CLASSIFICATION TECHNIQUES FOR NOISY AND IMBALANCED DATA

by

Amri Napolitano

A Dissertation Submitted to the Faculty of

The College of Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Florida Atlantic University

Boca Raton, Florida

December 2009

CLASSIFICATION TECHNIQUES FOR NOISY AND IMBALANCED DATA

by

Amri Napolitano

This dissertation was prepared under the direction of the candidate's dissertation advisor, Dr. Taghi M. Khoshgoftaar, Department of Computer and Electrical Engineering and Computer Science, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fullfillment of the requirements for the degree of Doctor of Philosophy.

SUPERVISORY COMMITTEE:

[illegible]

Taghi M. Khoshgoftaar, Ph.D.
Dissertation Advisor

[illegible]

Martin K. Solomon, Ph.D.

[illegible]

Xinquan Zhu, Ph.D.

[illegible]

Hanqi Zhuang, Ph.D.

[illegible]

Borko Furht, Ph.D.
Chair, Department of Computer and Electrical Engineering and Computer Science

[illegible]

Karl K. Stevens, Ph.D., P.E.
Dean, College of Engineering and Computer Science

[illegible]

Barry T. Rosson.
Dean, Graduate College

Nov. 10, 2009
Date

ACKNOWLEDGMENTS

I would like to thank Dr. Taghi M. Khoshgoftaar for his invaluable guidance and support of my research and graduate studies at Florida Atlantic University. I also thank Dr. Martin K. Solomon, Dr. Xingquan Zhu, and Dr. Hanqi Zhuang for serving on my dissertation committee. I would also like to thank my current and former colleagues in the Data Mining and Machine Learning Laboratory, especially Dr. Jason Van Hulse for all of his contributions to and feedback on my research. Finally, I'm most grateful to my parents and family for their support and encouragement during my graduate studies and all other endeavors.

ABSTRACT

Author:	Amri Napolitano
Title:	Classification Techniques for Noisy and Imbalanced Data
Institution:	Florida Atlantic University
Dissertation Advisor:	Dr. Taghi M. Khoshgoftaar
Degree:	Doctor of Philosophy
Year:	2009

Machine learning techniques allow useful insight to be distilled from the increasingly massive repositories of data being stored. As these data mining techniques can only learn patterns actually present in the data, it is important that the desired knowledge be faithfully and discernibly contained therein. Two common data quality issues that often affect important real life classification applications are class noise and class imbalance. Class noise, where dependent attribute values are recorded erroneously, misleads a classifier and reduces predictive performance. Class imbalance occurs when one class represents only a small portion of the examples in a dataset, and, in such cases, classifiers often display poor accuracy on the minority class. The reduction in classification performance becomes even worse when the two issues occur simultaneously.

To address the magnified difficulty caused by this interaction, this dissertation performs thorough empirical investigations of several techniques for dealing with class noise and imbalanced data. Comprehensive experiments are performed to assess the effects of the classification techniques on classifier performance, as well as how the level of class imbalance, level of class noise, and distribution of class noise among the classes affects results. An empirical analysis of classifier based noise detection efficiency appears first. Subsequently, an intelligent data sampling technique, based on noise detection, is proposed and tested. Several hybrid classifier ensemble techniques for addressing class noise and imbalance are introduced. Finally, a detailed empirical investigation of classification filtering is performed to determine best practices.

Classification Techniques for Noisy and Imbalanced Data

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

With constant decreases in the cost of storage and computing power, giant stores of data have become pervasive in many industries. The amount of data stored precludes manual human analysis, necessitating the use of automated methods to extract useful information from these massive warehouses of data. Techniques from data mining and machine learning are able to process structured data to extract meaningful patterns. Data mining algorithms learn by induction, so the data to be mined must be structured as a collection of examples of the target concept to be learned. Each example, or instance, is described by a set of attribute values, which typically are either numeric or categorical values.

From a set of structured examples of a target concept, a data mining algorithm can extract useful patterns, typically in one of four ways. *Clustering* and *association rule mining* are unsupervised learning processes, because they don't involve the prediction of values for a specific attribute. Clustering involves partitioning the set of instances into groups such that examples in the same group are similar to each other based on some measure of similarity. Association rule mining involves looking for useful predictive patterns between any combinations of attributes in the data. This differs from supervised learning in that potentially interesting associations between any attributes or sets of attributes are sought. In supervised learning, one of the attributes, called the class attribute or dependent attribute, is meant to be predicted based on the values for the other attributes, called independent attributes. The process is called *classification* if the class attribute is categorical and *regression* or *numeric prediction* if the class attribute is numeric. This work specifically addresses the problem of classification.

In classification, the objective is to successfully predict the values for the nominal class attribute (class label) of an example given the values for its independent attributes. Classically, success in this endeavor is measure by overall accuracy, the percentage of instances for which the class label is correctly predicted. Classification algorithms, of which there are many, were often designed in the spirit of achieving the maximum possible number of correct class label predictions. In order to extrapolate reliable patterns from a dataset so that future instances can be classified accurately, the information contained in the data must be valid. "Garbage in, garbage out" is one of the mantras of computer science after all. Unfortunately, in real world classification applications, data is rarely perfect. A number of issues can affect the quality of the data used to train a classifier, often leading to reduced classification accuracy.

1.1.1 Data Noise

Perhaps the most common data quality problem is that of data noise, or incorrect values for one or more of the attributes that describe an example. There are two types of data noise: attribute noise and class noise. Attribute noise occurs when there are erroneous values in the independent attributes of a dataset, while class noise refers to incorrect values in the dependent attribute. Previous research has examined these two types of noise and found that class noise has a more detrimental effect on classification performance than attribute noise [91]. To confidently identify noisy data would usually require a domain expert, however, due to either the lack of availability of such expertise or the intractibility of manually investigating the data, data mining techniques are often used to handle the noise in the data in addition to learning from it.

The generally proposed solutions for handling class noise fall into three categories. One way is to use a learner that is robust to noise in the data [28]. The C4.5 decision tree learner [61], for example, uses a tree pruning step intended to remove statistically insignificant portions of the induced decision trees. Similarly, several boosting methods have been developed for being robust to class noise, either by ignoring potentially noisy instances [45] or by preventing extreme increases in instance weights [58]. Another method for dealing with class noise is noise filtering, where potentially noisy instances are identified and removed from the training dataset. Another very similar method is noise correction, in which noisy instances are again identified, but have their class labels changed instead of removing the instances entirely. Various approaches have been explored for deciding which instances to consider noisy. Some procedures involve analysis of the data points to look for outliers given the class labels (without explicitly using a standard classification algorithm) [77, 76]. The more frequent approach, however, tends to be classification filtering [49, 86], in which a classifier is built and evaluated on the training data (usually using cross-validation) and the instances that are misclassified are considered to be noisy.

Despite it being widely implemented and studied, the classification filtering approach to noise handling has not received thorough, unified investigation. There are several options involved in the process which seem to only be treated individually, if at all, in past research. One must choose what classifier(s) to use for performing the classification filtering. There is also the option of which class(es) to identify noisy instances from, as well as how to decide which instances are noisy based on the classification output. Typically, any instances which are incorrectly predicted using the default decision threshold are considered noisy, however, the amount of noise to detect can be adjusted by altering the decision threshold used. Once the suspected noisy instances are identified, the practitioner then must choose how to deal with these examples, usually either removing them from the training data or changing their class label to the presumably correct one.

Four different empirical investigations are presented in this dissertation, all of which involve the use of classification filtering in some form or another. The work presented in Chapter 7 thoroughly investigates the effects of the different classification filtering options mentioned, motivated by the lack of research which gives a complete perspective on this noise handling technique. Chapter 4 focuses specifically on the aspect of determining which instances to consider noisy. Eschewing performing classifier evaluation, the study presented in Chapter 4 focuses specifically on the noise identification procedures to gauge the actual efficiency of different classifiers for performing the classification filtering by examining the actual noise identification accuracy while varying the decision threshold used for filtering.

1.1.2 Class Imbalance

Another common data quality issue that tends to impair classification performance is that of imbalanced classes. For many important real world classification applications, such as software quality prediction [35], fraud detection [23], diagnosis of rare diseases [88], and text classification [90], examples of the class of interest may represent only 1% or less of the total number of instances [67]. A classifier that learns to always predict the majority class would achive 99% accuracy, but such a classifier would not be useful for identifying the class of interest. Many classifiers were designed with the goal of maximizing overall classification accuracy, which isn't affected much by a rare class, and thus that classifier might not give much weight to minority class instances. In other cases, the classifier may perform poorly on the minority class in an attempt to be robust to noisy data, which rare class examples can resemble due to their scarcity. In cases such as these, many classification algorithms tend to have poor accuracy in identifying the minority (positive) class, a situation that is all the more unfortunate because, in such cases, it is usually the minority class which is most important to identify and carries the highest cost of misclassification.

