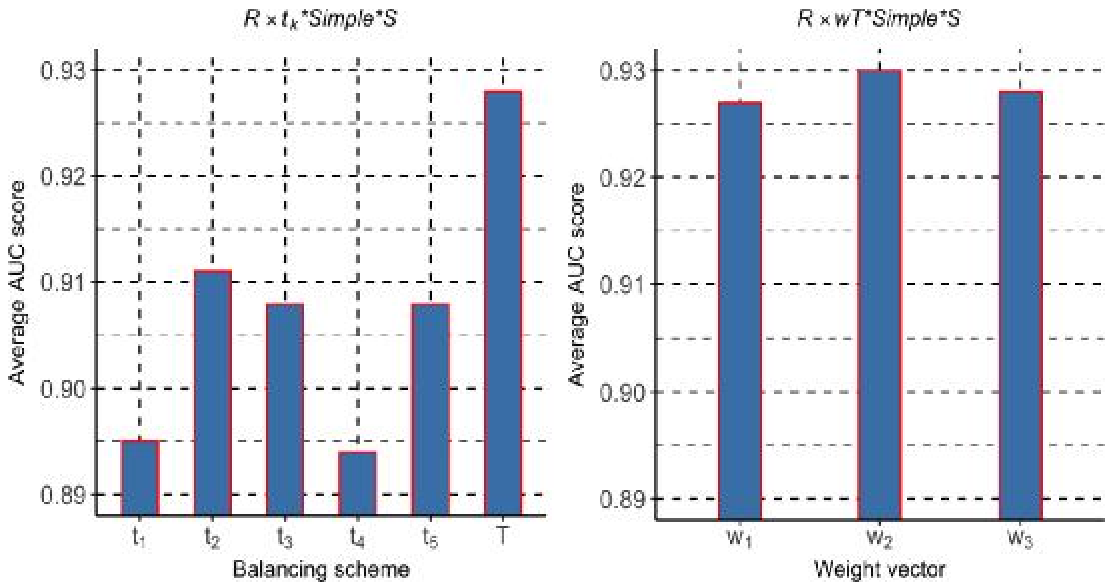


Figure 5: The average AUC scores achieved using simple serial ensembles.

It should be noted that the prototype distribution, prDist is selected as uniform in the case of T. The experimental results show that utilizing multiple prototypes is also effective in serial ensembles. In Figure 5, the effect of using different distributions is presented on the right. The selection of prDist is based on the col-AVG values reported in Table 6. The first distribution is set as $w_1 = (\frac{2}{14}, \frac{2}{14}, \frac{8}{14}, \frac{1}{14}, \frac{1}{14})$, aiming at using the most successful prototypes determined using Table 6 in vast majority of the members. The second distribution $w_2 = (\frac{3}{13}, \frac{3}{13}, \frac{4}{13}, \frac{1}{13}, \frac{2}{13})$ is used to select the prototypes more uniformly by considering their relative performances into account. w_3 corresponds to the uniform distribution that is provided as a reference. It can be seen that better scores are obtained using w_2. The experiments are repeated for bagging- and boosting-based serial ensembles as presented in Figures 6 and 7, respectively.

