Handling the Class Imbalance Problem in Binary Classification

By

Nadyah Obaid Salem Saeed Al Abdouli

A Thesis Presented to the
Masdar Institute of Science and Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computing and Information Science

Abstract

Natural processes often generate some observations more frequently than others. These processes result in an unbalanced distributions which cause the classifiers to bias toward the majority class especially because most classifiers assume a normal distribution. The quantity and the diversity of imbalanced application domains necessitate and motivate the research community to address the topic of imbalanced dataset classification. Therefore, imbalanced datasets are attracting an incremental attention in the field of classification. In this work, we address the necessity of adapting data pre-processing models in the framework of binary imbalanced datasets, focusing on the synergy with the different cost-sensitive and class imbalance classification algorithms. The results of this empirical study favored the Synthetic Minority Over-sampling Technique (SMOTE) in the case of relativity high Imbalance Ratio (IR) and favored Neighborhood Cleaning Rule (NCL) in the case of relativity small IR. Further improvement was suggested to enhance NCL scalability with IR, and the proposed method is named NCL+. The outcomes showed that NCL+ outperformed NCL especially with the datasets of relatively high IR.

This research was supported by Masdar Institute of Science and Technology to help fulfill the vision of the late President Sheikh Zayed Bin Sultan Al Nahyan for sustainable development and empowerment of the UAE and humankind.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Problem Definition

Natural processes often generate some observations more frequently than others. Therefore, they produce samples that may not have a normal class distribution. The distribution could be close to normal. However, in other cases the class distribution could be highly imbalanced. Generally, most classification algorithms assume normal class distribution. However, in the case of imbalanced datasets, there are majority classes that bias the classifiers' decision. Classifiers tend to focus on the majority classes and ignore the minority classes. There are many cases in which the minority class represents the class of interest. The problem appears in several real-world data assembled from different real application areas. In some cases, such as (Johnson, Ryutaro, & Hoan, 2013), it is necessary to correctly classify the minority class. Generally, normal classifiers would misclassify many samples of the minority class that represents diseased trees. Because the imbalance distribution poses many challenges to widely used classifiers such as decision trees, induction models, and multilayer perceptrons (Jo & Japkowicz, 2004).

1.2 Motivation

Classifiers are developed to minimize the error rate. Consequently, they tend to be negatively affected by the majority class in the case of imbalanced data classification problems, and often perform poorly. A general-purpose framework is needed to handle the classification of imbalanced datasets. Therefore, in further details, this research addresses different data pre-processing models along with a diverse set of classifiers. The aim of this research is to find the best method to tune the combination of imbalanced sampling and classification paradigms with respect to the Imbalance Ratio (IR). The IR is defined as the ratio of the number of instances in the majority and the minority classes (Fernándeza, Garcíaa, Jesusb, & Herreraa, 2008).

Specifically, the objective of this research is to answer the following questions:

1. Should we sample the data in case of employing any class imbalance or cost-sensitive classifiers?
2. What is the best re-sampling technique that improves the classification when using class imbalance or cost-sensitive classifiers?
3. Could the successful re-sampling technique be further improved?

Those questions are answered using two empirical studies that will be discussed in further details in the following sections. This paper is organized as follows: Chapter 2 discusses the imbalanced data classification handling and the related work. Chapter 3 demonstrates the empirical studies conducted. While Chapter 4 includes the results and discussion. Finally, the conclusion summarizes the findings and suggestions.

CHAPTER 2

Dealing With Class Imbalance

The machine learning community addresses the issue of imbalanced dataset classification using two main approaches on two levels. In this section, the two main approaches are demonstrated. Furthermore, several data preprocessing techniques, which will be analyzed in later sections, are reviewed. The focus of this research is based on the second approach.

2.1 Method Level (Internal)

Many algorithms have been proposed to address the issue of class imbalance. Furthermore, algorithms have been modified to consider imbalanced datasets.

Specifically, the modification may focus on adjusting the cost function, changing the probability estimation, or adapting recognition-based learning (Fernándeza, Garcíaa, Jesusb, & Herreraa, 2008). The set of algorithms that work on the method level could be efficient. However, in many cases, these algorithms are application specific. Thus, they need special knowledge about the classifier and application (Fernándeza, Garcíaa, Jesusb, & Herreraa, 2008); (Galar, Fernandez, Barrenechea, Bustince, & Herrera, 2011).

2.2 Data Level (External):

Several pre-processing research papers such as (Leo, 1996) (Veropoulos, Campbell, & Cristianini, 2007) (Laurikkala, 2001) (Hu, Liang, He, & Ma, 2009) (Zhou & Liur, 2006) (Wang & Yao, 2009) have introduced and evaluated data sampling techniques. The purpose of pre-processing is to balance and normalize the class distribution before passing the dataset to the classifier. Sampling is the most commonly used approach for overcoming misclassification problems due to imbalanced data sets (Lokanayaki & Malathi, 2013). Sampling follows two basic approaches:

2.2.1 Over-Sampling

The models in this category modify the size of the minority class. The aim is to increase the samples of the minority class. Five different over-sampling models will be reviewed in this section.

2.2.1.1 Synthetic Minority Over-sampling Technique (SMOTE)

SMOTE is an over-sampling approach that is responsible for creating synthetic samples of the minority class using the feature space. Random samples are selected from the K nearest neighbor (Chawla, Bowyer, Hall, & Kegelmeyer, 2002).

The key factor of success of SMOTE is the broader decision regions of the minority class that is created using the nearby minority samples (Chawla, Bowyer, Hall, & Kegelmeyer, 2002).

2.2.1.2 SMOTE_TL

Synthetic Minority Over-sampling TEchnique + Tomek's SMOTE_TL is an integration of both SMOTE and TL. TL in this case operates as a cleaning method. Thus, instead of removing elements from the majority class, this model removes samples from both classes. The main objective of this technique is to balance the dataset and to enhance the class clusters (Batista, Prati, & Monard, 2004).

2.2.1.3 Selective Preprocessing of Imbalanced Data2 (SPIDER2)

The SPIDER2 consists of two main phases. In the first phase, the Edited Nearest Neighbor Rule ENNR is used to address the samples' local characteristic. Then in the second phase, the model removes the majority class samples that resulted in misclassification. Simultaneously, it performs a local over-sampling to the minority class (Napieralla, Stefanowski, & Wilk, 2010).

2.2.1.4 Random Over-Sampling (ROS)

ROS's main objective is to balance the data distribution by creating random replications of the samples of the minority class. The main challenge to ROS is overfitting, since it generates similar copies to the samples of the minority class (Fernándeza, Garcíaa, Jesusb, & Herreraa, 2008).

2.2.1.5 Adaptive Synthetic Sampling (ADASYN)

ADASYN is an adaptive model. There are two basic objectives of ADASYN that are adaptive learning and minimizing the imbalance ratio. So, based on the input the algorithm calculates the degree d of class imbalance. The d value is then compared to a threshold. If it passed he comparison, then the number of synthetic data examples is determined using $G = (m_l - m_s).\beta$ where m_l is the number of majority class samples, m_s is the number of minority class samples and $\beta \epsilon [0\ ,1]$ is a parameter that defines the required balance level after synthetic data is generated. When $\beta = 1$ a fully balanced dataset is produced. Afterwards, k's nearest neighbors are calculated using the Euclidean Distance function. The final step is to normalize by randomly selecting one minority data example from the k nearest neighbors (He, Bai, Garcia, & Li, 2008).

2.2.2 Under-Sampling

The aim of the under-sampling models is to reduce the size of the majority class set by removing some of majority class instances.

2.2.2.1 Neighborhood Cleaning Rule (NCL)

Neighborhood cleaning rule (NCL) is oriented toward employing Wilson's edited nearest neighbor rule ENN. In (Laurikkala, 2001), experimental results showed that the NCL contributed 20-30% improvement in imbalance classification. The NCL model maintains all the samples of the class of interest C and removes samples from the rest of the data O where $O = T - C$. This process is accomplished in two phases.

In the first phase, ENN is used to find the noisy data A_1 in O. Specifically, 3-ENN is used to remove, samples with a different class to the majority class of the three nearest neighbors, It removes samples that have different classes to at least two of their three nearest neighbors. Subsequently, the neighborhoods are processed again and a set A_2 is created. Then, the three nearest neighbor samples that belong to O and lead to C samples misclassification are inserted in the set A_2. Finally, the data is reduced by eliminating sampling that belongs to both sets A_1 and A_2 $A_1 \cup A_2$ (Laurikkala, 2001). Figure 2.1 shows the NCL algorithm.

1. Split data T into the class of interest C and the rest of data O.
2. Identify noisy data A_1 in O with edited nearest neighbor rule.
3. For each class C_i in O

   if ($x \in C_i$ in 3-nearest neighbors of misclassified $y \in C$)
   and ($|C_i|$ $0.5 \cdot |C|$) then $A_2 = \{ x \} \cup A_2$
4. Reduced data $S = T - (A_1 \cup A_2)$

Figure 2.1, NCL pseudo code **(Laurikkala, 2001)**

2.2.2.2 Condensed nearest Neighbor rule + Tomek links (CNN_TL)

The First Condensed Nearest neighbor rule CNN is applied to reduce the majority subset by removing samples that are far from the decision border. Then, Tomek links is applied to remove the noisy samples and the majority samples that are near the decision border (Fernándeza, Garcíaa, Jesusb, & Herreraa, 2008).

2.2.2.3 Class Purity Maximization (CPM)

The CPM algorithm follows a recursive procedure. At first, it defines two center point samples. One of them represents the minority class while the other represents the majority class. Afterwards, it uses those two points to partition that dataset into two clusters C_1 and C_2. Then, it calculates the impurity of each cluster. Finally, it makes a comparison between the parent's impurity and the resulting cluster impurity. It recursively calls itself until the stopping condition occurs. The stopping condition is reached when one of the clusters has less impurity than its parent or a singleton is reached (Yoon & Kwek, 2005).

2.2.2.4 Under-sampling Based on Clustering (SBC)

SBC adapts clustering methods to process the imbalanced class distribution. The algorithm follows two basic steps. First, it divides the training dataset into clusters. Then for each cluster, the ratio of each class is calculated and considered. If a cluster contains more minority classes than the majority, then it will act like the minority class. Likewise, if a cluster contains more majority classes than the minority, then it will act like the majority class (Yen & Lee, 2006) .

2.2.2.5 Tomek Links (TL)

TL uses a distance function to determine the noisy samples and remove the majority class samples that lay on the borderline of minority class. TL can be used as a cleaning method to remove~~s~~ data from both classes. So, given that two samples E_i and E_j from different classes and the distance between E_i and E_j is given by $d(E_i, E_j)$, then the pair (E_i, E_j) represent Tomek link if there is no sample E_x exist such as $d(E_i, E_x) < d(E_i, E_j)$ or $d\left(E_j, E_x\right) < d(E_i, E_j)$ (Batista, Prati, & Monard, 2004).