Typical methods for improving performance when classifying imbalanced data include data sampling, cost sensitive learning, and ensemble classification schemes. Numerous data sampling techniques have been investigated, all of which either undersample the majority class [50] or oversample the majority class [14]. Random undersampling can suffer from the drawback of losing important information by indiscriminantly discarding majority class instances. Conversely, random oversampling can lead to overfitting by duplicating examples without adding any new information. To avoid these shortcomings, more sophisticated undersampling [6] and oversampling [30] techniques have been proposed. Another drawback of data sampling is that it's not totally clear what class distribution to impart on a training dataset by sampling. Some work [85] suggests that an even class distribution produces the best results, depending on the performance metric, while another study showed that an approximate 2:1 class distribution (35% minority class) works well for highly imbalanced data [48].

In cost sensitive learning, the instances in a dataset are assigned costs of misclassification (and sometimes also costs of correct classification), and the learner then attempts to minimize the total cost of misclassification of the resulting model. Cost sensitive learning techniques can be used to address class imbalance by assigning the minority class a much higher cost of misclassification than the other class(es), thus biasing the learner to give more weight to those instances despite their scarcity in the data. Ensemble methods, such as variants of Bagging [9] that adjust the class distribution and modified boosting algorithms that adjust the weight update formulas [43] or sample the training datasets at each iteration [72] can successfully improve performance. Additionally, several cost sensitive boosting techniques have been proposed, combining the cost sensitive and ensemble approaches [22].

Imbalanced data must be accounted for when addressing almost any aspect of classification. Activities such as attribute selection can be impacted by imbalanced class distributions [4]. The same is true with addressing class noise. The poor classification performance on imbalanced data is even further exacerbated when the class imbalance is combined with class noise in the data. If instances that should actually belong to the minority class are mislabeled as majority class instances, it reduces the support for the class that is already underrepresented. If, on the other hand, actual majority class instances are mislabeled as belonging to the minority class, it can crowd the actual minority instances and make it even harder to discern where the actual pockets of minority class instances are. Further, some research [40] proposes that class imbalance by itself is not the primary cause of poor classifier performance, but rather that issues such as small disjuncts, overlapping classes, concept complexity, or absolute rarity may be more important. All of these issues can easily be caused by class noise, as mislabeled instances can create small disjuncts and overlapping classes and increase concept complexity.

Despite the increased threat to performance that is imposed by the simultaneous presence of class imbalance and class noise, relatively little research addresses both issues at the same time. Some sampling techniques for class imbalance are essentially just classification filters that only work on the majority class, which does address both issues. In such cases, though, the amount of instances found as noisy is rarely going to be enough to improve performance on significantly imbalanced data. In an effort to circumvent that obstacle, Chapter 5 presents the Filtered Undersampling (FUS) technique which attempts to remove noise and balance the classes enough to address the reduced performance stemming from the class imbalance. Chapter 6 also combines solutions for class imbalance and class noise, proposing six new hybrid boosting algorithms intended to address both data quality issues.

1.2 Contributions

This research examines various machine learning techniques for improving classification performance for noisy, imbalanced data. The major contributions are listed here.

1. We provide thorough, statistically valid empirical analysis of a number of data mining techniques in the context of noisy, imbalanced data. Whereas most previous studies have ignored the subject entirely or provided shallow treatments of the area, this work provides comprehensive investigations of classification of

noisy and imbalanced data, classification filtering, and boosting techniques for handling both noisy and imbalanced data.

2. We provide, to our knowledge, the first examination of the actual noise identification efficiency of classification filtering in Chapter 4. Rather than focus on its effects as a data preprocessing technique, we rigidly control the artificial noise injected into four imbalanced datasets and record how eight different learners perform as classification filters. Instead of choosing a single threshold to identify noisy instances, we examine the performance across all the thresholds and create ROC curves depicting how efficiently each of the eight learners can identify the actually noisy instances.

3. The Filtered Undersampling technique is proposed for noisy, imbalanced data in Chapter 5. This new data sampling technique is intended to remove class noise while also having enough of an effect on the class distribution that it improves the classifier performance on imbalanced data.

4. Six new hybrid boosting techniques are proposed in Chapter 6. The six new algorithms combine the strengths of boosting techniques aimed at class imbalance with those of boosting techniques intended for noisy data to produce ensemble classification techniques that can thrive in the presence of both data quality issues.

5. Chapter 7 provides the first comprehensive examination of the intricacies of classification filtering. Classification filter parameters defining the learner used for filtering, which class(es) to detect noise from, whether to remove those noisy instances or change their class label, and what strategy to use for determining how many instances to identify as noisy are all examined, though previous studies on the topic have examined only one at a time, if that. Additionally, the effects of varying levels of class distribution, noise level, and noise distribution, as well as the learner used to perform classification after this preprocessing, are measured and reported.

1.3 Dissertation Structure

Related work regarding the topics of data noise and class imbalance is summarized in Chapter 2. Chapter 3 provides details of the empirical experiments performed throughout the work. This chapter describes the datasets used for empirical testing, performance metrics used throughout the work, details of the classification filtering procedure used throughout the experiments, and the statistical techniques utilized to verify the significance of the results. An empirical examination of classification filtering efficiency is given in Chapter 4. A new intelligent undersampling technique, FUS, is proposed in Chapter 5. Chapter 6 presents six new hybrid boosting techniques for addressing both class noise and class imbalance. Chapter 7 provides a thorough empirical investigation of classification filtering parameters, specifically in the context of noisy, imbalanced data. Finally, conclusions and suggestions for future work are supplied in Chapter 8.

CHAPTER 2

RELATED WORK

2.1 Data Noise

Real world classification initiatives are frequently plagued by the problem of data noise, which can occur in the independent values (attribute noise) or the dependent value (class noise). For identifying attribute noise, Zhao and Nishida [89] employed a fuzzy logic approach using the support coefficient function (SCF) that they proposed. Alternatively, Van Hulse et al. [81] developed the Pairwise Attribute Noise Detection Algorithm (PANDA), which can detect outlier attribute values, even in an unsupervised context. The work of Zhu and Wu [91], however, showed that class noise has a more detrimental impact on classifier performance than attribute noise. Class noise is usually handled in one of three ways: either using a classifer that is robust to noisy data, identifying potentially noisy instances and removing them from the training data, or finding potentially noisy instances and changing their class labels to the presumably correct values.

C4.5 [61] is an example of a learner with mechanisms in place to be robust to noisy data. Its secondary tree pruning step is intended to remove statistically insignificant portions of the induced decision tree to avoid overfitting outliers in the training data. Folleco et al. [24] performed a case study to investigate which learners are robust to low quality data. AdaBoost [26], a popular metalearning technique, has been found to be susceptible to overfitting noisy instances by Jiang [41], who examined some of the theoretical issues involved when boosting is applied to noisy data. To remedy this, several boosting variants have been proposed to make the process more robust to the presence of class noise. Oza proposed the AveBoost2 technique [58] to improve boosting's generalization performance, and thus accuracy in the presence of class noise, moderating the growth of instance weights to improve generalization error (at the expense of higher training error). Karmaker and Kwek [45] proposed a boosting technique which takes a different approach to improving performance for noisy data. Outlier Removal Boosting (ORBoost) works by disregarding any instances whose weight grows to exceed a certain threshold, assuming that such instances are likely to be noisy and harm the learning process.

Noise filtering, where instances suspected to be noisy are removed from the training dataset before building the classification model, has been examined in numerous studies. Some other work instead chose the alternative of noise correction, where the instances identified as noisy have their class label changed instead of being removed from the data. The biggest challenge is identifying the noisy instances. Typically, instances are determined to be potentially noisy using classification algorithms themselves. Examples which a classifier has trouble correctly predicting are considered likely to be outliers, and can then be dealt with by either removing them or correcting their class label. Even automated noise detection methods which don't explicitly appeal to a classification algorithm are essentially performing the same task. For example, a "Tomek link" [76, 50] is shared by two instances of different classes when they are the nearest neighbors of each other. Instances participating in Tomek links can be considered potentially noisy and handeled accordingly. While it's not described in the context of building a nearest neighbor classifier and evaluating it on the data and flagging any misclassified instances as noisy, that is essentially what is happening. Van Hulse and Khoshgoftaar, alternatively, used the concept of frequent itemsets to identify noisy instances in software engineering data[77] with categorical independent attributes. Though not proposed as a general classification algorithm, their noise detection algorithm includes a function called ItemsetClassification(), which is essentially a rule-based classifier.

The process of using a learning algorithm to identify noisy instances in the data is typically called *classification filtering*. Usually, the learner is built and evaluated on the training data using cross-validation. In cross-validation, the data is split into k equal subsets called folds. For each fold, the instances are evaluated using a classifier built on the other $k-1$ folds. This process is repeated until the instances in all k folds have been evaluated. In classification filtering, the instances that are misclassified by this process are identified as noisy, though it may just be the case that the instance is difficult to classify correctly, either in general or by the specific classifier used for classification filtering.