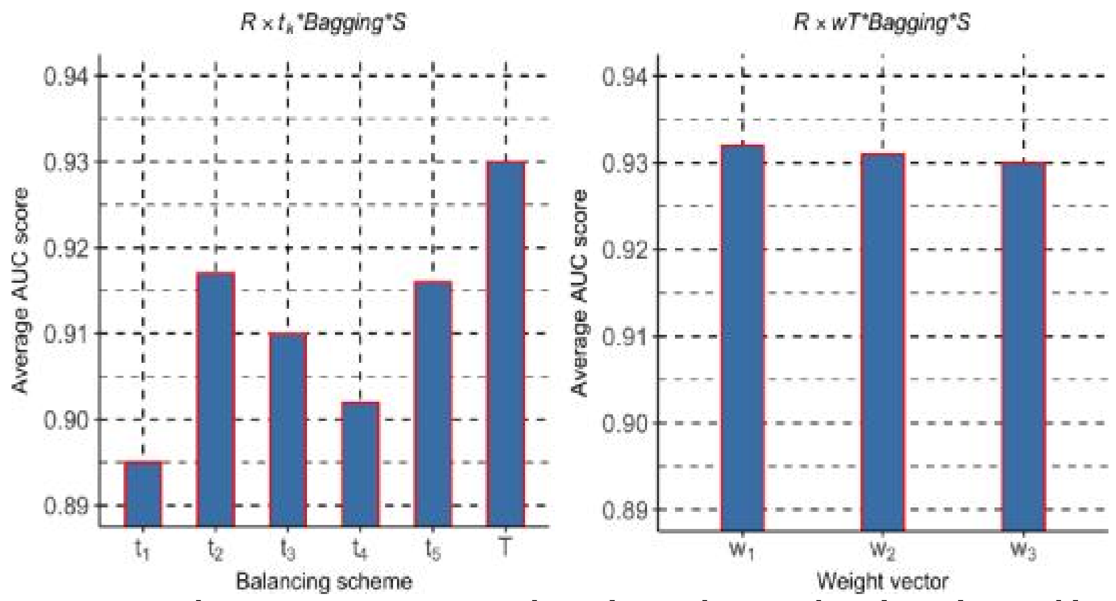


Figure 6: The average AUC scores achieved using bagging-based serial ensembles.

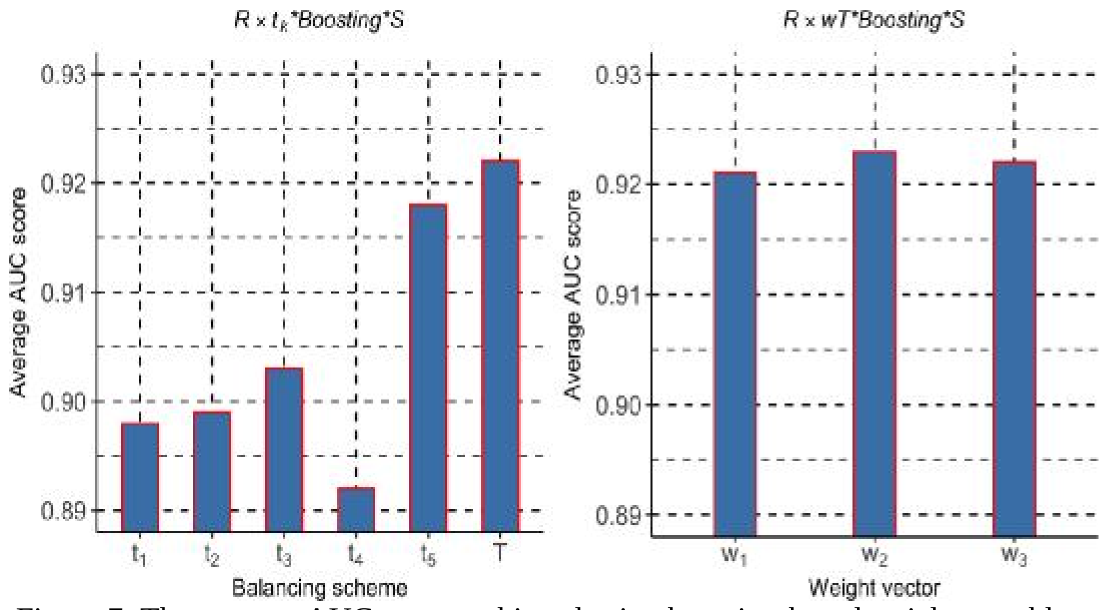


Figure 7: The average AUC scores achieved using boosting-based serial ensembles.