2.3 Evaluation Criteria in Imbalanced Domains

The evaluation process is a critical factor in assessing classifier's performance. In binary classification problems, the confusion matrix, shown in Table 2.1, is used to assess the performance. Accuracy is the most common measurement used to evaluate classifiers' performance. The accuracy can be calculated using the following formula:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP} + \mathrm{TN}}$$

where TP is the percentage of correctly classified positive instances, TN is the percentage of correctly classified negative instances, FP the percentage of misclassified positive instances, and FN the percentage of misclassified negative instances.

Generally in the case of imbalanced datasets the classifiers are biased towered the majority class. Assuming that negative is the majority class, then TN could be abnormally high in a normally distributed class dataset, depending on the Imbalance Ratio IR. The increase in TN will increase the classification accuracy, where, many of the minority class samples of interest are misclassified. Consequently, the accuracy could be an unreliable measurement for imbalanced dataset classification. To get a better understanding, consider a case of 100 samples in a dataset in which 97% are negative and 3% positive. Generally, predicting the majority would produce 97% accuracy. However, this strategy is not accurate enough to classify the positive class instances.

Table 2.1: Confusion Matrix for Binary Classification

	Positive Prediction	Negative Prediction
Positive Class	True Positive TP	False Negative FN
Negative Class	False Positive FP	True Negative TN

Alternatively, the Area Under the receiver operating characteristic ROC Curve (AUC) could be used to provide the necessary measurements to evaluate the imbalanced data classification (Ling & Li, 1998; Provost & Fawcett, 2001). The AUC provides the best criteria that suites this type of evaluation. It relies on the ROC that evaluates the algorithms' ability to correctly classify samples relative to each other. Specifically, the ROC provides a visualization of the relationship between TP_{rate} and FP_{rate}. The AUC can be measured by calculating the area under the ROC curve. The AUC of a perfect model is 1.

CHAPTER 3

Experimental Study

This section describes the empirical study that was carried out in two phase. In the first phase, a comparison between 11 classifiers was conducted. Machine Learning Repository UCI's imbalanced datasets (Center of Machine Learning and Intelligent, 2014) were trained using different cost-sensitive and class ensembles algorithms under 3 different data pre-processing scenarios. The main target of this phase is to find the best combination of data pre-processing and classification procedures that best suites the binary imbalanced data classification with respect to IR, number of instances, and number of attributes. The second phase of this experimental study was oriented to study and suggest improvements of a successful pre-processing model resulting from the first phase.

Ten real-world datasets with different Imbalance Ratios (IR) obtained from (Alcalá-Fdez, et al., 2011) were considered in this empirical study. The focus is on two-class imbalanced datasets. Thus there are only two classes, a positive and a negative. The majority class is represented by the negative class label whereas the minority class is represented by the positive class label. Furthermore, this study considers the IR. The datasets processed in this study range~~s~~ from low IR to high. Thus, they were categorized into two main groups based on the IR level. The datasets that had an IR between 1.5 and 9 were represented in the low IR category. Whereas, the datasets with an IR greater than 9 were represented in the high IR category. The datasets have different number of instances, features, and feature types. Furthermore, all the sets require a binary classification, since they are composed of two main classes, a negative majority class and a positive minority class.

Table 3.1 below summarizes the datasets' specifications for each set. It shows the IR category and ratio and identifies the minority and majority classes and their relative ratios. It also shows the number of features and instances. The table is sorted in ascending order based on IR.

Table 3.1: Datasets specification summary

DataSet Name	Imbalance ratio	IR	number of Features	Attribute Type	number of instances	Missing Values	Class(Min,Maj)
ecoli-0_vs_1	between 1.5 and 9	1.86	7	Real	220	No	im, cp
Wisconsin	between 1.5 and 9	1.86	9	Integer	683	YES	malignant,benign
vehicle1	between 1.5 and 9	3.23	18	Integer	846	No	Van, remainder
New-thyroid2	between 1.5 and 9	5.14	5	Integer/Real	215	No	hyper,remainder
page-blocks0	between 1.5 and 9	8.77	10	Integer/Real	5472	No	remainder, text
Vowel0	>9	10.1	13	Integer/Real	988	No	hid,remainder
Glass5	>9	15.47	9	Real	214	No	containers,remainder
Glass6	>9	22.81	9	Real	214	No	tableware,remainder
yeast6	>9	32.78	8	Real	1484	No	ME1, remainder
abalone19	>9	128.87	8	Real/ Nominal valued	4174	No	19, remainder

Note: positive = minority and negative = majority

3.1 Experimental Design

This research focused on addressing the synergy between cost-sensitive and ensembles for class imbalance classifiers and different pre-processing techniques. The target was to examine the performance of those classifiers under different data pre-processing techniques.

In this research a 5-fold cross validation was used. Thus, training and test sets were partitioned into five training and test sets. In the results, the average of the five data/test set partitions were considered for each dataset.

The design of this empirical study can be summarized as the following. In total, 11 5-fold CV trials were conducted for 10 different datasets resulting in 550 training datasets. These datasets were examined under 11 different pre-processing conditions, no-sampling, under-sampling, and over-sampling resulting in 6050 processed datasets.

The trials were conducted using different datasets with different IRs and number of instances. The experiment was carried out using 11 classifiers categorized into two main classification categories. The first group is cost-sensitive classification. In this study, 3 classifiers from the cost-sensitive classification were deployed, namely C4.5 Cost-Sensitive (C4.5CS), Multilayer Perceptron with Backpropagation Training Cost-Sensitive (NNCS), and SVM Cost-Sensitive (SVMCS). The remaining 8 classifiers were deployed from the second category named ensembles for class imbalance. The eight classifiers used were, Adaptive Boosting with C4.5 Decision Tree as Base Classifier (AdaBoost), Adaptive Boosting Second Multi-Class Extension with C4.5 Decision Tree as Base Classifier (AdaBoostM2) , Cost Sensitive Boosting with C4.5 Decision Tree as Base Classifier (AdaC2) , Bootstrap Aggregating with C4.5 Decision Tree as Base Classifier (Bagging) , Over-sampling Minority Classes Bagging 2 with C4.5 Decision Tree as Base Classifier (OverBagging2) , Modified Synthetic Minority Over-sampling TEchnique Bagging with C4.5 Decision Tree as Base Classifier (MSMOTEBagging) , Under-sampling Minority Classes Bagging 2 with C4.5 Decision Tree as Base Classifier (UnderBagging2) , and Under-sampling Minority Classes Bagging to Over-sampling Minority Classes Bagging with C4.5 Decision Tree as Base Classifier (UnderOverBagging). The classifiers are discussed in further details below.

3.1.1 Cost-Sensitive Classification

Cost-sensitive learning is a process of inducing models from imbalanced distributed data. It quantifies and processes the imbalance. There are 9 main cost types that cost-sensitive learning models depend on for reducing the total classification cost. The most researched two types are misclassification total cost and test cost (Qin, Zhang, Wang, & Zhang, 2010). The following is a brief description of the three classifiers used from this category.

3.1.1.1 C4.5CS

This classifier is an instance–weighting cost-sensitive model. Depending on the greedy divide-and-conquer technique, it produces considerably smaller trees. It concentrates on total misclassification cost reduction. Additionally, it targets the cost of the size of the tree and the quantity of high cost errors. It is mainly designed for binary classification problems. The results showed that although C4.5CS has more total misclassification errors than C5 as shown in Table 3.2 below, which is 0.07 on average, C4.5CS is less likely to make high cost errors (Ting, 2002).

Table 3.2: Average misclassification costs (Ting, 2002)

Dataset	C4.5CS	C5	CART	Discrim	NaiveBayes
Heart	0.404	0.430	0.452	0.393	0.374
German Credit	0.303	0.304	0.613	0.535	0.703

Table 3.3 demonstrates the specification used in this study.

Table 3.3: Parameter specification for C4.5CS

Parameters	Value
Prune	True
Confidence Level	0.25
Instances per leaf	2
Minimum Espected Cost	True

3.1.1.2 Neural Networks Cost-Sensitive (NNCS)

The cost-sensitive neural networks concentrates on minimizing the total misclassification cost. The original backpropagation learning process consists of a multilayered feed-forward neuron network that implements the backpropagation technique to achieve the weight gradient descent. However, this procedure alone is not sufficient for the cost-sensitive classification performance. Therefore, a modification in the probability estimates of the network throughout the test phase was suggested. The probability P(i) that a sample belongs to a class i is modified to account for misclassification cost using $\mathrm{P}'(\mathrm{i}) = \frac{\mathrm{CostVector[i]P(i)}}{\sum_{\mathrm{j}} \mathrm{CostVector[i]P(i)}}$. The modified probability promotes the class of higher estimated misclassification costs (Kukar & Kononenko, 1998). (Zhou & Liur, 2006) empirical study revealed that generally soft-ensemble and threshold-moving improve the cost-sensitive neural networks training. Table 3.4 shows the parameter specification used for NNCS.

Table 3.4: Parameter specification for NNCS

Parameters	Value
Hidden Layers	2
Hidden Nodes	15
Transfer	Htan
Eta	0.15
Alpha	0.10
Lamda	0.0
Cycles	10000
Improve	0.01

3.1.1.3 SVM Cost-Sensitive (SVMCS)

A Support Vector Machine (SVM) was successfully applied to many classification problems. However, it is negatively affected by the majority class in case of imbalanced datasets. Consequently, a cost-sensitive SVM was developed to overcome the biased classification produced by the imbalance class distribution and to lower the error rate and misclassification cost. The cost-sensitive SVM formula targets two types of errors using two loss functions. SVMCS follows this formula (Cao,Zhao, & Zaiane, 2013; Zheng, Zou, Sun, & Chen, 2011; Veropoulos, Campbell, & Cristianini, 2007).

$$R(w, \xi) = \frac{1}{2}||w||^2 + C\left(\sum_{i=1}^{k} \xi_i\right),$$

$$s.t\ y_i(x_i.w + b) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, \ldots, k$$

where k denotes the sample number, x_i denotes the predictive attribute, $||w||^2$ estimates the complexity of the model, $\sum_{i=1}^{n} \xi_i$ estimates the training error, whereas the constant C controls the tradeoff between the model complexity and the training error. Table 3.5 shows the parameter specification related to SVMCS classifier.

Table 3.5: Parameter specifications for SVMCS

Parameters	Value
KERNEL type	POLY
C	100.0
eps	0.001
degree	1
gamma	0.01

3.1.2 ENSEMBLES FOR CLASS IMBALANCE

Ensembles aim to enhance a classifier's accuracy. They are developed to merge the output of collection of training classifiers into a single output. The proposed taxonomy for the employed algorithms from this category is described in Figure 3.1 below.