Wilson, attempting to prune a dataset for use with a 1NN classifier, used a 3NN learner to perform classification filtering on the dataset [86]. Khoshgoftaar et al. [47] used their Rule-Based Modeling classifier, based on the Kolmogorov-Smirnov statistic, to remove noise from software quality classification data. Zhu et al. [93] use a partioning approach to perform classification filtering on extremely large datasets with the C4.5rules learner [61]. To avoid being pigeonholed by the specific inductive bias of the single classifier selected to perform classification filtering, ensembles of classifiers have been put to the task of identifying noise. Brodley and Friedl [11] used a consensus ensemble filter, whereby all the classifiers in the ensemble had to misclassify an instance for it to be considered noisy, for classifying satellite imagery with artificially injected class noise. Khoshgoftaar et al. [49] used various levels of majority voting with an ensemble of 25 classifiers to identify noise in imbalanced software quality datasets. By varying the size of the majority necessary to identify an instance as noisy, this is the only previous work, that we are aware of, that considers the issue of the decision threshold to use for classifying an instance. Most classifiers, even single classifiers where there aren't multiple votes, can produce some measure of their confidence in their predictions for each instance. By adjusting the threshold on this confidence rating, one can control how liberal or conservative to be with the classifications (and thus detecting noise in this case).

In most cases, instances identified as noisy in the classification filtering process are subsequently removed from the training dataset. Some work, however, has also examined the effects of correcting the noisy instance rather than discarding it. Teng proposed a noise correction technique for categorical data called polishing [74], in which both class and attribute noise can be identified. Another noise correction technique is proposed by Van Hulse et al. [80], utilizing Bayesian Multiple Imputation [64] for noise correction, though it targets problems with numeric class variables rather than categorical.

2.2 Class Imbalance

Class imbalance is another problem with data quality that often sabotages classification performance. Class imbalance, where one class represents a small proportion of the instances in a dataset, afflicts many real world domains, such as fraud detection [23], diagnosis of rare diseases [88], text classification [90], and software quality prediction [46]. Many classifiers, in the spirit of either maximizing overall classification accuracy or being robust to noisy data, tend to sacrifice performance on the minority class in favor of classifying more of the majority instances correctly. If both classes were of equal importance, this wouldn't be a problem, however, in most cases of class imbalance, it is the rare class that carries the highest cost of misclassification. For class imbalance also, the solutions tend to fall into three categories. The most common is data sampling, whereby instances of the majority class are undersampled, instances of the minority class are oversampled, or both. Another approach to improving classification performance is cost sensitive learning, whereby the minority class is assigned a higher cost of misclassification. This biases the cost sensitive learner towards putting more weight on the minority class as it attempts to minimize overall cost of misclassification rather than overall classification error. Having become more and more popular, ensemble methods are the final class of solutions to the problem of classifying imbalanced data, often involving modified versions of boosting [26] or bagging [9].

As it often occurs in real world data, the problem of class imbalance has received increased attention in recent years, with many researchers analyzing specifically how class imbalance results in decreased performance, as well as myriad techniques for improving classification results when it is present. A broad survey of the topic is presented by Weiss [84], detailing the problem and many of the approaches proposed for addressing it. Weiss and Provost [85] study the effects of class distribution on classification performance, finding that, in general, a balanced class distribution will result in better results when evaluated using ROC curves while the natural distribution tends to produce the best overall accuracy. Other work by Khoshgoftaar et al. [48] suggests that the optimal class distribution (when using ROC curves) is often dependent on the natural class distribution and that, in general, when datasets are severely imbalanced, a class distribution of 2:1 usually performs better than the more balanced (1:1) class distribution.

Perhaps the most popular way to deal with imbalanced data is to sample the data before training a classifier with it. All such techniques aim to add instances to the minority class, remove instances from the majority class, or do both, in order to balance out the class distribution of the training data and improve classification performance. The most widely studied sampling techniques are simple random oversampling of the minority class and random undersampling of the majority class. These two techniques were investigated by Drummond and Holte [20], who found that undersampling tends to outperform oversampling when training models with the popular C4.5 decision tree learner, a result corroborated by Napolitano et al. [57]. Alternatively, Maloof [53] used Naive Bayes and C5.0 (the commercial successor to C4.5) to show that over and undersampling techniques result in roughly equivalent classifiers. Research by Japkowicz [39] using neural networks and artificially generated one-dimensional data found that oversampling and undersampling perform similarly on smaller datasets, while undersampling performs better on larger datasets. A later study by Japkowicz and Stephan [40] found that for C5.0, oversampling performs better than undersampling.

More sophisticated methods for balancing class distributions have also been proposed. Chawla et al. [14] proposed the Synthetic Minority Oversampling Technique (SMOTE), which produces new minority class examples by extrapolating between existing ones. Expanding on this idea, Han et al. [30] introduced Borderline-SMOTE, which applies SMOTE only to those minority instances deemed to be on the decision boundary. Jo and Japkowicz [42] proposed the Cluster-Based Oversampling technique, which clusters the instances of each class separately and randomly oversamples the instances within individual clusters to achieve parity both between and within classes (intended to address the problem of small disjuncts). Similarly, several intelligent undersampling methods have been tried. Wilson's Editing [86] was proposed to prune a training dataset by removing any instances misclassified by a 3-NN classifier. Barandela et al. [6] modify this older strategy for use with imbalanced data by only removing the majority class instances. Basically a straightforward classification filter, the technique removes majority instances that might potentially be noisy and make the minority class even more difficult to discern. Kubat and Matwin [50] proposed another intelligent undersampling technique called One-Sided Selection. This preprocessing method aims to remove majority class instances that are either redundant or noisy. In this case, noisy instances are identified by the concept of Tomek links [76].

Another class of techniques used to address problems with classifying imbalanced data is cost-sensitive learning. Cost-sensitive learning techniques take, as input, costs for certain types of misclassifications (and sometimes correct classifications as well), and perform the learning task with the aim of minimizing the overall cost of classification rather than overall error rate. Higher costs can be assigned to misclassifications of minority class instances, leading the classifier to put more focus on that class. Elkan [21] describes methods for constructing cost-sensitive learners by either appropriate sampling of the training data or thresholding of the probability estimates produced by a classifier. Weiss et al. compared undersampling, oversampling, and cost-sensitive learning [83] on imbalanced datasets, finding no clear optimal choice. Viaene and Dedene [82] discuss the method of changing the decision threshold and the use of both undersampling and oversampling for cost sensitive classification. Domingos proposed the MetaCost technique [18], which uses bagging to achieves cost sensitive learning with error-based learners. Chawla et al. [13] perform a detailed evaluation of a wrapper-based sampling approach to minimizing misclassification cost. A wrapper-based optimization process is used to determine the sampling percentages to use for SMOTE [14] and random undersampling (both applied in combination). Seiffert et al. [66] performed an empirical comparison of four sampling techniques and two types of cost sensitive learning using 15 datasets and two learners, finding that random undersampling performed as well as or better than any of the other techniques for addressing class imbalance.

The standard AdaBoost has been utilized for imbalanced data [68, 71], and there have also been several ensemble techniques proposed specifically for addressing class imbalance. Chawla et al. developed SMOTEBoost, which adds oversampling with SMOTE to the AdaBoost [26] algorithm to improve minority class performance [15]. Similarly, RUSBoost [72] follows the same strategy, replacing SMOTE with random undersampling. Joshi et al. proposed RareBoost [43], which adjusts the boosting process so that false positives, false negatives, true positives, and true negatives are all treated separately and have their weights adjusted accordingly so that the minority class is given more importance. In addition to boosting techniques, bagging techniques have also been adapted to incorporate sampling. Two that have been successfully applied to improve performance when classifying imbalanced data include Exactly Balanced Bagging [55, 44, 52] and Roughly Balanced Bagging [32]. In Exactly Balanced Bagging, the bootstrap samples for each iteration are undersampled so that the class distribution is equal. Roughly Balanced Bagging, trying to stay more faithful to the theory of the original bagging technique [9], in which the class distribution of each bootstrap sample will vary, probabilistically samples instances for each bootstrap sample such that the class distribution will be even, on average, but can vary. A study by Van Hulse et al. [36] compares the performance of several of these ensemble classification techniques based on repetitive undersampling. A number of proposed methods combine both cost-sensitive and ensemble learning to produce cost-sensitive boosting techniques which have also been applied to class imbalance problems. Fan et al. [22] proposed AdaCost, Ting [75] proposed CSB0, CSB1, and CSB2, and Sun et al. [73] proposed AdaC1, AdaC2, and AdaC3, all of which adjust the weight update process in AdaBoost to minimize the misclassification cost rather than training error.