In bagging-based ensembles, the weights are selected as $w_1 = (\frac{1}{16}, \frac{6}{16}, \frac{2}{16}, \frac{1}{16}, \frac{6}{16})$, and $w_2 = \left(\frac{2}{14}, \frac{4}{14}, \frac{3}{14}, \frac{1}{14}, \frac{4}{14}\right)$ using the same logic as in case of simple ensembles. In boosting-based ensembles the weights are selected as $w_1 = \left(\frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{16}{20}\right)$ and $w_2 = \left(\frac{1}{13}, \frac{3}{13}, \frac{3}{13}, \frac{2}{13}, \frac{4}{13}\right)$. The experimental results show that multiple prototypes provides superior scores when compared to individual prototypes for all three type of ensembles. The additional improvements that could be achieved by utilizing different weights demonstrate that a good ensemble should include a rich set of prototypes, where the percentage of each prototype is proportional to its individual performance.

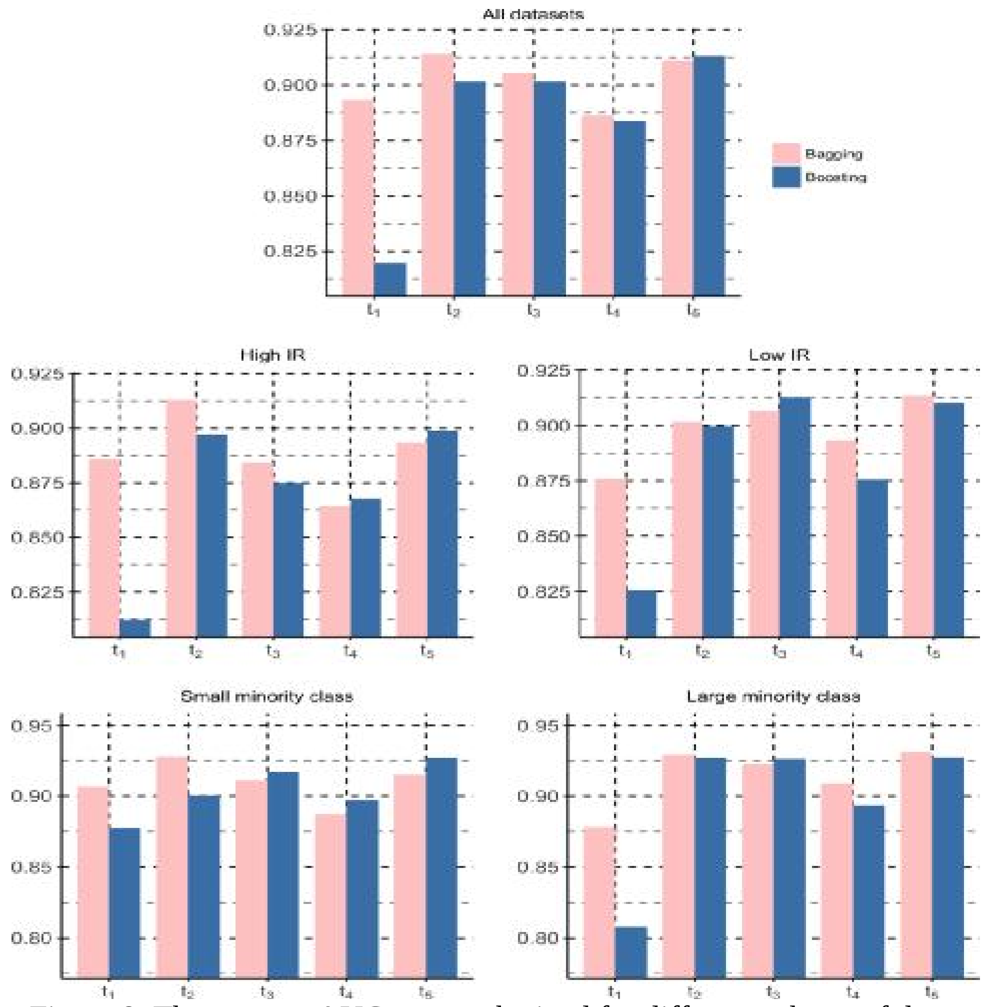


Figure 8: The average AUC scores obtained for different subsets of datasets. The first row presents the scores over all datasets. The second row corresponds to scores on datasets having high and low imbalance ratios (IR). The last row is average scores on datasets having small and large minority classes.