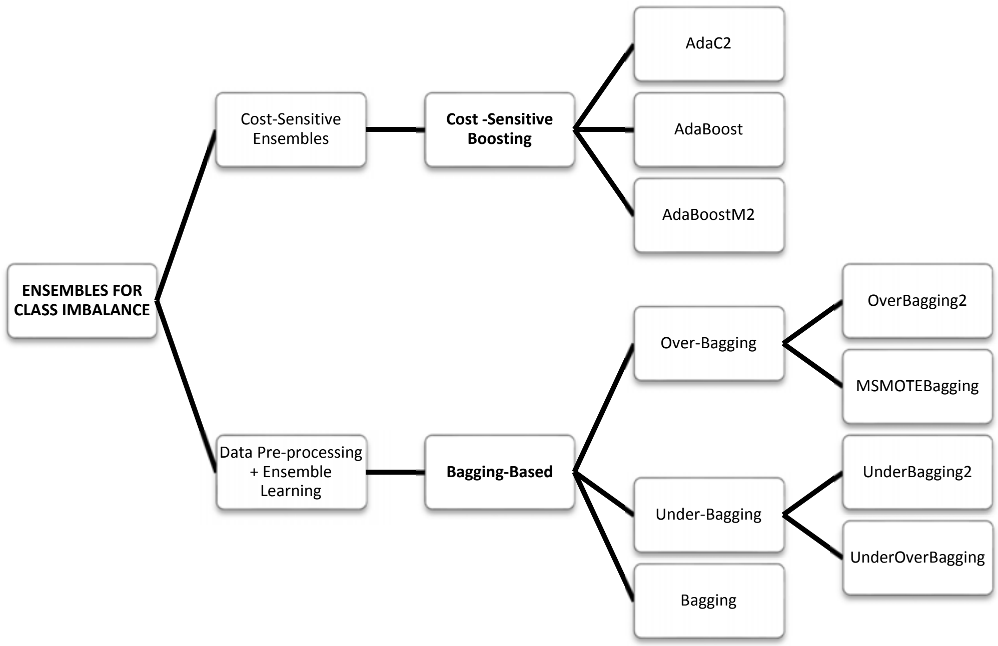

Figure 3.1, "Ensembles for class imbalance" algorithms proposed taxonomy

3.1.2.1 Boosting-Based Cost-Sensitive Ensembles

The boosting-based algorithms were added to this category. The boosting-based algorithms embed data preprocessing procedures into boosting algorithms. After each round, the boosting-based algorithms modify the distribution of the weight toward the minority class (Galar, Fernandez, Barrenechea, Bustince, & Herrera, 2011). AdaC2, AdaBoost, and AdaBoostM2 are included in this category.

3.1.2.1.1 Adaptive Boosting with C4.5 Decision Tree as Base Classifier (AdaBoost)

This classifier gradually improves the classification after each iteration. It uses the whole training dataset to improve the classification for each misclassified instance after each iteration by increasing their weights. On the other hand, the classifier decreases the weights of the properly classified instances. Consequently, it changes the distribution of the training dataset (Galar, Fernandez, Barrenechea, Bustince, & Herrera, 2011; Hu, Liang, He, & Ma, 2009).

3.1.2.1.2 Cost Sensitive Boosting with C4.5 Decision Tree as Base Classifier (AdaC2)

The AdaC2 algorithm is a Boosting-based. Its weight function was edited to consider the cost. The weight function was replaced by the following formula:

$$D_{t+1}(i) = C_i\, D_t(i).\, e^{-\propto_t h_t(x_i)\, y_i} \text{ where } \propto_t = \frac{1}{2} \ln \frac{\sum_i y_i = h_t(x_i)\; C_i\, D_t(i)}{\sum_i y_i \neq h_t(x_i) C_i\, D_t(i)}$$

Table 3.6 shows the parameter specifications of AdaC2.

Table 3.6: Parameter specifications of AdaC2

Parameters	Value
Pruned	True
Confidence	0.25
Instances per Leaf	2
Number of Classifiers	10
Cost Setup	Adaptive
Cost Majority Class	0.25
Cost Minority Class	1

3.1.2.1.3 Adaptive Boosting Second Multi-Class Extension with C4.5 Decision Tree as Base Classifier (AdaBoostM2)

Table 3.7 shows a description of parameter specification of AdaBoostM2.

Table 3.7: Parameter specification of AdaBoostM2

Parameters	Value
Pruned	True
Confidence	0.25
Instances per Leaf	2
Number of Classifiers	10
Train Method	No-Resampling
Cost Majority Class	0.25
Cost Minority Class	1

3.1.2.2 Bagging-based Ensembles

Bootstrap aggregating based methods rely on producing different predictor versions and use them to construct an aggregated predictor. Thus, the methods use a bootstrapped replicate~~s~~ of the original dataset to train the classifiers. In Bagging, the original dataset is randomly sampled and passed to the classifiers. However, in overBagging an over-sampling teqnique is embeded in data pre-processing procedure. Additionally, in underBagging, an under-sampling teqnique is used to process the dataset (Leo, 1996 ;Wang & Yao, 2009). Generally, a 10 bootstrap replicate~~s~~ is enough to improve the classification. Finally, the bootstrap aggregating methods use a plurality vote to determine the predicting class (Leo, 1996). In summary, they follow three basic steps that are re-sampling, building ensembles, and vote.

The algorithms of this category do not need weight computation. Therefore, they are easier to integrate with data pre-processing methods (Galar, Fernandez, Barrenechea, Bustince, & Herrera, 2011).

The general parameter specification used for Bagging algorithms is shown in Table 3.8.

Table 3.8: Parameter specification for all the Bagging Algorithms used in this study

Parameters	Value
Pruned	True
Confidence	0.25
Instances per Leaf	2
Number of Classifiers	10

3.1.2.2.1 Bagging

The following is the procedure of Bagging algorithm.

Input: S: Training set; T: Number of iterations;
n: Bootstrap size; I: Weak learner

Output: Bagged classifier: $H(x) = sign\left(\sum_{t=1}^{T} h_t(x)\right)$ where $h_t \in [-1, 1]$ are the induced classifiers

for $t = 1$ to T **do**
  $S_t \leftarrow$ RandomSampleReplacement(n,S)
  $h_t \leftarrow$ I(S_t)
end for

Figure 3.2, Bagging pseudo code **(Leo, 1996)**

3.1.2.2.2 Over-sampling Minority Classes Bagging 2 with C4.5 Decision Tree as Base Classifier (OverBagging2)

OverBagging is a special case of Bagging. In OverBagging the datasets are pre-processed using an over-sampling technique instead of random sampling (Wang & Yao, 2009). In this case, the re-sampling doubles the size of the negative instances in the processed dataset (Galar, Fernandez, Barrenechea, Bustince, & Herrera, 2011).

3.1.2.2.3 Modified Synthetic Minority Over-sampling TEchnique Bagging with C4.5 Decision Tree as Base Classifier (MSMOTEBagging)

MSMOTE is a modified version of SMOTE and is designed to use the feature space to generate synthetic instances of the minority class. Likewise, MSMOTE is designed to generate the synthetic instances of the minority class. The main difference is that MSMOTE uses the samples' type as a selection criteria to select its nearest neighbors (Hu, Liang, He, & Ma, 2009). The MSMOTE procedure is shown in Figure 3.3 below. MSMOTEBagging is an integration of both MSMOTE and Bagging procedures. In (Hu, Liang, He, & Ma, 2009), their empirical study showed that the performance of MSMOTE outperformed the performance of SMOTE.

```
Algorithm MSMOTE(L,T, N, k)
Input: All the samples L, The minority class samples T; Amount of SMOTE N%; Number of nearest neighbors k
Output: synthetic minority class samples (N%*T)
1 k = Number of nearest neighbors
2 N=N%*T //Number of generating samples
3 numattrs = Number of attributes
4 Sample[ ][ ]: array for original minority class samples
5 newindex: keeps a count of number of synthetic samples generated, initialized to 0
6 Synthetic[ ][ ]: array for synthetic samples
(Compute k nearest neighbors for each sample)
7 for i • 1 to T//(Number of the minority class)
8 Compute k nearest neighbors for i, and save the indices in the nnarray and judge the type of this sample
9 If (type!=0) // 0 ,latent noises
10 Populate(N, i, nnarray， type)
11 endfor
12 Populate (N, i, nnarray， type) // (Function to generate the synthetic samples.)
13 while N _= 0
14 If (type==1) //1:secutity samples 2 border samples
15 This step randomly chooses one of the k nearest neighbors of i. call it nn.
16 else
17 This step chooses the nearest neighbors of i., call it nn.
18 for attr • 1 to numattrs
19 Compute: dif = Sample[nnarray[nn]][attr] • Sample[i][attr]
20 Compute: gap = random number between 0 and 1
21 Synthetic[newindex][attr] = Sample[i][attr] + gap* dif
22.endfor
23 newindex++
24 N = N • 1
25 endwhile
26 return// (End of Populate.)
```

Figure 3.3, MSMOTE pseudo code **(Hu, Liang, He, & Ma, 2009)**

3.1.2.2.4 Under-sampling Minority Classes Bagging 2 with C4.5 Decision Tree as Base Classifier (UnderBagging2)

UnderBagging2 is a Bagging procedure in which the re-sampling model doubles the size of the positive instances in the processed dataset.

3.1.2.2.5 Under-sampling Minority Classes Bagging to Over-sampling Minority Classes Bagging with C4.5 Decision Tree as Base Classifier (UnderOverBagging)

UnderOverBagging is a bagging model in which the instances of every bag are treated using either under-sampling or oversampling technique.

3.2 Experiment Framework

The data was examined under 3 conditions. The first one is without implementation of any data pre-processing or sampling techniques. The second condition used 5 different over-sampling techniques**:** ADAptive SYNthetic Sampling (ADASYN), Random over-sampling (ROS), Synthetic Minority Over-sampling Technique (SMOTE), Synthetic Minority Over-sampling TEchnique + Tomek's modification of Condensed Nearest Neighbor (SMOTE_TL), and Selective Preprocessing of Imbalanced Data 2 (SPIDER2). The third condition used 5 under-sampling techniques: Neighborhood Cleaning Rule (CNNTL), Class Purity Maximization (CPM), Neighborhood Cleaning Rule (NCL), Undersampling Based on Clustering (SBC), and Tomek's modification of Condensed Nearest Neighbor (TL).

The framework described above is showed in Figure 3.4 below.