2.3 Classifying Data That is Noisy and Imbalanced

The presence of class imbalance requires special attention in other aspects of data mining. For example, feature selection can be affected by skewed class distributions [37, 29]. Similarly, when class noise and class imbalance are combined, the negative effects on classification performance can be even worse. Noisy data especially harms the minority class. Minority class instances which are mislabeled as the majority class cause the imbalance to be even more severe. On the other hand, majority instances which are incorrectly labeled as the minority class obfuscate the patterns of the actual minority instances, making them even more difficult for classifiers to discern. One theory proposed in related research [40, 84] is that class imbalance by itself is not the primary cause of poor classifier performance, but instead other issues such as small disjuncts, overlapping classes, concept complexity, or absolute rarity are much more important. Class noise can be a cause of all of these issues. For example, noisy data generally increases concept complexity and can create small disjuncts in the data. Despite the increased severity of the problem caused by the union of both class noise and class imbalance, relatively little research has been performed to analyze solutions to both problems at once. Zhu and Wu [92] propose a Cost-guided Iterative Classification Filter (CICF) for removing noisy instances for cost-sensitive learning. C5.0 was used to test CICF, bagging, and boosting with various misclassification costs using 20 datasets, including some imbalanced datasets. Anyfantis et al. [5] examine the classification performance of Naive Bayes, C4.5, and

5NN in conjunction with several cost sensitive classification techniques using several moderately imbalanced datasets. Huang et al. [34] investigated the impact of data cleansing on learning from imbalanced credit assessment data. Van Hulse et al. [79] studied the impact of class noise on the classification performance of 11 different classifiers on imbalanced data. The work showed that mislabeled minority class examples have a greater impact on classification performance than mislabeled majority class examples.

CHAPTER 3

EXPERIMENTAL METHODOLOGIES

To explore, in detail, the effects of various classification techniques on noisy, imbalanced data, the WEKA machine learning tool [87], written in Java, was used. Section 3.1 provides information about the datasets used for experimentation. The two performance metrics utilized for performance analysis are described in Section 3.2. Section 3.3 provides the details of the classification filtering procedure used throughout this work. Finally, Section 3.4 outlines the analysis of variance model for testing the statistical significance of the empirical results.

3.1 Datasets

In total, 35 real world datasets were used for the experiments in this work. All of them are imbalanced, ranging from 1.33% to 34.90% minority instances. Some are proprietary datasets, while the others were aquired from the UCI Machine Learning Repository [8]. Many of the datasets originally contained multiple class values, but were transformed into binary class problems by leaving one class as the minority class and combining the rest of the classes into one large majority class. The datasets

Dataset	# Positive	# Negative	# Total	% Positive	% Negative
SP3	47	3494	3541	1.33%	98.67%
SP4	92	3886	3978	2.31%	97.69%
MAMMOGRAPHY	260	10923	11183	2.32%	97.68%
SOLAR-FLARE-F	51	1338	1389	3.67%	96.33%
CAR-3	69	1659	1728	3.99%	96.01%
SP2	189	3792	3981	4.75%	95.25%
CCCS-12	16	266	282	5.67%	94.33%
SP1	229	3420	3649	6.28%	93.72%
PC1	76	1031	1107	6.87%	93.13%
MW1	31	372	403	7.69%	92.31%
GLASS-3	17	197	214	7.94%	92.06%
KC3	43	415	458	9.39%	90.61%
CM1	48	457	505	9.50%	90.50%
CCCS-8	27	255	282	9.57%	90.43%
PENDIGITS-5	1055	9937	10992	9.60%	90.40%
SATIMAGE-4	626	5809	6435	9.73%	90.27%
ECOLI-4	35	301	336	10.42%	89.58%
SEGMENT-5	330	1980	2310	14.29%	85.71%
KC1	325	1782	2107	15.42%	84.58%
JM1	1687	7163	8850	19.06%	80.94%
LETTER-VOWEL	3878	16122	20000	19.39%	80.61%
CCCS-4	55	227	282	19.50%	80.50%
KC2	106	414	520	20.38%	79.62%
CONTRACEPTIVE-2	333	1140	1473	22.61%	77.39%
VEHICLE-1	212	634	846	25.06%	74.94%
HABERMAN	81	225	306	26.47%	73.53%
YEAST-2	429	1055	1484	28.91%	71.09%
PHONEME	1586	3818	5404	29.35%	70.65%
CCCS-2	83	199	282	29.43%	70.57%
CREDIT-GERMAN	300	700	1000	30.00%	70.00%
DIABETES	268	500	768	34.90%	65.10%
Datasets for noise injection					
NURSERY-3	328	12632	12960	2.53%	97.47%
LETTER-A	789	19211	20000	3.95%	96.06%
OPTDIGITS-8	554	5066	5620	9.86%	90.14%
SPLICE-2	768	2422	3190	24.08%	75.92%

Table 3.1: Characteristics of experimental datasets

selected for use in these experiments, as well as information about their size and class distributions, can be found in Table 3.1 and in the list that follows.

1. The Glass dataset comes from the UCI repository and has 214 instances with 9 numeric independent attributes and the nominal class attribute. It originally contains six classes, however, the dataset was transformed to have a binary class. The third class, containing 17 instances, 7.94% of the total, was kept as the minority class, while the rest of the classes were combined together into the majority class (92.06%).

2. The CCCS-12 dataset has 282 instances with 8 numeric independent attributes. It has 16 examples in the minority class (5.67%) and 266 examples in the majority class (94.33%).

3. The CCCS-8 dataset has 282 instances with 8 numeric independent attributes. There are 27 examples in the minority class (9.57%) and 255 examples in the majority class (90.43%).

4. The CCCS-4 dataset has 282 total instances and 8 numeric independent attributes. The minority class contains 55 examples (19.50%) and the majority class contains 227 examples (80.50%).

5. The CCCS-2 dataset has 282 instances with 8 numeric attributes in addition to the nominal class attribute. The minority class has 83 examples (29.43%) and the majority class has 199 examples (70.57%).

6. The Haberman dataset comes from the UCI repository. It has 306 total instances and 3 independent attributes. The UCI website describes all 3 as numeric, however we treated the second attribute as nominal. The minority class contains 81 examples (26.47%) and the majority class has 225 examples (73.53%).

7. The E.coli dataset has 336 instances, 7 numeric independent attributes, and comes from the UCI repository. It has 35 examples (10.42%) in the minority class and 301 examples (89.58%) in the majority class. The dataset originally has 8 class values, so to convert it to a binary class we used the fourth value as the minority class and combined all other values for that attribute into a single minority class.

8. The MW1 dataset consists of 403 total instances with 15 numeric independent attributes. The minority class has 31 examples (7.69%) and the majority class has 372 (92.31%) examples.

9. The KC3 dataset contains 458 instances with 15 numeric independent attributes. Its minority class has 43 examples (9.39%) and its majority class contains 415 examples (90.61%).

10. The CM1 dataset has 505 total instances and 15 numeric independent attributes in addition to the nominal class attribute. It has 48 examples (9.50%) in the minority class and 457 examples (90.50%) in the majority class.

11. The KC2 dataset has 520 total instances with 15 numeric independent attributes. The minority class has 106 examples (20.38%) and the majority class has 414 examples (79.62%).

12. The Pima Indian Diabetes dataset comes from the UCI repository. It is a two-class dataset containing 768 examples with eight numeric attributes and the nominal class attribute. The minority class has 268 instances, representing 34.90% of the total number of examples, while the majority class has 500 examples (65.10%).

13. The Vehicle dataset has 856 instances, 18 numeric independent attributes, and comes from the UCI repository. The dataset has 212 instances in the minority class (25.06%) and 634 instances in the majority class (74.94%). It originally contains 4 class attribute values. To achieve two classes, we kept the first value as the minority class and combined the other values into one majority class.

14. The German Credit dataset from the UCI repository has 1000 total instances, 7 numeric independent attributes, and 13 nominal independent attributes. The minority class contains 300 examples (30.00%) and the majority class contains 700 examples (70.00%).

15. The PC1 dataset contains 1107 examples and has 15 numeric independent attributes. Its minority class has 76 instances (6.87%) and its majority class has 1031 instances (93.13%).

16. The Solar Flare dataset from the UCI repository contains 1389 instances. We used the first attribute as the class attribute. It originally contains 7 values, so we isolated the sixth value (F) as the minority class, and combined the others into the majority class. We also treated the last three attributes as nominal, though the UCI website describes them as numeric. This led to all 12 independent attributes being nominal. It also left 51 instances in the minority class (3.67%) and 1338 instances in the majority class (96.33%).

17. The Contraceptive dataset from the UCI repository has 1473 total instances with 9 independent attributes, 7 of which are nominal and 2 of which are numeric. The class attribute originally has 3 values, so the second one was kept as the minority class and the other two were combined into the majority class. This left 333 minority instances (22.61%) and 1140 majority instances (77.39%).

18. The Yeast dataset from the UCI repository has 1484 examples and 8 independent numeric attributes. The class attribute originally has 10 different values, so the second one was kept as the minority class with 429 instances (28.91%) and the others were combined to form the majority class with 1055 instances (71.09%).