The average AUC scores obtained using r_i*t_k*Bagging and r_i*t_k*Boosting on different subsets of datasets are presented in Figure 8. It should be noted that each bar corresponds to the average for five balancing schemes. The figures show that the highest scores are generally achieved using bagging-based ensembles. When the imbalance ratio (IR) is low or the size of minority class is large, bagging J48 provides the highest scores whereas bagging LR provides the highest scores in the other two cases. For LR, bagging is the best ensembling scheme for all four groups of datasets whereas, in the case of J48, the relative performance of bagging and boosting depends on the characteristic of the dataset. More specifically, boosting J48 leads to better scores when the dataset has high imbalance or small minority class.

Table 10: The average AUC scores and ranks obtained for the ensembles implemented in this thesis. Ensembles in first six rows utilize multiple-prototypes. The size of all ensembles is 100.

Type	Ensemble	Avg. AUC score	Avg. rank
Multiple prototypes	R × T * Simple * P	0.9297	6.13
	R × T * Bagging * P	0.9300	5.59
	R × T * Boosting * P	0.9253	7.76
	R × T * Simple * S	0.9293	6.83
	R × T * Bagging * S	0.9298	5.83
	R × T * Boosting * S	0.9233	9.09
Single prototype	r_1 * t_5 * Simple	0.8871	13.24
	r_1 * t_2 * Bagging	0.8940	12.14
	R × t_2 * Simple * P	0.9097	13.47
	R × t_2 * Bagging * P	0.9176	11.61
	R × t_5 * Boosting * P	0.9198	9.78
	R × t_2 * Simple * S	0.9112	12.48
	R × t_2 * Bagging * S	0.9175	11.35
	R × t_5 * Boosting * S	0.9175	11.22
	SMOTEBagging	0.9066	13.29
	UnderBagging	0.9093	11.92
	OverBagging	0.8975	15.65
	SMOTEBoost	0.9153	10.73
	RUSBoost	0.9131	10.65
	DataBoost-IM	0.9157	11.22

Table 10 presents the average AUC scores and average ranks of a selected subset of ensembles developed in this study. First six rows are multi-balancing and multi-prototype simple, bagging- and boosting-based ensembles. Other 14 ensembles are based on a single prototype.

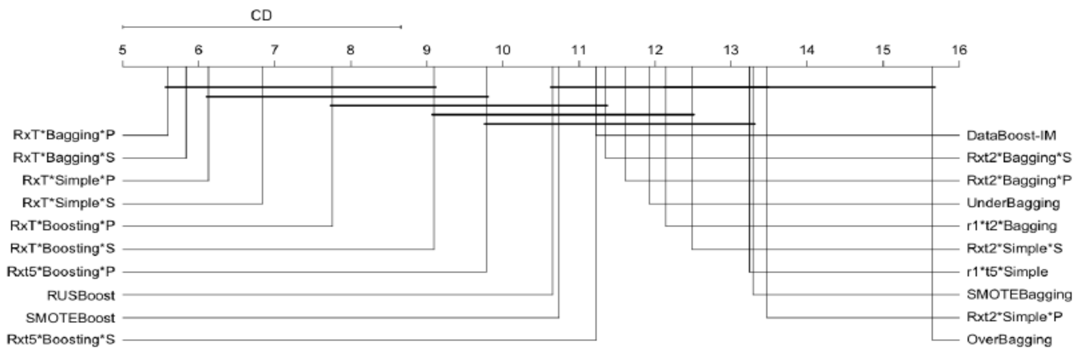
More specifically, r_1*t_5*Simple is the best simple ensemble (highest score in part (a) of Table 6) and r_1*t_2*Bagging is the best bagging-based ensemble (highest score in part (b) of Table 6), both utilizing a single balancing scheme and prototype. Following three ensembles are the best simple, bagging- and boosting-based parallel ensembles utilizing a single prototype but multiple balancing schemes. The best simple, bagging- and boosting- based serial ensembles employing a single prototype but multiple balancing schemes are listed next. SMOTEBagging, UnderBagging, OverBagging, SMOTEBoost, RUSBoost and DataBoost-IM are listed in last six rows. Wilcoxon signed-rank test [41] is also performed to compare the multi-prototype ensembles presented in top six rows of Table 10 with the other 14 ensembles using $\alpha = 0.05$. The p-values obtained, numbers of wins, losses, ties are presented in Table 11.