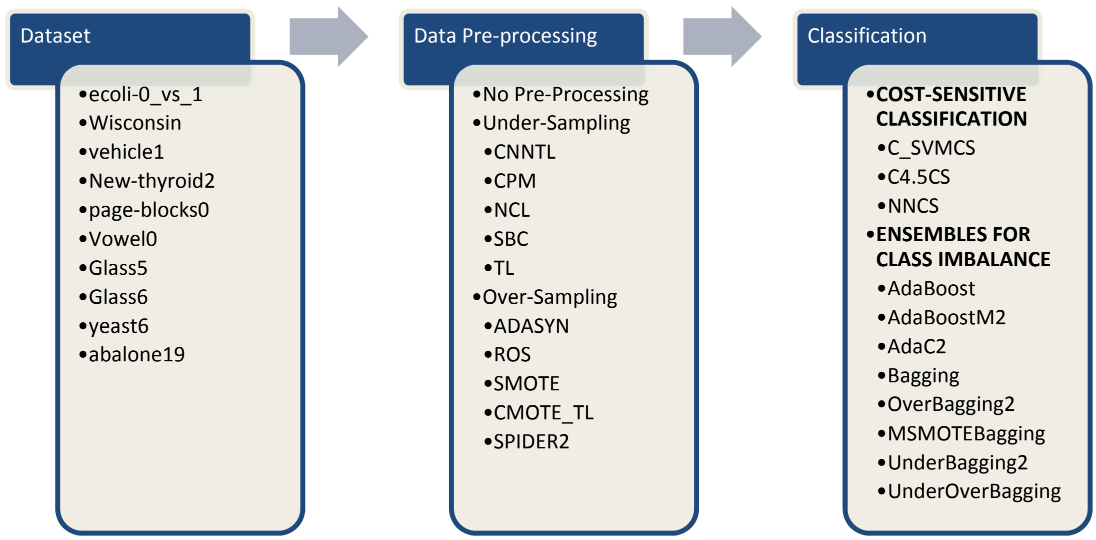

Figure 3.4, Framework of experiment

3.3 Validation

In order to evaluate the association of pre-processing techniques and the classifiers' performance, the five-fold cross validation CV and the global classification Area Under the ROC Curve AUC were calculated. The AUC evaluates the classifier's ability to separate between the positive and negative classes. Therefore, it is the best measurement of the imbalanced data classification performance; consequently it is the best measurement that suites this experiment.

3.4 Experimental Results

Depending on the AUC measurement, the performance of each sampling technique was investigated. The following is the average performance of non-Sampling, over-Sampling, and under-Sampling, calculated the average of all sampling techniques for each dataset using the best performance of each combination of sampling and classification model for each sampling technique.

The study revealed that the performance of pre-processing models are mainly affected by the degree of imbalance of the dataset. It also showed that the data pre-processing models was not significantly affected by the number of instances. Generally, the NCL performed the best in datasets with relatively small IR. This result confirms the results of (Laurikkala, 2001) that argued that NCL improves small class modeling. On the other hand, SMOTE and SMOTE_TL performed best in relatively high IR datasets. Table 3.9 shows the best combination of both data preprocessing models and classifiers for each dataset. The table is sorted in ascending order based on IR value.

Table 3.9: Data pre-processing and imbalanced models
the best performance for each dataset

DATASET	IR	# OF INSTANCES	BEST COMBAINATION
ecoli-0_vs_1	1.86	220	NCL + Bagging
Wisconsin	1.86	683	NCL + AdaC2
vehicle1	3.23	846	TL + SVMCS
New-thyroid2	5.14	215	NCL + SVMCS
page-blocks0	8.77	5472	
Vowel0	10.1	988	ADASYN + [AdaBoost, AdaBoostM2, AdaC2]
Glass5	15.47	214	CNN_TL + AdaC2
Glass6	22.81	214	SMOTE_TL + AdaC2
yeast6	32.78	1484	SMOTE + SVMCS
abalone19	128.87	4174	SMOTE +NNCS

Figure 3.5 below shows an evaluation of the performance of different classifiers against non-sampled imbalanced dataset. The results emphasize the negative effect of IR on a non-preprocessed dataset. The figure also shows that the number of features and the sample size contribute to the classification performance , the possible reason for the drop in balance in the third dataset vehicle1, IR 3.23, is the relatively high number of features and instances, 18 feature and 846 instances.

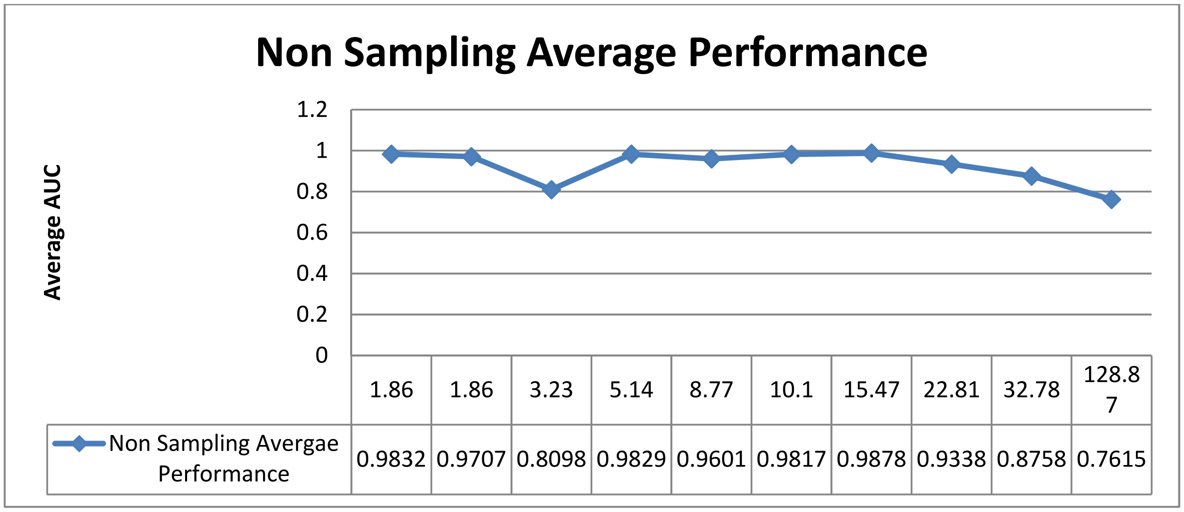

Figure 3.5, Non-sampling average performances

The over-Sampling average performance showed an overall similar curve to the non-sampled datasets. On the other hand, the under-sampling performance demonstrated lower overall performance. However, unlike the non-sampled and oversampled datasets, it showed a steady performance against relatively high IR as shown in Figure 3.6.

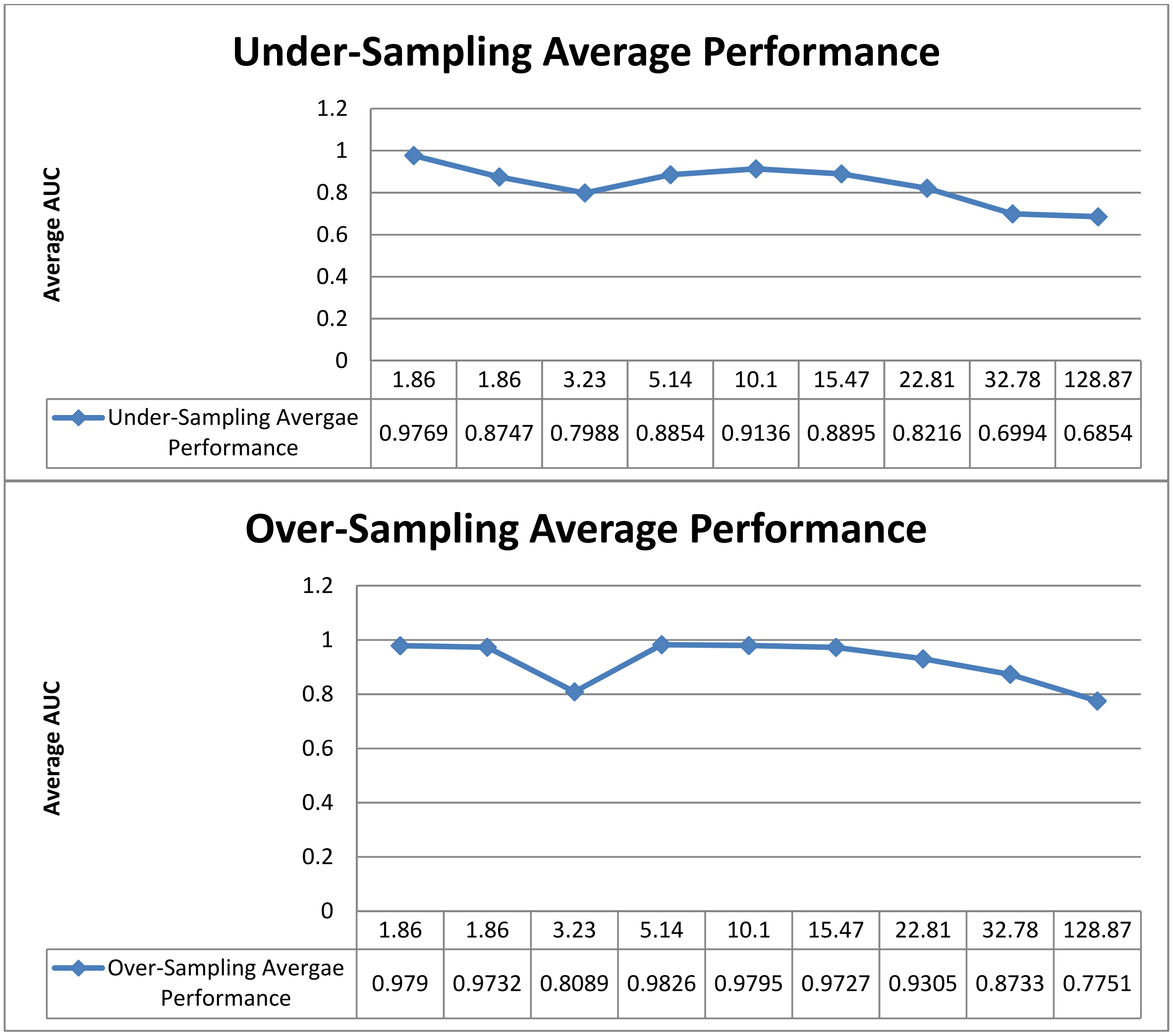

Figure 3.6, Over-sampling and under-sampling average performances

The results demonstrated above and the results shown earlier were motivation to investigate and improve the under-sampling model NCL. A possible reason for the lower performance is that the NCL model removes an informative data more than needed. The model shows a high performance in relatively small IR. The objective is to introduce enhancments to make it scale better with IR. In the following chapter, the proposal (named NCL+) is discussed in detailes. Then an empirical study is conducted to evaluate the relative performance of the proposal to the original model.

CHAPTER 4

Proposed NCL+ Method

4.1 NCL+ Method

The proposed improvement of NCL follows similar architecture of the NCL. The data reduction process will be held in two phases. In the first phase, the ENN will be used to create a subset named A_1. The second phase will be modified to employ an evolutionary instance selection algorithm CHC to create the second subset named A_2. Finally, $A_1 \cup A_2$ will be removed from the original dataset as illustrated in Figure 4.1.