19. The Car dataset from the UCI repository contains 1728 total instances with 6 nominal independent attributes. It originally has 4 different class attribute

values, so the third was isolated as the minority class with 69 instances (3.99%) and the rest were combined into the majority class with 1659 instances (96.01%).

20. The KC1 dataset has 2107 total instances with 15 independent numeric attributes. The minority class has 325 examples (15.42%) and the majority class has 1782 examples (84.58%).

21. The Segment dataset from the UCI repository has 2310 total examples and 19 numeric independent attributes. Its nominal class attribute originally contains 7 different values, so we used the fifth value as the minority class and combined the others to form the majority class. This minority class contains 330 instances (14.29%) and the majority class contains 1980 instances (85.71%).

22. The Splice Junction dataset from the UCI repository has 3190 total instances and 60 nominal independent attributes (after removing the 'Instance_name' attribute which has a unique value for each example) in addition to the class attribute. It originally has 3 different values of the class attribute, so we used the second value as the minority class and combined the other 2 into the majority class. The minority class then contained 768 examples (24.08%) and the majority class contained 2422 examples (75.92%).

23. The Software Project 3 dataset has 3541 total instances and 42 numeric independent attributes. The minority class has 47 instances (1.33%) and the majority class has 3494 examples (98.67%).

24. The Software Project 1 dataset has 3649 total examples with 42 numeric independent attributes. It has 229 instances (6.28%) in the minority class and 3420 instances (93.72%) in the majority class.

25. The Software Project 4 dataset has 3978 instances and 42 numeric independent attributes. The minority class contains 92 examples (2.31%) and the majority class contains 3886 examples (97.69%).

26. The Software Project 2 dataset has 3981 total instances and 42 numeric independent attributes. It has 189 instances in the minority class (4.75%) and 3792 instances in the majority class (95.25%).

27. The Phoneme dataset was obtained from the ELENA project website [1]. It has 5404 total instances and 5 numeric independent attributes. The minority class has 1586 examples (29.35%) and the majority class has 3818 examples (70.65%).

28. The Optdigits dataset from the UCI repository has 5620 instances and 64 numeric independent attributes. Its class attribute originally has 10 different values, so the ninth was kept as the minority class with 554 instances (9.86%) and the others were combined to form the majority class with 5066 instances (90.14%).

29. The Satimage dataset from the UCI repository has 6435 instances with 36 numeric independent attributes. The class attribute originally contains 6 different

values, so we utilized the fourth as the minority class and combined the rest to form the majority class. This left a minority class with 626 instances (9.73%) and a majority class with 5809 instances (90.27%).

30. The JM1 dataset contains 8850 total instances and 15 numeric independent attributes. The minority class has 1687 examples (19.06%) and the majority class has 7163 examples (80.94%).

31. The Pendigits dataset from the UCI repository has 10,992 total instances and 16 numeric independent attributes. It originally has 10 different class attribute values, so we used the sixth one as the minority class and combined the rest into a single majority class. The minority class contains 1055 instances (9.60%) and the majority class contains 9937 instances (90.40%).

32. The Mammography dataset has two classes and contains 11,183 instances, each with six numeric attributes and one nominal class attribute. The minority class contains 260 instances, representing 2.33% of the total, while the majority class contains 10,923 instances (97.68%).

33. The Nursery dataset from the UCI repository has 12,960 total examples and 8 nominal independent attributes. It originally has 5 different values in the class attribute, so we used the third value as the minority class with 328 instances (2.53%) and combined the rest of the values into a majority class with 12,632 instances (97.47%).

34. The Letter-A dataset is derived from the UCI Letter dataset. It has 20,000 total instances and 16 numeric independent attributes. The class attribute contains values for all the letters of the alphabet, so for this dataset we used ‘A’ as the minority class value and combined the other letters into one majority class. This minority class has 789 instances (3.95%) and the majority class has 19,211 instances (96.06%).

35. The Letter-Vowel dataset is also derived from the Letter dataset from the UCI repository, and thus also has 20,000 examples and 16 numeric independent attributes. This time, all five vowels (not ‘Y’) were combined to form the minority class while the rest were combined to form the majority class. This minority class contains 3878 instances (19.39%) and the majority class contains 16,122 instances (80.61%).

3.2 Performance Metrics

This study deals only with the binary classification problem. Binary classifiers make predictions that can be grouped into one of four categories. If x_i is an example from a dataset D with class c_j, $j = 0$ or 1, and if $c(x_i)$ and $\hat{c}(x_i)$ are the actual and predicted classes of x_i, then:

$$x_i \text{ is a } \textit{true positive} \text{ (tpos) if } \; c(x_i) = c_1 = \hat{c}(x_i).$$

$$x_i \text{ is a } \textit{true negative} \text{ (tneg) if } \; c(x_i) = c_0 = \hat{c}(x_i).$$

$$x_i \text{ is a } \textit{false positive} \text{ (fpos) if } \; c(x_i) = c_0 \neq \hat{c}(x_i).$$

$$x_i \text{ is a } \textit{false negative} \text{ (fneg) if } \; c(x_i) = c_1 \neq \hat{c}(x_i).$$

The total number of instances in a dataset that are false negatives are denoted $\#fneg$, and similarly for $\#fpos$, $\#tpos$, and $\#tneg$. If N_{c_j} denotes the number of instances in class c_j, then let $N = N_{c_0} + N_{c_1}$. The seven basic measures of classifier performance can be defined as:

$$\text{overall accuracy} = \frac{\#tpos + \#tneg}{N} \tag{3.1}$$

$$\text{misclassification rate} = \frac{\#fpos + \#fneg}{N} \tag{3.2}$$

$$\text{true positive rate} = \frac{\#tpos}{N_{c_1}} \tag{3.3}$$

$$\text{true negative rate} = \frac{\#tneg}{N_{c_0}} \tag{3.4}$$

$$\text{false positive rate} = \frac{\#fpos}{N_{c_o}} \tag{3.5}$$

$$\text{false negative rate} = \frac{\#fneg}{N_{c_1}} \tag{3.6}$$

$$\text{precision} = \frac{\#tpos}{\#tpos + \#fpos} \tag{3.7}$$

Overall classification accuracy (Eq. 3.1), or its complement, misclassification rate (Eq. 3.2), are commonly used to evaluate the performance of a classifier, often using the default threshold $t = 0.5$. However, such measures of classification performance are inadequate when false negatives and false positives carry different costs, or when the prior probabilities of the classes are uneven. Folleco et al. [25] examine four different performance metrics for low quality imbalanced data. In this work, we use two performance metrics that are more appropriate for evaluating models when data is imbalanced.

3.2.1 ROC Curves

One of the most popular methods for evaluating the performance of learners for class imbalance cases is *Receiver Operating Characteristic* [59], or ROC, curves. ROC curves graph true positive rate on the y-axis versus the false positive rate on the x-axis. The ROC curve is determined by calculating the true positive and false positive rates as the decision threshold varies from 0 to 1. The resulting curve illustrates the trade-off between detection rate and false alarm rate. The ROC curve illustrates the performance of a classifier across the complete range of possible decision thresholds, and accordingly does not assume any particular misclassification costs or class prior probabilities. The area under the ROC curve (AROC) is used to provide a single numerical metric for comparing model performances.

3.2.2 PRC Curves

Another curve, the *Precision-Recall Characteristic* (PRC) [51] curve has also been used to evaluate the performance of models built using skewed data. Like ROC curves, the PRC curve plots the *recall*, or true positive rate, on the y-axis. However, the difference between ROC and PRC curves is that PRC curve plots *precision* on the x-axis. Precision is the proportion of examples predicted as belonging to the positive class that actually are positive examples. The implications of this difference have been investigated by Davis and Goadrich [17]. As with ROC curves, the area under the PRC curve (APRC) is used as a single metric for comparing the performances of different models.

3.3 Classification Filtering Procedure

Classification filtering is a data preprocessing technique for handling class noise. It is performed by building and evaluating a selected classifier (or ensemble of classifiers) on the training data, and identifying misclassified instances as being potentially noisy. The classifier building/evaluation process is typically performed using cross-validation. In cross-validation, the data is divided into k folds of equal size, typically each with the same class distribution as the entire dataset. In each iteration, $\frac{k-1}{k}$ of the folds are combined together to use as training data, and the remaining fold is used as test data. This is repeated k times so that each fold is used as test data once. After performing cross-validation to collect predictions on each of the training instances, the desired instances can be officially determined as noisy and dealt with accordingly. One must first choose what classifier to use to perform the filtering (we call this the 'FilterLearner'). Once the evaluation is performed, the method for dealing with them must be performed (we call this the ‘Treatment’). Noisy instances identified can be either removed from the training dataset (‘Remv’), or have their class changed to the opposite one (‘Rlbl’). The ‘BothClasses’ option refers to the choice of either treating noisy instances from both classes (‘Both’) or only from the less sensitive majority class (‘Maj’). When the Maj option is being used, all of the minority instances were included in the training set (and excluded from the test set) for each fold of cross-validation to improve the minority class performance. Finally, the method for determining which instances will be considered noisy must be made (‘Threshold’). Typically, the default classification provided by the learner is used. Many learners can provide predicted probabilities of membership for each class though. The instances with the highest probability of membership in the opposite class seem the most noisy. One can change the decision threshold to adjust how many instances will actually be treated as noise. The default threshold on this probability (0.5 in the case of a binary dataset), often is inappropriate for imbalanced data. Throughout this research we use 10 for the number of folds to use for cross-validation. The steps of the classification filtering algorithm we implemented are given in Figure 3.1.