Table 11: Statistical evaluation of multi-balancing and multi-prototype systems developed. The size of all ensembles is 100. [1]: R×T*Simple*P, [2]: R×T*Bagging*P, [3]: R×T*Boosting*P , [4]: R×T*Simple*S, [5]: R×T*Bagging*S, [6]: R×T*Boosting*S.

	[1]	[2]	[3]	[4]	[5]	[6]
r_1 * t_5 * Simple	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
r_1 * t_2 * Bagging	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
R × t_2 * Simple * P	0.0000	0.0000	0.0002	0.0000	0.0000	0.0011
R × t_2 * Bagging * P	0.0000	0.0000	0.0119	0.0000	0.0000	0.0744
R × t_5 * Boosting * P	0.0000	0.0000	0.0001	0.0000	0.0000	0.0417
R × t_2 * Simple * S	0.0000	0.0000	0.0009	0.0000	0.0000	0.0059
R × t_2 * Bagging * S	0.0000	0.0000	0.0145	0.0000	0.0000	0.0809
R × t_5 * Boosting * S	0.0000	0.0000	0.0000	0.0000	0.0000	0.0030
SMOTEBagging	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
UnderBagging	0.0000	0.0000	0.0001	0.0000	0.0000	0.0015
OverBagging	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
SMOTEeBoost	0.0000	0.0000	0.0004	0.0000	0.0000	0.0486
RUSBoost	0.0000	0.0000	0.0131	0.0000	0.0000	0.2764
DataBoost-IM	0.0000	0.0000	0.0001	0.0000	0.0000	0.0042
Number of wins	14	14	14	14	14	11
Number of losses	0	0	0	0	0	0
Number of ties	0	0	0	0	0	3

These two tables show that the ensembles constructed using multiple balancing schemes and multiple prototypes perform significantly better that the others which are based on a single prototype. “Nemenyi” test is used for pair wise comparison of all ensembles [41].

Figure 9: Comparative evaluation of all ensembles using Nemenyi test where α=0.05. The R package “scmamp” is used.



As presented in Figure 9, Ensembles for which the difference in average ranks is lower than the critical difference (CD) is connected by a black line. According to this test, R×T*Bagging*P and R×T*Bagging*S are significantly better than all homogeneous ensembles employing only one type of a classifier.

Chapter 5

CONCLUSIONS

In this thesis, the use of multi-prototype and multi-balancing ensembles for imbalance learning is addressed. Simple, bagging- and boosting-based ensembles are implemented for this purpose. Experiments conducted on 66 datasets in KEEL repository have shown that multiple prototype ensembles provides significantly better AUC scores when compared to the ensembles utilizing a single prototype. The contribution of employing multiple balancing schemes is also taken into consideration. Experimental results have shown that slight improvements can be achieved for majority of the classifier prototypes, especially in simple and bagging-based ensembles. Prototype weighting is also addressed using serial ensembles. Additional improvements are achieved by utilizing weights that are proportional to the individual performance scores of the classifier prototypes.