NCL+ Pseudo Code

1. Split the dataset T into the class of interest C and the rest of data O
2. For the subset O do the following:
    1. Using ENN identify the noisy data and insert them to subset A_1
    2. Run CHC with the following spesifications:
        Euclidean Distance Function
        Fitness Function:
        $Fitness\ (S) = \propto . Clas_{rate} + (1 - \propto). perc\_red$
        where: S is a subset of the data set
        Alfa equilibrate factor $\propto = 0.5$
        Clas_rate : classification rate
        Perc_red: persentage of reduction
    3. insert the result into subset A_2
    4. Reduce the dataset $\mathrm{S} = \mathrm{T} - (A_1 \cup A_{2)}$

Figure 4.1, Pseudo code of proposed NCL+ method

In many cases there could be too much data. However, generally, the data is not equally informative during the training phase. Thus, algorithms such as the CHC (Eshelman, 1990) were developed to interpret the data independently of their location in the search space. Also, the CHC chooses the most representative instances. Consequently, it gains high reduction rates while maintaining the accuracy. Furthermore, (Cano, Herrera, & Lozano, 2003) showed that CHC gained the best ranking in data reduction rates.

The CHC relies on reducing the data by means of evolutionary algorithm EA and instance selection IS. The EAs are adaptive models that rely on the principle of natural evolution. In the CHC, an EA is used as instance selector to select the data to be removed. The decision-tree induction algorithm C4.5 is built using the selected instances. Then the new examples are classified using the resultant tree.

In particular, during every generation, the CHC follow some basic steps that can be summarized as the following: First, it generates an intermediate population of size N using the parent population of size N. Then it randomly pairs them and use them to produce N potential offspring. Then a survival competition is held in order to select the next generation population. The best N from the parent population and offspring are selected to form the next generation (Cano, Herrera, & Lozano, 2003).

Table 4.1 shows the CHC parameters used in the empirical study

Table 4.1: CHC parameters

Population Size	50
Number of Evaluations	10000
Alfa Equilibrate Factor	0.5
Percentage of Change in Restart	0.35
0 to 1 Probability in restart	0.25
0 to 1 Probability in Diverge	0.05
Number of Neighbors	3
Distance Function	Euclidean

4.2 Experimental Results on NCL+

The initial results showed a recognizable improvement in the performance for the NCL+ over the NCL with respect to IR. In some cases, the performance was close. However, the target was to improve the model to enhance its scalability with respect to IR. Table 4.2 shows a comparison between the two models and the improvement.

Table 4.2: Performance using relatively high IR dataset

Data Set	Abalone19	
IR	128.87	
# of Instances	4147	
Model	AUC_{st} NCL	AUC_{st} NCL+
C_SVMCS-I	0.760457	0.797602
C4.5CS-I	0.548425	0.799648
NNCS-I	0.507473	0.508451
AdaBoost-I	0.515339	0.799035

AdaBoostM2-I	0.51558	0.799155
AdaC2-I	0.554457	0.799648
Bagging-I	0.5	0.5
OverBagging2-I	0.529512	0.791638
MSMOTEBagging-I	0.579465	0.795147
UnderBagging2-I	0.71252	0.732963
UnderOverBagging-I	0.547419	0.791638
AVERAGE	0.570059	0.73772

Results also suggest that under-sampling could be further improved using adaptive learning and evolutionary training set selection algorithms such as CHC, Generational Genetic Algorithm for Instance Selection GGA, and Population-Based Incremental Learning PBI. Figure 4.2 demonstrates the difference between NCL and NCL+. It shows the average AUC_{st} for both models over the 11 classifiers and under the 11 previously reviewed sampling methods. Table 4.3 shows the average numerical values of $\mathrm{AUC}_{st}$of both NCL and NCL+

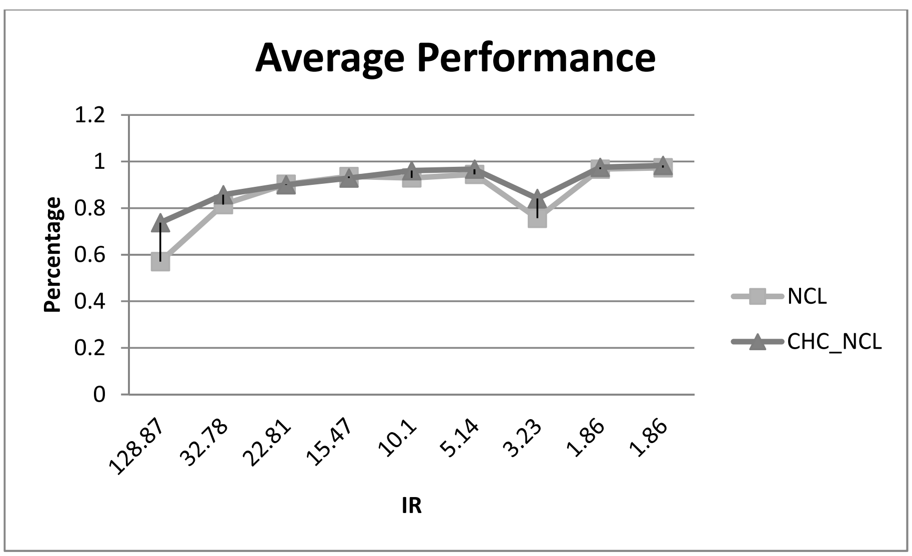

Figure 4.2, Average Performance of NCL and the NCL+.

Table 4.3: NCL and NCL+ Average performance

IR	**128.87**	**32.78**	**22.81**	**15.47**	**10.1**	**5.14**	**3.23**	**1.86**	**1.86**
NCL	0.570059	0.815282	0.902482	0.935809	0.929885	0.944949	0.756345	0.96757	0.973403
NCL+	0.73772	0.85761	0.900066	0.929047	0.961053	0.96728	0.841288	0.975394	0.983174

Looking at the data distribution of processed data of both models and given the dataset size. It is noticeable the CHC does not operate on quantity. It removed relatively few samples. However, the removed samples noticeably enhanced the classification. The difference in data distribution is shown in both Figure 4.3 and Figure 4.4.

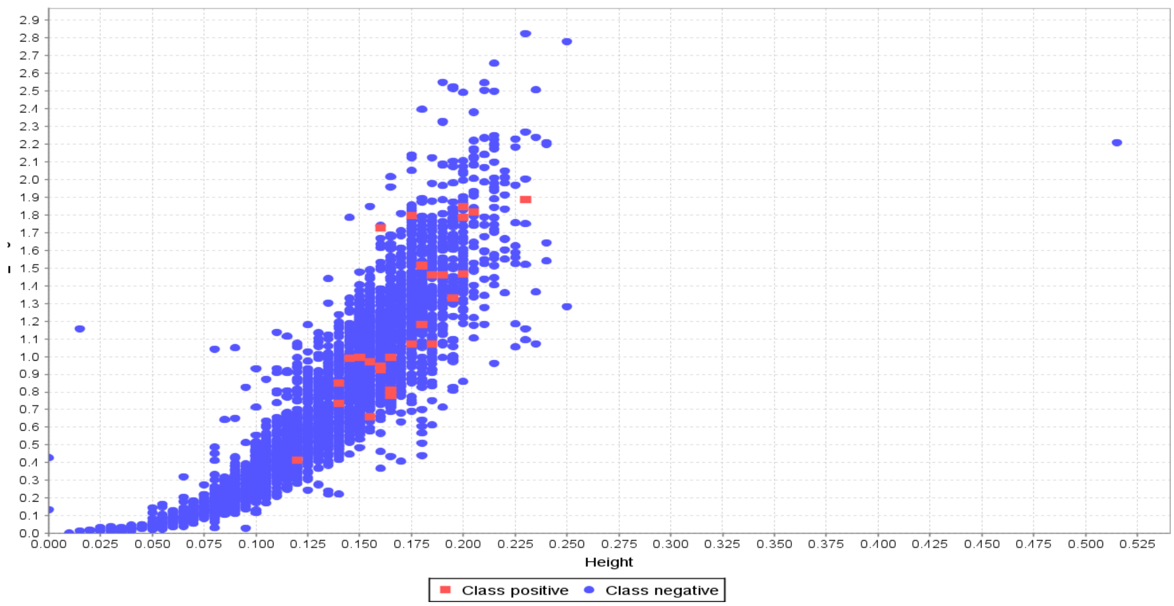

Figure 4.3, Class distribution using the NCL+

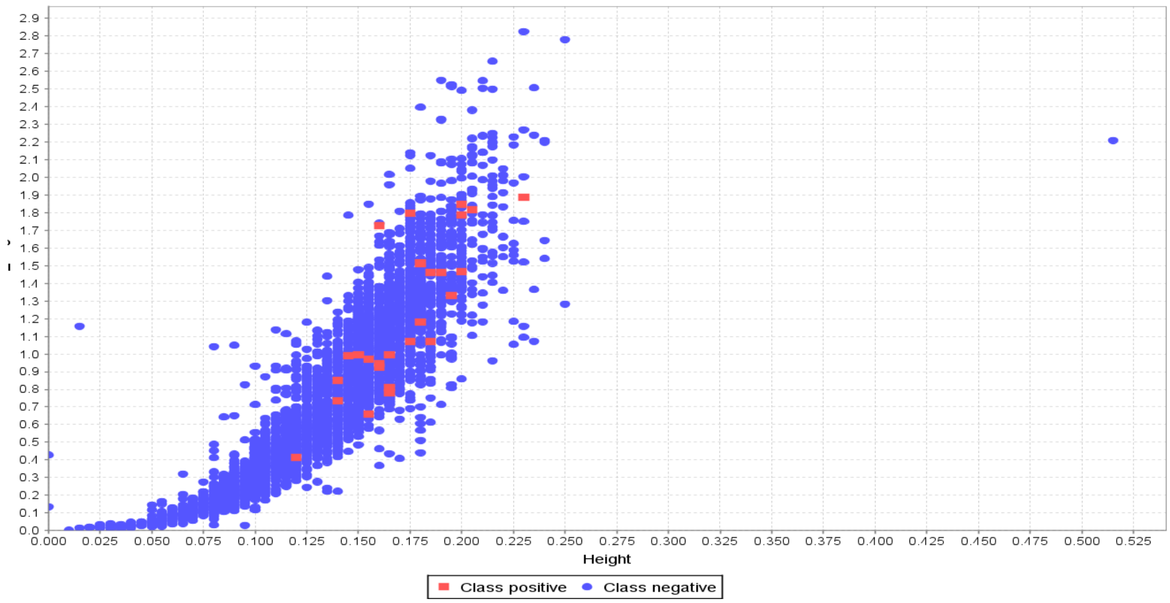

Figure 4.4, Class distribution using the NCL

CHAPTER 5

Conclusion

In this research, an empirical study was conducted using 11 UCI datasets with 11 classifiers under 3 different data pre-processing methods. The results suggested that the IR influences the performance of the preprocessing models. Basically, NCL was successfully operating on datasets that had a relatively small IR. Then, an improvement to the NCL was suggested, to scale better with IR. Subsequently, a comparison study between the NCL and the proposed improvement was conducted. The results showed the suggested improvement outperformed the NCL on relatively larger IRs. This research was focusing on imbalanced binary datasets. However, there is a difference between a binary and a multiclass classification. Thus, for a future work it is planned to study the effect of multiclass datasets with different IRs on reviewed combinations of pre-processing and classification models.