3.4 Analysis of Variance

An analysis of variance (ANOVA) model [7] is used to understand the impact of experimental factors (or independent variables) on the performance measure(s) (or

```
**Algorithm ClassificationFilter**
**Given:** Set $S$ of examples $(x_1, y_1), ..., (x_m, y_m)$ with minority class $y^0$ and majority class $y^1$, $|Y| = 2$
Filtering learner: *Learn*
Number of cross-validation folds: $T$
Whether to identify noise from both classes: *Both*
Whether to remove noisy instances (rather than relabeling them): *Remv*
Decision thresholds for for identifying instances of each class as noisy: $D_{y^0}$ and $D_{y^1}$

  1. Perform cross-validation on training data:
     Do for $t = 1, 2, ..., T$
       (a) Create temporary training dataset $S_t$: Add the $t$th set of $\frac{T-1}{T}$ of the majority instances to $S_t$.
             • If Both is true, add the $t$th set of $\frac{T-1}{T}$ of the minority instances to $S_t$.
             • If Both is false, add all of the minority instances to $S_t$.
       (b) Create temporary test dataset $E_t$:
             i. Add remaining $\frac{1}{T}$ of the majority instances to $E_t$.
             ii. If Both is true, add remaining $\frac{1}{T}$ of the minority instances to $S_t$.
       (c) Train Learn on $S_t$ to get model $h_t$.
       (d) Evaluate $h_t$ on $E_t$:
             i. For each majority class instance in $E_t$, store its predicted probability of membership in the minority class in $P_{y^1}$.
             ii. If Both is true, for each minority class instance in $E_t$, store its predicted probability of membership in the majority class in $P_{y^0}$.
  2. Treat the noisy instances:
       (a) For each majority instance $i$ such that $P_{y^1} \geq D_{y^1}$:
             • If Remv is true, remove instance $x_i$ from the training data.
             • If Remv is false, change $y_i$ to $y^0$.
       (b) For each minority instance $j$ such that $P_{y^0} \geq D_{y^0}$:
             • If Remv is true, remove instance $x_j$ from the training data.
             • If Remv is false, change $y_j$ to $y^1$.
  3. Output the final, filtered dataset $S'$.
```

Figure 3.1: The classification filtering algorithm for binary imbalanced data

dependent variable(s)). If N observations with an overall mean performance μ are randomly partitioned into k subsets, then it is expected that the mean of each subset, μ_k, is very similar to the overall mean μ. If, however, the observations were given one of k different treatments, does the treatment significantly change the average performance? In other words, is $\mu_b \neq \mu_a$ for some a and b, $1 \leq a, b \leq k$. In other words, if the treatment has no impact on the dependent variable, then μ_b and μ_a would not be significantly different. On the other hand, if the treatment had a significant effect on the dependent variable (assuming certain assumptions are satisfied), then the mean performance is impacted by the treatment. In this manner, an ANOVA model allows the analyst to evaluate how the various experimental factors influence the dependent variable.

The experimental attributes being tested are called *main effects* or *factors*. Crossed effects, which consider the interaction between two or more main effects, can also be included in the model. ANOVA can be used to test the hypothesis that the average performance for each level of the main factors are equal against the alternative hypothesis that at least one is different. If the alternative hypothesis (i.e., that at least one average performance is different) is accepted, numerous procedures can be used to determine which of the values are significantly different from the others. This involves a comparison of two average performance values, with the null hypothesis that they are equal. A Type I error occurs when the null hypothesis is incorrectly rejected. In this study, we use Tukey's Honestly Significant Difference (HSD) test

[63], which controls the Type I experiment-wise error rate [7].

CHAPTER 4

CLASSIFIER BASED NOISE DETECTION EFFICIENCY

4.1 Introduction

Quality of data is an extremely important facet of the real world application of machine learning techniques. Classification performance can be quite negatively impacted by noise in the examples used for induction. Several general techniques have been investigated and applied for handling class noise. One is to attempt to increase the robustness of a learner to noisy instances in the training data [28]. Another, noise filtering [49], removes examples from the training data that are identified as noisy. A third, noise correction [74], identifies potentially noisy instances and attempts to correct them. For both filtering and correction, the identification of the noisy examples is the key. Typically this involves classifying the training data using some classifier [86], or ensemble of classifiers [12], and considering the misclassified instances to be mislabeled. To our knowledge, however, no research has been done comparing different classification schemes to identify which are the most successful at identifying mislabeled examples, especially in the presence of class imbalance. Our goal in this study is to identify which learners work best for identifying class noise, and how the level of class imbalance, the amount of class noise, and the class distribution of the noise affects the filtering performance of the classifers. To achieve this we perform a broad suite of experiments comparing eight popular classification algorithms on their ability to efficiently identify class label noise in imbalanced data. In these experiments, we inject noise into real world datasets, controlling both the level of noise as well as the amount of noise coming from the instances of each class. This is in addition to controlling the class distribution of the datasets in order to gauge the effects of different levels of class imbalance.

Section 4.2 of this chapter summarizes the experimental methodology used in these experiments, including descriptions of the datasets, classification algorithms, and noise injection procedures utilized. The results of our empirical experimentation are presented in Section 4.3. Finally, a summary of the findings is provided in Section 4.4.

4.2 Experiments

In order to assess the noise detection abilities of the eight classifiers used in this study, the WEKA machine learning tool [87], written in Java, was used. Section 4.2.1 provides information about the datasets used for this empirical experimentation. Section 4.2.2 describes the classification algorithms used. The noise detection procedure is described in Section 4.2.3. Finally, Section 4.2.4 details the overall experimental design, including the data sampling and noise injection procedures.

4.2.1 Datasets

The experiments in this work are based on four datasets, all acquired from the UCI Repository [8]. Each of the datasets originally contained multiple class values, but were transformed into binary class problems as described below. The datasets selected for use in these experiments, as well as information about their size and class distributions, can be found in Table 4.1. The first row of Table 4.1 contains the name of the dataset. The row labeled "Size" indicates the total number of examples available in the given dataset. The rows "#min" and "%min" identify the number of minority class examples, and the percentage of all examples belonging to the minority class, respectively. We selected these four datasets due to their sizes, and more importantly the number of minority class examples. These characteristics facilitiated our experimental design, as decribed in Section 4.2.4. Most importantly, however, is that these datasets are relatively clean. That is, models constructed on these datasets (using ten runs of ten-fold cross validation) resulted in nearly perfect classification performance prior to noise injection.

	LetterA	Nursery3	OpdDigits8	Splice2
Size	20000	12960	5620	3190
#min	789	328	554	768
%min	3.95	2.53	9.86	24.08

Table 4.1: Characteristics of initial datasets

The LetterA dataset is based on a letter recognition database originally containing 26 different classes (one for each letter). The class attribute was transformed to a binary class by selecting the letter "A" to be the positive class and the remaining letters to be the negative class. The Nursery3 dataset is based on a dataset used to rank nursery schools. The dataset originally contained 5 classes, one of which ("very recommended") was selected to be the positive class, while the remaining class values were grouped together to make up the negative class. The Optdigits8 dataset is used to train models to recognize handwritten digits from zero to nine. The digit "8" was selected as the positive class, and the remaining digits make up the negative class. The Splice2 dataset is a molecular biology dataset used to identify different "splice junctions" on a DNA sequence. The class "ie" (known as "acceptors") was selected as the positive class, while the classes "ei" ("donors") and "Neither" were combined to form the negative class.

4.2.2 Learners

All eight classifiers were used as implemented in the WEKA machine learning tool [87]. Unless otherwise noted, the default parameters for the classifier were used.

4.2.2.1 C4.5

C4.5 is the benchmark decision tree learning algorithm proposed by Quinlan [61]. C4.5 is among the most commonly used learning algorithms in data mining research. The decision tree is built using an entropy-based splitting criterion stemming from information theory [2]. C4.5 improves Quinlan's older ID3 [60] decision tree algorithm by adding support for tree pruning and dealing with missing values and numeric attributes. The WEKA version of C4.5 is called J48 and we refer to this learner in the paper as C4.5N. Two of the paramaters were changed from their default. The ‘unpruned’ parameter was set to true to disable pruning, and the ‘use-Laplace’ parameter was set to true to enable Laplace smoothing of the probability estimates [85].