The experimental results have also shown that the highest scores are achieved when all five prototypes are used. Since multiple prototypes provide significant improvements, further research should be conducted for obtaining an even better prototype set. The effect of adding more classifier prototypes should be investigated. Also, the characteristics of the classifiers that form the best-fitting set should also be explored. The weights selected in serial ensembles were not tuned according to a particular objective function. Further research should be conducted to investigate the effect of alternative weighting strategies.

REFERENCES

[1] J. F. Díez-Pastor, J. J. Rodríguez, C. I. García-Osorio, and L. I. Kuncheva. Random balance: Ensembles of variable priors classifiers for imbalanced data. Knowledge-Based Systems, 85:96–111, 2015.

[2] L. I. Kuncheva, A. Arnaiz-Gonzalez, J. Diez-Pastor, and I. A. D. Gunn. Instance selection improves geometric mean accuracy: A study on imbalanced data classification, 2018. arXiv:1804.07155.

[3] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing. Learning from class-imbalanced data: Review of methods and applications. Expert Systems with Applications, 73:220–239, 2017.

[4] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrer. A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches. IEEE Transactions on Systems Man and Cybernetics Part C, 42(4):463–484, July 2012.

[5] A. Fernandez, M. J. del Jesus, and F. Herrera. Hierarchical fuzzy rule based classification systems with genetic rule selection for imbalanced datasets. International Journal of Approximate Reasoning, 50(3):561–577, 2009.

[6] X. Y. Liu, J. Wu, and Z. H. Zhou. Exploratory undersampling for class-imbalance learning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(2):539–550, April 2009.

[7] A. Irtaza, S. M. Adnan, K. T. Ahmed, A. Jaffar, A. Khan, A. Javed, and M. T. Mahmood. An ensemble based evolutionary approach to the class imbalance problem with applications in CBIR. Applied Sciences, 8(495), 2018.

[8] M. Galar, A. Fernandez, E. B. Tartas, and F. Herrera. EUSBoost: Enhancing ensembles for highly imbalanced datasets by evolutionary undersampling. Pattern Recognition, 46(12):3460–3471, 2013.

[9] J. F. Diez-Pastor, J. Rodriguez, C. Garcia-Osorio, and L. Kuncheva. Diversity techniques improve the performance of the best imbalance learning ensembles. Information Sciences, 325:98–117, 2015.

[10] V. Lopez, A. Fernandez, S. Garcia, V. Palade, and F. Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. Information Sciences, 250:113–141, 2013.

[11] J. Gong and H. Kim. RHSBOOST: Improving classification performance in imbalance data. Computational Statistics and Data Analysis, 111:1–13, 2017.

[12] D. A. Cieslak and N. V. Chawla. Learning decision trees for unbalanced data. In Proceedings of the 2008 European Conference on Machine Learning and Knowledge Discovery in Databases - Part I, ECML PKDD ’08, pages 241–256. Springer-Verlag, 2008.

[13] W. Liu, S. Chawla, D. A. Cieslak, and N. V. Chawla. A robust decision tree algorithm for imbalanced data sets. In Proceedings of the SIAM International

Conference on Data Mining, pages 766–777, 2010.

[14] J. R. Quinlan. Improved estimates for the accuracy of small disjuncts. Machine Learning, 6(1):93–98, Jan 1991.

[15] B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’01, pages 204–213, NY, USA, 2001.

[16] C. X. Ling, V. S. Sheng, and Q. Yang. Test strategies for cost-sensitive decision trees. IEEE Transactions on Knowledge and Data Engineering, 18(8):1055–1067, 2006.

[17] K. Veropoulos, C. Campbell, and N. Cristianini. Controlling the sensitivity of support vector machines. In Proceedings of the International Joint Conference on AI, pages 55–60, 1999.