APPENDIX A

Detailed Experimental Results

A.1 Performance of Different Classifiers on Different Datasets

The results in all the tables included in the appendix will be sorted according to the following classifiers

Table 6.1: Classifiers and their codes (for future references in the later tables).

Code	Classifier
C1	C_SVMCS-I
C2	C4.5CS-I
C2	NNCS-I
C3	AdaBoost-I
C4	AdaBoostM2-I
C5	AdaC2-I
C6	Bagging-I
C7	OverBagging2-I
C8	MSMOTEBagging-I
C9	UnderBagging2-I
C10	UnderOverBagging-I
C11	MAX AUC for each sampling Technique

Abalone19

Table 6.2: Abalone19 without sampling

Classifier	WITHOUT-SAMPLING
C1	0.761543276
C2	0.570075469
C2	0.499617743
C3	0.498672663
C4	0.499034545
C5	0.508464013
C6	0.5
C7	0.53542739
C8	0.551684349
C9	0.641058473
C10	
C11	0.761543276

Table 6.3: Abalone19 over-sampling

Classifier	ADASYN-I	ROS-I	SMOTE-I	SMOTE_TL-I	SPIDER2-I
C1	0.057461543	0.7642	0.765444119	0.764236535	0.7760704
C2	0.546253354	0.559257	0.596327757	0.606001539	0.501096
C2	0.764806039	0.681044	0.789638168	0.780828008	0.5894157
C3	0.507283797	0.496741	0.538566904	0.536273092	0.4986722
C4	0.507405007	0.494809	0.538808304	0.550559243	0.4986722
C5	0.50692177	0.496741	0.537119231	0.535219331	0.5262114
C6	0.529271612	0.536031	0.52676843	0.555515305	0.5152187
C7	0.529271612	0.536031	0.52676843	0.556722306	0.5496254
C8	0.673826735	0.67515			0.5662039
C9	0.529271612	0.536031	0.52676843	0.538485737	0.5741403
C10	0.52057698	0.566655	0.55032469	0.529674702	0.5023347
C11	0.764806039	0.7642	0.789638168	0.780828008	0.7760704

Table 6.4: Abalone19 under-sampling

Classifier	CNNTL-I	CPM-I	NCL-I	RUS-I	TL-I
C1	0.552891	0.603229235	0.760457	0.646771	0.761423
C2	0.486525	0.509782754	0.548425	0.547487	0.568988
C2	0.54408	0.564040943	0.507473	0.592433	0.616639
C3	0.442944	0.501729844	0.515339	0.672402	0.5157
C4	0.442944	0.502333709	0.51558	0.672402	0.51558

C5	0.548913	0.512158087	0.554457	0.672402	0.511232
C6	0.575822	0.5	0.5	0.657494	0.5
C7	0.550602	0.549388809	0.529512	0.657494	0.520297
C8	0.54101	0.52121739	0.579465	0.714439	0.583001
C9	0.553289	0.574570541	0.71252	0.657494	0.657445
C10	0.58764	0.474540655	0.547419	0.692468	0.548265
C11					

ecoli-0_vs_1

Table 6.5: ecoli-0_vs_1 without sampling

Classifier	0.979647
C1	0.983218
C2	0.979647
C2	0.969179
C3	0.972627
C4	0.969179
C5	0.983218
C6	0.979647
C7	0.983218
C8	0.969302
C9	0.969302
C10	
C11	

Table 6.6: ecoli-0_vs_1 over-sampling

Classifier	ADASYN-I	ROS-I	SMOTE-I	SMOTE_TL-I	SPIDER2-I
C1	0.969179	0.983218	0.979647	0.976199	0.969178982
C2	0.948596	0.97275	0.983218	0.976076	0.958587849
C2	0.954663	0.969286	0.972627	0.966254	0.948596059
C3	0.955263	0.965731	0.972627	0.969179	0.979770115
C4	0.955263	0.965731	0.972627	0.965608	0.979770115
C5	0.955263	0.965731	0.972627	0.87275	0.972627258
C6	0.958711	0.983218	0.97977	0.979647	0.972627258
C7	0.958711	0.983218	0.97977	0.976076	0.965607553
C8				0.969302	0.958711002
C9	0.958711	0.983218	0.97977	0.976076	0.962159278
C10	0.944918	0.969302	0.969302	0.969302	0.958711002
C11					

Table 6.7: ecoli-0_vs_1 under-sampling

Classifier	CNNTL-I	CPM-I	NCL-I	RUS-I	TL-I
C1	0.973103	0.953044	0.976199	0.970302	0.979647
C2	0.927693	0.968933	0.958588	0.979647	0.962159
C2	0.915993	0.949179	0.969056	0.972627	0.969056
C3	0.867348	0.955016	0.972874	0.969179	0.969425
C4	0.867348	0.958588	0.972874	0.969179	0.969425
C5	0.888038	0.958588	0.969425	0.969179	0.969425
C6	0.923998	0.955156	0.983218	0.979647	0.979647
C7	0.906264	0.969072	0.976076	0.979647	0.976076
C8					0.972504
C9	0.9139	0.958727	0.979647	0.979647	0.969302
C10	0.927693	0.962299	0.976076	0.965854	0.965731
C11					

Vehicle1

Table 6.8: Vehicle1 without sampling

Classifier	0.809846069
C1	0.701320024
C2	0.609432008
C2	0.671699386
C3	0.705843968
C4	0.753546018
C5	0.655084963
C6	0.736612161
C7	0.714633733
C8	0.74053189
C9	0.757555589
C10	
C11	

Table 6.9: Vehicle1 over-sampling

Classifier	ADASYN-I	ROS-I	SMOTE-I	SMOTE_TL-I	SPIDER2-I
C1	0.812409	0.81304	0.807017	0.813036498	0.798957
C2	0.724089	0.732577	0.683535	0.720588402	0.7173
C2	0.614989	0.629241	0.589951	0.632309017	0.663756
C3	0.745741	0.703243	0.733039	0.768999651	0.721059
C4	0.750281	0.697044	0.719992	0.777455472	0.70864
C5	0.740482	0.678038	0.72385	0.77335777	0.689391

C6	0.776692	0.727257	0.742318	0.756978295	0.728721
C7	0.776692	0.727257	0.742318	0.774716037	0.735862
C8				0.717685318	
C9	0.776692	0.727257	0.742318	0.778147743	0.739842
C10	0.768646	0.746413	0.749924	0.763606987	0.739787
C11					

Table 6.10: Vehicle1 under-sampling

Classifier	CNNTL-I	CPM-I	NCL-I	SBC-I	TL-I
C1	0.799488	0.769965	0.814785	0.793484	0.816338
C2	0.690269	0.651984	0.754904	0.684821	0.754079
C2	0.587593	0.570157	0.648809	0.627449	0.634085
C3	0.745877	0.647688	0.760709	0.717169	0.770541
C4	0.746504	0.662545	0.753946	0.722641	0.781375
C5	0.726976	0.710674	0.783584	0.717428	0.760392
C6	0.751798	0.652673	0.761187	0.707724	0.745268
C7	0.732673	0.672227	0.763921	0.706366	0.757114
C8	0.749855	0.708195	0.731965	0.714532	0.738829
C9	0.735353	0.692745	0.766007	0.708955	0.773805
C10	0.730848	0.697268	0.77998	0.720981	0.769064
C11					

yeast6

Table 6.11: yeast6 without sampling

Classifier	0.875775
C1	0.808215
C2	0.542195
C2	0.738026
C3	0.752657
C4	0.71627
C5	0.669011
C6	0.801865
C7	0.859151
C8	0.86645
C9	0.811667
C10	
C11	

Table 6.12: yeast6 over-sampling

Classifier	ADASYN-I	ROS-I	SMOTE-I	SMOTE_TL-I	SPIDER2-I
C1	0.862796	0.87405	0.88661	0.871634	0.869841
C2	0.802002	0.803389	0.834151	0.81608	0.824913

C2	0.853278	0.870262	0.819061	0.806214	0.579728
C3	0.801866	0.694823	0.816493	0.812009	0.806696
C4	0.802211	0.723049	0.815458	0.812009	0.792756
C5	0.802211	0.708764	0.816148	0.798415	0.797038
C6	0.818013	0.801862	0.834024	0.817323	0.832506
C7	0.818013	0.801862	0.834024	0.830918	0.809247
C8	0.834492	0.872531	0.839682	0.836577	0.845893
C9	0.818013	0.801862	0.834024	0.830918	0.8715
C10	0.812839	0.800483	0.829194	0.856041	0.836436
C11					

Table 6.13: yeast6 under-sampling

Classifier	CNNTL-I	CPM-I	NCL-I	SBC-I	TL-I
C1	0.870945	0.87646	0.874049		0.87543
C2	0.627274	0.768676	0.846242		0.846927
C2	0.828441	0.774003	0.718246		0.643485
C3	0.804008	0.739202	0.792754		0.810834
C4	0.776077	0.750381	0.792754		0.810489
C5	0.746729	0.569852	0.792705		0.784281
C6	0.774617	0.641132	0.781572		0.769357
C7	0.760718	0.856179	0.814083		0.814079
C8	0.796238	0.841545	0.851761		0.825604
C9	0.812629	0.823475	0.867498		0.853903
C10	0.793868	0.830227	0.836442		0.824919
C11					

page-blocks0

Table 6.14: page-blocks0 without sampling

Classifier	0.925361
C1	0.94579
C2	0.762835
C2	0.930295
C3	
C4	0.881564
C5	0.930021
C6	0.948444
C7	
C8	0.960127
C9	0.952746
C10	
C11	

Table 6.15: page-blocks0 over-sampling

Classifier	ADASYN-I	ROS-I
C1		
C2	0.929913	0.933758
C2	0.784548	0.828767
C3	0.938714	
C4	0.938214	0.933725
C5	0.943775	0.9349
C6	0.943042	0.94178
C7	0.943042	0.94178
C8		
C9		
C10		
C11		

Wisconsin

Table 6.16: Wisconsin without sampling

Classifier	0.97074
C1	0.963632
C2	0.958196
C2	0.966561
C3	0.965614
C4	0.965273
C5	0.960191
C6	0.961119
C7	0.963366
C8	0.964105
C9	0.965286
C10	
C11	