4.2.2.2 Naive Bayes

Naive Bayes [54] (NB) is a quick and simple classifier that utilizes Bayes’s rule of conditional probability. It is ‘naive’ in that it assumes that all predictor variables are independent. Although this assumption rarely holds true in real-world data, Naive Bayes has been shown to often perform well, even in the presence of strong attribute dependencies [19]. We use the default parameters for Naive Bayes in our experiments.

4.2.2.3 Multilayer Perceptron

Multilayer perceptrons [62] (MLP) attempt to artificially mimic the functioning of a biological nervous system. Multiple nodes, or ‘neurons,’ are connected in layers, with the output of each node being the thresholded weighted sum of its inputs from the previous layer. The network weights are usually learned using a gradient descent algorithm called back-propagation. It has been shown that a multiple hidden layer neural network can approximate any function [38]. We change two parameters in our experiments. The ‘hiddenLayers’ parameter was changed to 3 to define a network with one hidden layer containing three nodes, and the ‘validationSetSize’ parameter was changed to 10 so that 10% of the data would be used as a validation set to determine when to stop the iterative training process.

4.2.2.4 K Nearest Neighbors

K nearest neighbors [3], abbreviated kNN, is a "lazy learning" technique. The training dataset serves as a case library which is used to predict the class of an instance based on it's k 'nearest neighbors' in the case library. The nearest neighbors are those instances in the case library that are closest in the feature-space to the instance being classified based on some distance measure, usually Euclidean distance. Denoted IBk in WEKA, we set the 'KNN' parameter to 5 to indicate the use of five nearest neighbors (we refer to this learner henceforth as 5NN) and the 'distanceWeighting' parameter was set to 'Weight by 1/distance.'

4.2.2.5 Support Vector Machines

A *support vector machine* (SVM) [65] is a linear classifier which attempts to learn the maximum margin hyperplane separating two classes in the data. This maximum margin hyperplane is usually determined by a small subset of instances called *support vectors*. In order to learn nonlinear decision boundaries, the data can be transformed by a kernel function. The linear maximum margin hyperplane then found for the transformed data can represent a nonlinear partitioning of the original feature space. SMO is the implementation of support vector machines in WEKA. Two changes to the default parameters were made: the complexity constant 'c' was changed from 1.0 to 5.0, and the ‘buildLogisticModels’ parameter, which allows proper probability estimates to be obtained, was set to true. By default, SMO uses a linear kernel.

4.2.2.6 Random Forests

Random forests [10] (RF100) give accuracy that compares favorably with Adaboost [26] and is relatively robust to noise. This learner utilizes bagging [9] and the ‘random subspace method’ [33] to construct an ensemble (forest) of randomized, unpruned decision trees. The outputs of this forest of decision trees are combined to produce the ultimate prediction. The ‘numTrees’ parameter was changed to 100 to specify using a forest of 100 trees.

4.2.2.7 Radial Basis Function Network

A *radial basis function network* [56] (RBF), like MLP, is a type of artificial neural network. The radial basis function outputs a value based on the radial distance of a feature vector from a ‘center’ in the feature space. RBFs typically have two processing layers. The data is first input to each RBF in the first layer. The second layer then linearly combines the output of the radial basis functions to produce the final output. The only parameter change for RBF (called RBFNetwork in WEKA) was to set the parameter ‘numClusters’ to 10.

4.2.2.8 Logistic Regression

Logistic regression (LR) is a a type of generalized linear model which can be used to predict a binary dependent variable [31]. While LR does allow categorical or continuous independent variables, categorical variables must be encoded (e.g., 0,1) to facilitate classification modeling. No changes to the default parameter values were made.

4.2.3 Noise Detection Procedure

The noise detection procedure we utilize in these experiments is, for the most part, a standard classification filter as described in Section 3.3. Classification filters build and evaluate a chosen classifier on the training data and then any instances that are misclassified are considered to be noisy, and subsequently are removed from the data or have their class label changed. In our application, instead of judging instances to be noisy based on a standard default classification of one class or the other, we examine which instances are determined to be noisy as the decision threshold is varied between 0 and 1. Since we are only examining the accuracy of identifying mislabeled instances in this work, we don't actually perform removal or relabeling of the instances that are determined as noisy.

We apply 10-fold cross-validation to the training data using one of the eight classifiers listed in Section 4.2.2 and collect the classification results. Each instance will have a predicted probability of membership in each class. For a given instance, the two predicted probabilities (probability of membership in the majority class and probability of membership in the minority class) sum to 1. Separately for each class, we sort the instances labeled as that class by their predicted probability of membership in the opposite class. The instances with the highest such probabilities are the ones that the classifier has deemed most unlike the class they're labeled as (and thus most likely to be noisy). The threshold on these probabilities then determines which of the instances are officially deemed noisy. The threshold can be set explicitly (for example, it would be 0.5, the default classification threshold, for standard classification filtering), or it can be set dynamically if one desires, for example, a certain number or percentage of instances to be removed (or relabeled). In this work we don't choose a particular threshold, but vary it and record the resulting detection and false alarm rates for identifying mislabeled instances.

4.2.4 Experimental Design

While the basis for our experiments is composed of the four datasets described in Section 3.1, our experiments are performed using subsets of those datasets with controlled class distributions, noise levels, and noise distributions. For each combination of the different levels of these parameters, 100 independent trials are performed, so that the set of instances which are included in the subset and the set of those instances which have their class labels switched (from the minority class to the majority class or vice versa) is different each time. Each subset of the original data is sampled to contain 1500 examples and a class distribution according to our experimental parameters. These subsets are then corrupted to include the desired level and distribution of class noise. It is then, once this final, processed subset of data is constructed, that the noise detection procedure is applied. By keeping track of which instances were actually injected with noise and comparing those to the instances that are identified by the noise detection procedure, we can examine how good a classifier is for detecting actually noisy instances, rather than just report the effect the cleansing procedure has on classification performance after being applied to training data. The main experimental factors (other than the classifier used to perform noise detection), class distribution, noise level, and noise distribution, are described in this section.

4.2.4.1 Experimental Factor: Class Distribution

The first experimental factor, class distribution (CD), indicates the percentage of examples in the derived dataset belonging to the minority class. The experiments in this work consider six levels of class distribution, $CD = \{1, 2, 4, 6, 8, 10\}$, where the value of CD indicates the percentage of examples in the training data that belong to the minority class. For example, a derived dataset with $CD = 4$ has 4% of its examples from the minority class and 96% of its examples from the majority class. Note that this is the ratio prior to noise injection (discussed in the following sections). Depending on the noise distribution, the final dataset may contain a different class distribution.

4.2.4.2 Experimental Factor: Noise Level

The second factor, noise level (NL), determines the quantity of noisy examples in the training data. The selected datasets are relatively clean, so NL is varied by artificially injecting noise into the dataset. This is accomplished by swapping the class value of some of the examples. The number of examples with their classes swapped is a function of NL and the number of minority examples in the derived dataset.

While many works involving noise injection often inject noise by simply selecting $x\%$ of the examples and corrupting their class, this technique may be inappropriate when dealing with imbalanced datasets. For example, if a dataset contains only 1% of its examples belonging to the minority class, and as little as 10% of its examples are corrupted (injected with noise), the minority class will become overwhelmed by noisy examples from the majority class. Instead, we corrupt a percentage of the examples based on the number of minority examples in the dataset.

In our experiments, we use five levels of noise, $NL = \{10, 20, 30, 40, 50\}$, where NL determines the number of examples, based on the size of the minority class, that will be injected with noise. The actual number of noisy examples will be:

$$2 \times \tfrac{NL}{100} \times P$$

where P is the number of positive (minority) examples in the dataset. For example, a dataset with $CD = 10$ and $NL = 20$ will have 150 minority examples (10% of

1500) and $2 \times 0.2 \times 150 = 60$ noisy examples. Note that this does not indicate which examples (minority or majority) will be corrupted. That is determined by the final experimental factor: noise distribution.

4.2.4.3 Experimental Factor: Noise Distribution

The final experimental factor, noise distribution (ND), determines the type of noise that will be injected into the data. When dealing with binary class datasets (the only kind considered in this work), there are two possible noise types: P→N and N→P. Noise of type N→P is when a class that should be labeled "negative" is incorrectly labeled as "positive." Conversely, P→N noise is when an example that should be labeled "positive" is instead labeled "negative."

The experiments in this work use five levels of noise distribution, $ND = \{0, 25, 50, 75, 100\}$, where the value of ND indicates the percentage of noisy examples that are of the P→N variety. For example, if a derived dataset is to contain 60 noisy examples, and $ND = 25$, then 25% (15) of those noisy examples will be minority ("positive") class examples that have their labels changed to "negative" (P→N), while the remaining 75% (45) of the noisy examples will be of the N→P variety. Due to the definition of ND, the combination of $ND = 100$ and $NL = 50$ can not be used, since the resulting dataset would have zero minority class examples.