[18] C. Jian, J. Gao, and Y. Ao. A new sampling method for classifying imbalanced data based on support vector machine ensemble. Neurocomputing, 193:115–122, 2016.

[19] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research, 16:341–378, 2002.

[20] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap. DBSMOTE: Density-based synthetic minority over-sampling technique. Applied Intelligence, 36(3):664–684, 2012.

[21] W. Siriseriwan and K. Sinapiromsaran. Adaptive neighbor synthetic minority oversampling technique under 1NN outcast handling. Songklanakarin Journal of Science and Technology, 39(5):565–576, 2017.

[22] H. Haibo, Y. Bai, E. A. Garcia, and S. Li. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In IEEE International Joint Conference on Neural Networks (IEEE WorId Congress on Computational Intelligence), pages 1322–1328, 2008.

[23] H. Guo and H. L. Viktor. Learning from imbalanced data sets with boosting and data generation: The databoost-IM approach. SIGKDD Explorations, 6(1):30–39, June 2004.

[24] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. Adacost: Misclassification cost-sensitive boosting. In Proceedings of the Sixteenth International Conference on Machine Learning, ICML'99, pages 97–105, San Francisco, CA, USA, 1999.

[25] Y. Sun, M. S. Kamel, A. K. C. Wong, and Y. Wang. Cost-sensitive boosting for classification of imbalanced data. Pattern Recognition, 40:3358–3378, 2007.

[26] S. Wang and X. Yao. Diversity analysis on imbalanced data sets by using ensemble models. In IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2009, pages 324–331, March 2009.

[27] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. Bowyer. SMOTEBoost: Improving prediction of the minority class in boosting. In Knowledge Discovery in Databases: PKDD 2003, pages 107–119. Springer Berlin Heidelberg, 2003.

[28] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano. RUSBoost: A hybrid approach to alleviating class imbalance. IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans, 40(1):185–197, 2010.

[29] C. J. Whitaker and L. I. Kuncheva. Examining the relationship between majority vote accuracy and diversity in bagging and boosting. Technical Report, School of Informatics, University of Wales, Bangor, 2003.

[30] L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Machine Learning, 51:181–207, 2003.

[31] B. Krawczyk. Learning from imbalanced data: open challenges and future directions. Progress in Artificial Intelligence, 5(4):221–232, Nov 2016.

[32] L. Nanni, S. Brahnam, S. Ghidoni, and A. Lumini. Toward a general-purpose

heterogeneous ensemble for pattern classification. Computational Intelligence and Neuroscience, 2015.

[33] J. Large, J. Lines, and A. J. Bagnall. The heterogeneous ensembles of standard classification algorithms (HESCA): the whole is greater than the sum of its parts. CoRR, abs/1710.09220, 2017.

[34] E. N. de Souza and S. Matwin. Extending adaboost to iteratively vary its base classifiers. In Proceedings of the 24th Canadian Conference on Advances in Artificial Intelligence, Canadian AI'11, pages 384–389, Berlin, Heidelberg, 2011. Springer-Verlag.

[35] J. Alcal´a-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. Garc´ıa, L. S´anchez, and F. Herrera. Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework. Journal of Multiple-Valued Logic and Soft Computing, 17:255–287, 2011.

[36] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In Machine Learning: Proceedings of the thirteenth national conference, Morgan Kauffmann, pages 148–156, 1996.

[37] H. A. Guvenir and M. Kurtcephe. Ranking instances by maximizing the area under roc curve. IEEE Transactions on Knowledge and Data Engineering, 25(10):2356–2366, 2013.

[38] M. Kurtcephe and H. A. Guvenir. A discretization method based on

maximizing the area under receiver operating characteristic curve. International Journal of Pattern Recognition and Artificial Intelligence, 27(01):1350002, 2013.

[39] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning. Neural Computation, 7(10):1895–1923, 1998.