Table 6.17: Wisconsin over-sampling

Classifier	ADASYN-I	ROS-I	SMOTE-I	SMOTE_TL-I	SPIDER2-I
C1	0.974434	0.97074	0.97074	0.971195	0.973487
C2	0.95814	0.94903	0.957868	0.964945	0.965773
C2	0.948694	0.941857	0.948151	0.938353	0.965569
C3	0.966056	0.958063	0.964313	0.961902	0.973607
C4	0.96814	0.958063	0.964313	0.96415	0.975734
C5	0.966056	0.96319	0.966397	0.972496	0.973487
C6	0.967869	0.962243	0.962199	0.972527	0.971568
C7	0.953224	0.962243	0.962199	0.969276	0.966069
C8	0.958355	0.966082	0.962553	0.964649	0.962553
C9	0.962389	0.962243	0.962199	0.973487	0.968361
C10	0.967869	0.971523	0.970576	0.963853	0.973487

C11					

Table 6.18: Wisconsin under-sampling

Classifier	CNNTL-I	CPM-I	NCL-I	SBC-I	TL-I
C1	0.970059	0.945696	0.971359	0.5	0.971195
C2	0.939815	0.903423	0.956948	0.380363	0.951954
C2	0.594887	0.912765	0.964446	0.380363	0.962362
C3	0.94628	0.91901	0.970444		0.967577
C4	0.947404	0.90651	0.970444		0.967577
C5	0.905288	0.874804	0.978777		0.9704
C6	0.928152	0.953582	0.96932		0.961315
C7	0.946319	0.940172	0.97028		0.967357
C8	0.957023	0.860014	0.95874		0.96641
C9	0.936311		0.962111		0.964933
C10	0.954196	0.93865	0.9704		0.962956
C11					

Vowel0

Table 6.19: Vowel0 without sampling

Classifier	0.970509
C1	0.942194
C2	0.694401
C2	0.970556
C3	0.970556
C4	0.970556
C5	0.971111
C6	0.968318
C7	0.981654
C8	0.950509
C9	0.9622
C10	
C11	

Table 6.20: Vowel0 over-sampling

Classifier	ADASYN-I	ROS-I	SMOTE-I	SMOTE_TL-I	SPIDER2-I
C1	0.961567	0.966068	0.966614	0.968287	0.953281
C2	0.968849	0.956654	0.971639	0.982197	0.941077
C2	0.806484	0.85879	0.894891	0.830422	0.725435
C3	0.988876	0.956654	0.976092	0.988873	0.970556
C4	0.988876	0.956654	0.974981	0.988873	0.970556

C5	0.988876	0.956654	0.976092	0.988873	0.970556
C6	0.981096	0.954988	0.980543	0.987768	0.953886
C7	0.981096	0.954988	0.980543	0.987768	0.967765
C8	0.957694	0.946605	0.953793	0.957145	0.947207
C9	0.981096	0.954988	0.980543	0.987768	0.947188
C10	0.977753	0.96554	0.983315	0.982197	0.962762
C11					

Table 6.21: Vowel0 under-sampling

Classifier	CNNTL-I	CPM-I	NCL-I	SBC-I	TL-I
C1	0.942173	0.901657	0.962728		0.970509
C2	0.920534	0.895472	0.942194	0.756086	0.942194
C2	0.814395	0.752505	0.663935	0.592259	0.678277
C3	0.946645	0.853876	0.969997		0.970556
C4	0.946645	0.853876	0.969997		0.970556
C5	0.941089	0.91212	0.958327		0.970556
C6	0.931642	0.878318	0.929988		0.971111
C7	0.92554	0.924407	0.964435		
C8	0.94388	0.903845	0.95888		0.952204
C9	0.924978	0.892204	0.947166		0.950509
C10	0.92387	0.888296	0.961086		0.9622
C11					

Glass6

Table 6.22: Glass6 without sampling

Classifier	0.911712
C1	0.88964
C2	0.915315
C2	0.872523
C3	0.872523
C4	0.88964
C5	0.86982
C6	0.897748
C7	0.933784
C8	0.917568
C9	0.928378
C10	
C11	

Table 6.23: Glass6 over-sampling

Classifier	ADASYN-I	ROS-I	SMOTE-I	SMOTE_TL-I	SPIDER2-I
C1	0.882432	0.925676	0.925676	0.925675676	0.906306

C2	0.89009	0.878378	0.892342	0.896486486	0.928378
C2	0.874324	0.893784	0.857748	0.888378378	0.925495
C3	0.931081	0.886486	0.855045	0.871711712	0.917117
C4	0.928378	0.886486	0.855045	0.892342342	0.917117
C5	0.928378	0.886486	0.855045	0.933783784	0.896486
C6	0.877568	0.897748	0.897748	0.931081081	0.91982
C7	0.877568	0.897748	0.897748	0.917117117	0.917117
C8	0.890541	0.928378	0.92027	0.917567568	0.914865
C9	0.877568	0.897748	0.897748	0.931081081	0.866757
C10	0.907207	0.933784	0.917568	0.928378378	0.917568
C11					

Table 6.24: Glass6 over-sampling

Classifier	CNNTL-I	CPM-I	NCL-I	SBC-I	TL-I
C1	0.922973	0.781532	0.911712		0.917117
C2	0.693243	0.691441	0.923423	0.5	0.92027
C2	0.803694	0.836126	0.893694	0.237838	0.83
C3	0.771622	0.773874	0.922973		0.925225
C4	0.771622	0.776577	0.885676		0.925225
C5	0.787838	0.795495	0.866757		0.89964
C6	0.760811	0.595045	0.871712		0.882523
C7	0.841441	0.542342	0.914865		0.914414
C8	0.792342		0.917568		0.925676
C9	0.904054	0.452703	0.909459		0.914865
C10	0.846847	0.641892	0.909459		0.92027
C11					

Glass5

Table 6.25: Glass5 without sampling

Classifier	0.973171
C1	0.942683
C2	0.853659
C2	0.947561
C3	0.947561
C4	0.973171
C5	0.795122
C6	0.887805
C7	0.840244
C8	0.94878
C9	0.987805
C10	
C11	

Table 6.26: Glass5 over-sampling

Classifier	ADASYN-I	ROS-I	SMOTE-I	SMOTE_TL-I	SPIDER2-I
C1	0.946341	0.968293		0.941463	0.934146
C2	0.893902	0.787805	0.953659	0.84878	0.79878
C2	0.907317	0.915854	0.879268	0.876829	0.885366
C3	0.915854	0.847561	0.880488	0.920732	0.842683
C4	0.920732	0.847561	0.880488	0.920732	0.842683
C5	0.868293	0.847561	0.880488	0.92561	0.928049
C6	0.920732	0.920732	0.965854	0.918293	0.990244
C7	0.920732	0.840244	0.965854	0.973171	0.878049
C8	0.881707	0.985366	0.931707	0.889024	0.815854
C9	0.920732	0.840244	0.965854	0.915854	0.871951
C10	0.915854	0.940244	0.968293	0.908537	0.878049
C11					

Table 6.27: Glass5 under-sampling

Classifier	CNNTL-I	CPM-I	NCL-I	SBC-I	TL-I
C1	0.929268	0.796341	0.953659		0.973171
C2	0.963415	0.903659	0.987805	0.5	0.940244
C2	0.902439	0.797561	0.853659	0.513415	0.846341
C3	0.937805	0.842683	0.937805		0.890244
C4	0.937805	0.842683	0.937805		0.890244
C5	0.995122	0.858537	0.92561		0.970732
C6	0.870732	0.908537	0.940244		0.837805
C7	0.968293	0.960976	0.890244		0.887805
C8	0.84878		0.978049		0.987805
C9	0.963415	0.797561	0.94878		0.94878
C10	0.960976	0.94878	0.940244		0.990244
C11					

New-thyroid2

Table 6.28: New-thyroid2 without sampling

Classifier	0.982937
C1	0.980159
C2	0.980159
C2	0.937302
C3	0.951587
C4	0.95754
C5	0.925794
C6	0.934524
C7	0.946032
C8	0.935317
C9	0.923413

C10	
C11	

Table 6.29: New-thyroid2 over-sampling

Classifier	ADASYN-I	ROS-I	SMOTE-I	SMOTE_TL-I	SPIDER2-I
C1	0.983333	0.982937	0.977778	0.961111	0.980159
C2	0.963889	0.931746	0.944048	0.935714	0.940079
C2	0.963889	0.960317	0.983333	0.972222	0.882143
C3	0.986111	0.934524	0.954762	0.963492	0.931746
C4	0.986111	0.934524	0.963492	0.963492	0.931746
C5	0.986111	0.934524	0.954762	0.960714	0.925794
C6	0.986111	0.928968	0.957937	0.980556	0.923016
C7	0.986111	0.928968	0.957937	0.955159	0.94881
C8	0.972222	0.946032	0.969444	0.961111	0.963095
C9	0.986111	0.928968	0.957937	0.975	0.943651
C10	0.972222	0.926587	0.96627	0.975	0.94881
C11					

Table 6.30: New-thyroid2 under-sampling

Classifier	CNNTL-I	CPM-I	NCL-I	SBC-I	TL-I
C1	0.954762	0.977381	0.997222		0.982937
C2	0.926984	0.835317	0.94881	0.5	0.963095
C2	0.929365	0.944048	0.932937	0.442063	0.899603
C3	0.969444	0.795238	0.946032		0.934524
C4	0.969444	0.795238	0.94881		0.923016
C5	0.946429	0.795238	0.951984		0.931746
C6	0.904762	0.838095	0.940079		0.943254
C7	0.926984	0.838095	0.92619		0.931746
C8	0.92381		0.943254		0.951984
C9	0.909524	0.809524	0.918254		0.92381
C10	0.93254	0.798016	0.940873		0.920635
C11					

A.2 NCL vs. NCL+ Comparison

Table 6.31: Abalone19 NCL&NCL+

Data Set	Abalone19	
Model	NCL	NCL+
C_SVMCS-I	0.760457	0.797602
C4.5CS-I	0.548425	0.799648
NNCS-I	0.507473	0.508451
AdaBoost-I	0.515339	0.799035
AdaBoostM2-I	0.51558	0.799155
AdaC2-I	0.554457	0.799648
Bagging-I	0.5	0.5
OverBagging2-I	0.529512	0.791638
MSMOTEBagging-I	0.579465	0.795147
UnderBagging2-I	0.71252	0.732963
UnderOverBagging-I	0.547419	0.791638

Table 6.32: yeast6 NCL & NCL+

Data Set	yeast6	
Model	NCL	NCL+
C_SVMCS-I	0.874049	0.880263
C4.5CS-I	0.846242	0.907171
NNCS-I	0.718246	0.620673
AdaBoost-I	0.792754	0.896202
AdaBoostM2-I	0.792754	0.896202
AdaC2-I	0.792705	0.886339
Bagging-I	0.781572	0.797929
OverBagging2-I	0.814083	0.888614
MSMOTEBagging-I	0.851761	0.888405
UnderBagging2-I	0.867498	0.888131
UnderOverBagging-I	0.836442	0.883782

Table 6.33: Glass6 NCL & NCL+

Data Set	Glass6	
Model	NCL	NCL+
C_SVMCS-I	0.911712	0.911712
C4.5CS-I	0.923423	0.88964
NNCS-I	0.893694	0.901802
AdaBoost-I	0.922973	0.885225
AdaBoostM2-I	0.885676	0.875225
AdaC2-I	0.866757	0.89964
Bagging-I	0.871712	0.82982

OverBagging2-I	0.914865	0.91982
MSMOTEBagging-I	0.917568	0.936486
UnderBagging2-I	0.909459	0.925676
UnderOverBagging-I	0.909459	0.925676

Table 6.34: Glass5 NCL & NCL+

Data Set	Glass5	
Model	NCL	NCL+
C_SVMCS-I	0.953659	0.973171
C4.5CS-I	0.987805	0.942683
NNCS-I	0.853659	0.880488
AdaBoost-I	0.937805	0.947561
AdaBoostM2-I	0.937805	0.947561
AdaC2-I	0.92561	0.973171
Bagging-I	0.940244	0.840244
OverBagging2-I	0.890244	0.887805
MSMOTEBagging-I	0.978049	0.887805
UnderBagging2-I	0.94878	0.94878
UnderOverBagging-I	0.940244	0.990244

Table 6.35: New-thyroid2 NCL & NCL+

Data Set	New-thyroid2	
Model	NCL	NCL+
C_SVMCS-I	0.997222	0.997222
C4.5CS-I	0.94881	0.988889
NNCS-I	0.932937	0.851587
AdaBoost-I	0.946032	0.997222
AdaBoostM2-I	0.94881	0.997222
AdaC2-I	0.951984	0.986111
Bagging-I	0.940079	0.94881
OverBagging2-I	0.92619	0.963095
MSMOTEBagging-I	0.943254	0.983333
UnderBagging2-I	0.918254	0.95754
UnderOverBagging-I	0.940873	0.969048

Table 6.36:Vowel0 NCL & NCL+

Data Set	Vowel0	
Model	NCL	NCL+
C_SVMCS-I	0.962728	0.971611
C4.5CS-I	0.942194	0.983315
NNCS-I	0.663935	0.708976

AdaBoost-I	0.969997	0.988333
AdaBoostM2-I	0.969997	0.988333
AdaC2-I	0.958327	0.987222
Bagging-I	0.929988	0.987778
OverBagging2-I	0.964435	0.996102
MSMOTEBagging-I	0.95888	0.993318
UnderBagging2-I	0.947166	0.976611
UnderOverBagging-I	0.961086	0.989981

Table 6.37: vehicle1 NCL & NCL+

Data Set	vehicle1	
Model	NCL	NCL+
C_SVMCS-I	0.814785	0.81837
C4.5CS-I	0.754904	0.863875
NNCS-I	0.648809	0.602263
AdaBoost-I	0.760709	0.909126
AdaBoostM2-I	0.753946	0.911452
AdaC2-I	0.783584	0.887962
Bagging-I	0.761187	0.883528
OverBagging2-I	0.763921	0.874451
MSMOTEBagging-I	0.731965	0.804011
UnderBagging2-I	0.766007	0.854458
UnderOverBagging-I	0.77998	0.844671

Table 6.38: Wisconsin NCL & NCL+

Data Set	Wisconsin	
Model	NCL	NCL+
C_SVMCS-I	0.971359	0.979118
C4.5CS-I	0.956948	0.97124
NNCS-I	0.964446	0.969825
AdaBoost-I	0.970444	0.981365
AdaBoostM2-I	0.970444	0.979118
AdaC2-I	0.978777	0.979118
Bagging-I	0.96932	0.973815
OverBagging2-I	0.97028	0.979901
MSMOTEBagging-I	0.95874	0.970621
UnderBagging2-I	0.962111	0.972527
UnderOverBagging-I	0.9704	0.972691

Table 6.39: ecoli-0_vs_1 NCL & NCL+

Data Set	ecoli-0_vs_1	
Model	NCL	NCL+
C_SVMCS-I	0.976199	0.979647
C4.5CS-I	0.958588	0.986667
NNCS-I	0.969056	0.979524
AdaBoost-I	0.972874	0.97977
AdaBoostM2-I	0.972874	0.983218
AdaC2-I	0.969425	0.97977
Bagging-I	0.983218	0.986667
OverBagging2-I	0.976076	0.986667
MSMOTEBagging-I		0.986667
UnderBagging2-I	0.979647	0.983095
UnderOverBagging-I	0.976076	0.983218

Table 6.40: NCL & NCL+ Average performance is shows the average performance comparison between NCL and NCL+. The table is sorted based on IR value.

Table 6.40: NCL & NCL+ Average performance

IR	AUC_{st}NCL	AUC_{st}NCL+
128.87	0.570059	0.73772
32.78	0.815282	0.85761
22.81	0.902482	0.900066
15.47	0.935809	0.929047
10.1	0.929885	0.961053
5.14	0.944949	0.96728
3.23	0.756345	0.841288
1.86	0.96757	0.975394
1.86	0.973403	0.983174
Average	0.866198	0.905848

Bibliography

Alcalá-Fdez, J., Fernandez, A., Luengo, J., Derrac, J., Garcia, S., Sanchez, L., & Herrera, F. (2011). KEEL Data-Mining Software Tool: Data Set Repository. *Journal of Multiple-Valued Logic and Soft Computing, vol: 17*, 255-287.

Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. *SIGKDD Explorations, vol: 6*, 20-29.

Cano, J., Herrera, F., & Lozano, M. (2003). Using Evolutionary Algorithms as Instance Selection for Data Reduction in KDD: An Experimental Study. *IEEE Transactions on Evolutionary Computing, vol: 7*, 561-575.

Cao, P., Zhao, D., & Zaiane, O. (2013). An Optimized Cost-Sensitive SVM for Imbalanced Data Learning. *Lecture Notes in Computer Science, vol: 7819*, 280-292.

Center of Machine Learning and Intelligent. (2014). *Machine Learning Repository*. Retrieved from UCI: http://archive.ics.uci.edu

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research, vol: 16*, 321-357.

Eshelman, L. J. (1990). The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, (pp. 265-283). Bloomington, Indiana, USA.

Fernándeza, A., Garcíaa, S., Jesusb, M., & Herreraa, F. (2008). A Study of The Behaviour of Linguistic Fuzzy Rule Based Classification Systems in the Framework of Imbalanced Data-Sets. *Fuzzy Sets and Systems, vol: 159*, 2378-2398.

Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., & Herrera, F. (2011). A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches. *IEEE Transactions on Systems, Man, and Cybernetics Part C, vol: 42*, 463-484.

He, H., Bai, Y., Garcia, E., & Li, S. (2008). ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning. *Proceedings of the 2008 International Joint Conference on Neural Networks*, (pp. 1322-1328). Hong Kong, China.

Hu, S., Liang, Y., He, Y., & Ma, L. (2009). MSMOTE: Improving Classification Performance When Training Data is Imbalanced. *Proceedings of the 2nd International Workshop on Computer Science and Engineering*, (pp. 13-17). Qingdao, China.

Jo, T., & Japkowicz, N. (2004). A Multiple Resampling Method for Learning from Imbalanced Data Sets. *Computational Intelligence, vol: 20*, 18-36.

Johnson, B. A., Ryutaro, T., & Hoan, N. T. (2013). A Hybrid Pansharpening Approach and Multiscale Object-Based Image Analysis for Mapping Diseased Pine and Oak Trees. *International Journal of Remote Sensing, vol: 34*, 6969-6982.

Kukar, M., & Kononenko, I. (1998). Cost-Sensitive Learning with Neural Networks. *Proceedings of the 13th European Conference on Artificial Intelligence*, (pp. 445-449). Brighton, UK.

Laurikkala, J. (2001). Improving Identification of Difficult Small Classes by Balancing Class Distribution. *Proceedings of the 8th Conference on AI in Medicine in Europe*, (pp. 63-66). Berlin, Germany.

Leo, B. (1996). Bagging Predictors. *Machine Learning, vol: 24*, 123-140.

Ling, C., & Li, C. (1998). Data Mining for Direct Marketing Problems and Solutions. *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, (pp. 217-225). New York City, New York, USA.

Lokanayaki, K., & Malathi, A. (2013). Data Preprocessing for Liver Dataset Using SMOTE. *International Journal of Advanced Research in Computer Science and Software Engineering, vol: 3*, 559-562.

Napieralla, K., Stefanowski, J., & Wilk, S. (2010). Learning from Imbalanced Data in Presence of Noisy and Borderline Examples. *Proceedings of the 7th International Conference on Rough Sets and Current Trends in Computing*, (pp. 158-167). Warsaw, Poland.

Provost, F., & Fawcett, T. (2001). Robust Classification for Imprecise Environments. *Machine Learning, vol: 42*, 203-231.

Qin, Z., Zhang, C., Wang, T., & Zhang, S. (2010). Cost Sensitive Classification in Data Mining. *Lecture Notes in Computer Science, vol: 6440*, 1-11.

Ting, K. M. (2002). An Instance-Weighting Method to Induce Cost-Sensitive Trees. *IEEE Transactions on Knowledge and Data Engineering, vol: 14*, 659-665.

Veropoulos, K., Campbell, C., & Cristianini, N. (2007). Controlling the Sensitivity of Support Vector Machines. *Proceedings of the 2007 International Joint Conferences on Artificial Intelligence*, (pp. 281-288). Stockholm, Sweden.

Wang, S., & Yao, X. (2009). Diversity Analysis on Imbalanced Data Sets by Using Ensemble Models. *Proceedings of the 2009 IEEE Symposium Series on Computational Intelligence and Data Mining*, (pp. 324-331). Nashville, Tennessee, USA.

Yen, S.-J., & Lee, Y.-S. (2006). Under-Sampling Approaches for Improving Prediction of the Minority Class in an Imbalanced Dataset. *Proceedings of the 2006 International Conference on Intelligent Computing*, (pp. 731-740). Kunming, China.

Yoon, K., & Kwek, S. (2005). An Unsupervised Learning Approach to Resolving the Data Imbalanced Issue in Supervised Learning Problems in Functional Genomics. *Proceedings of the 5th International Conference on Hybrid Intelligent Systems*, (pp. 303-308). Rio de Janeiro, Brazil.

Zheng, E.-h., Zou, C., Sun, J., & Chen, L. (2011). Cost-sensitive SVM with Error Cost and Class-Dependent Reject Cost. *International Journal of Computer Theory and Engineering, vol: 3*, 130-135.

Zhou, Z.-H., & Liur, X.-Y. (2006). Training Cost-Sensitive Neural Networks with Methods Addressing the Class Imbalance Problem. *IEEE Transactions on Knowledge and Data Engineering, vol: 18*, 63-77.