4.3 Results

This section presents the results of our experiments examining the efficiency of different learners for performing classification filtering to identify class label noise. Section 4.3.1 shows the performance for each level of the different experimental factors averaged across all other factors. Section 4.3.2 then illustrates the interaction of the learner with each of the three other factors: CD, NL, and ND. In Section 4.3.1, Figures 4.1 through 4.4 show how many instances had to be removed from a class (as a proportion of the total number of instances in that class) to successfully remove a certain percentage of the actually noisy instances with that class label. That is, the x-axis value (x) gives the percentage of actually noisy instances with that class label (which actually belong to the other class) while the corresponding y-axis value (y) gives the proportion of instances identified as noisy before successfully removing at least $x\%$ of the actually noisy instances. For clarity, only four x-axis values are shown (25%, 50%, 75%, and 100%). As explained in Section 4.2.3, the number of instances deemed noisy (and thus removed), is controlled by altering the threshold on the predicted probability of an instance's membership in the opposite class. If it's above the threshold, it's considered noisy and removed. Lowering the threshold increases the number of instances identified as noisy. The most efficient classifier for noise removal is one which, for a given level of actually noisy instances, produces the fewest instances above the threshold that aren't actually noisy.

Though it works slightly differently, this noise identification process is a lot like regular binary classification of instances. In regular classification, one adjusts the decision threshold on the probability of membership in the positive class for the optimal mix of maximizing successful identification of positive class instances (true positive rate) and minimizing incorrectly identifying negative class instances as positive (false positive rate). Changing the decision threshold changes these two values, and an ROC curve can be formed by plotting these points with true positive rate on the y-axis and false positive rate on the x-axis. In the noise identification procedure, the same probabilities are used, but instead of discriminating between the two classes in the dataset, for each class in the data, we are discriminating between instances that are correctly labeled and those that are incorrectly labeled. So, when performing this cleansing process, for each class in the data, a similar ROC curve can be created by plotting the true positive rate versus the false positive rate, where the positive class represents actually noisy instances with that class label (their real class labels should be the other class) and the negative class represents correctly labeled instances. From these noise-based ROC curves, as with standard classification performance ROC curves, the area under the curve (AUC) can be computed to give a general measure of the discrimination accuracy across the entire range of thresholds. Table 4.3 presents these noise-based AUC values for each of the single experimental factors, and Figures 4.5 through 4.7 show the AUC values for the interaction of each learner with the other experimental factors.

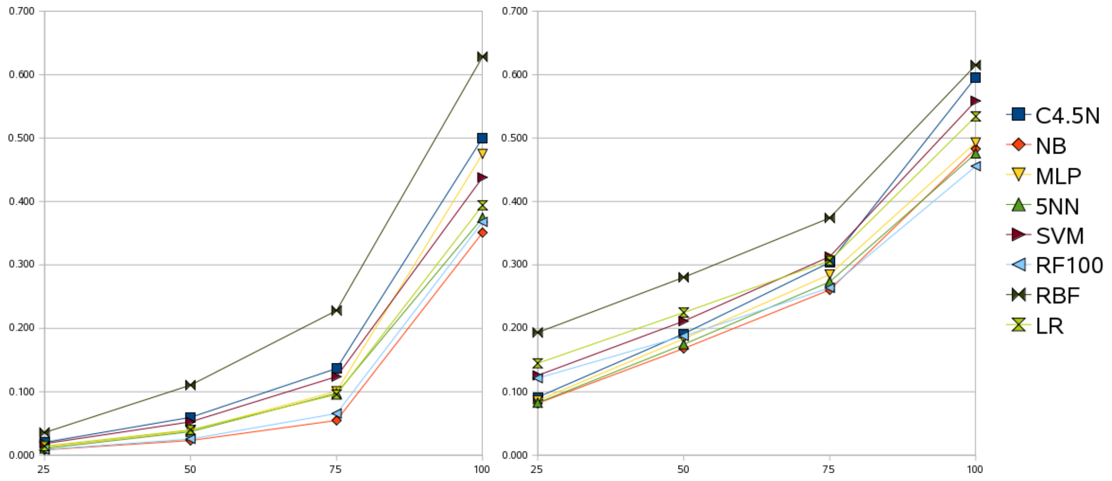

Figure 4.1: Noise identification efficiency per level of Lrn for the majority class (left) and minority class (right)

4.3.1 Individual experimental factors

Figure 4.1 displays the noise detection efficiency for each of the different learners in our experiments. The results for identifying noise in instances labeled as the majority class (P→N noise) are on the left side while the results for instances labeled as the minority class (N→P noise) are on the right. Each line represents a different learner. Lines that are lower in the graph represent learners that are better at identifying the mislabeled instances, and thus require less removal of correctly labeled instances when cleansing the dataset. One general characteristic displayed is the difference between the values for identifying noise in each of the classes (majority and minority). The values for the majority class are significantly lower than those for the minority class, a property which holds true for each of the experimental factors examined. On average, to successfully remove 25% of the noisy majority instances required removing less than 5% of the majority instances, and successfully removing all noisy majority instances required removing between 35% and 65% (50% if you don't count the worst performing learner) of the total number from that class. Meanwhile, for the minority class, successfully cleansing 25% of the noisy instances entailed removing between 8% and 20% of the total minority instances, and purging all of the noisy examples meant removing between 45% and 65% of all the examples of that class.

This disparity between the results for each class seems intuitive when examining the makeup of the datasets being cleansed. As described in Section 4.2.3, the number of noisy instances is based on the number of minority instances. The scenario which causes the most minority instances to be corrupted (thus there are the most noisy instances with a majority class label to be identified) is when CD=10, NL=40, and ND=100, meaning 80% of the minority instances (which represent 10% of the total number of instances) are relabeled as majority instances. This means that, at most, slightly over 8% of the majority instances will ever be actually noisy. This small subset of instances (presumably with a somewhat different pattern of independent attribute values since they actually come from the other class) should be comparatively easy to discriminate from the correctly labeled examples. Conversely, for the minority class, between approximately 5% (CD=10, NL=10, and ND=75) and 50% (NL=50 and ND=0 regardless of class distribution) of the instances can be mislabeled. When as many as half of the instances are noisy, it becomes much harder for a classifier to determine which are indicative of the actual class and which are mislabeled.

Another observation to note which also holds across all experimental factors and for both classes is the typically large jump in values between 75% and 100% on the x-axis. This indicates a significant increase in the difficulty discriminating the last 25% of noisy examples versus the previous 75%. In fact, though not shown in the figures, the slopes of the lines stay fairly constant past 75%, all the way up to 95% for the minority class, followed by a significant jump to reach 100%. For the majority class, the slopes typically start slowly increasing around 75% but again don't make very large jumps until the gap between 95% and 100%. This is due to the presence of some more atypical examples of each class. There might be a few examples of one class which tend to be closer in the feature space to the typical instances of the other class than most other instances in its class are. When such instances are relabeled (during the noise injection process), they are more difficult to discriminate from the actual instances of the newly labeled class because their attribute values are fairly similar. Thus the cleansing procedure may perform consistently as the amount of noise removed increases, up until the point where a small subset of atypical noisy instances are left which are difficult to find. For the datasets used in these experiments, that subset generally represented about 5% of the noisy instances, meaning up to 95% of the noisy examples could be removed with steady efficiency.

Looking at the performance of the individual learners (Figure 4.1), the most obvious feature is the poor results produced by RBF. For both majority (left) and minority (right) examples, RBF performs significantly worse than all the other learners for all four percentages (25%, 50%, 75%, 100%). For the majority class, the learners tend to group into consistent clusters from 25% to 75%. NB and RF100 perform pretty similarly to each other and better than the others, with NB having a slight edge. The next best group consists of 5NN, MLP, and LR. Above them, SVM and C4.5N perform similarly, beating only RBF. At 100%, though, the values are slightly more spread out and some of the relative slopes between different learners fluctuate. For the minority class, the learners don't rank as consistently as the noise identification rate (x-axis value) increases. For 25% through 75%, NB performs best, but moves to third at 100%. RF100 doesn't start off spectacularly at 25%, but by 75% is neck and neck with NB and is the best at 100%. C4.5N performs in contrast to RF100, performing well at 25%, losing ground steadily to close to last (other than RBF) by 75%, and then jumping more dramatically to last place (other than RBF) at 100%. 5NN performed well, ranking either second or third at all four percentages. MLP also performed decently, hovering around fourth. Finally, LR and SVM were towards the bottom of the rankings across all percentages.

Figures 4.2 through 4.4 show the noise detection results for the experimental factors NL, ND, and CD, respectively. These results don't hold too many surprises and are pretty consistent between the minority and majority classes. For the amount of noise added (NL), displayed in Figure 4.2, the results are as one would expect.

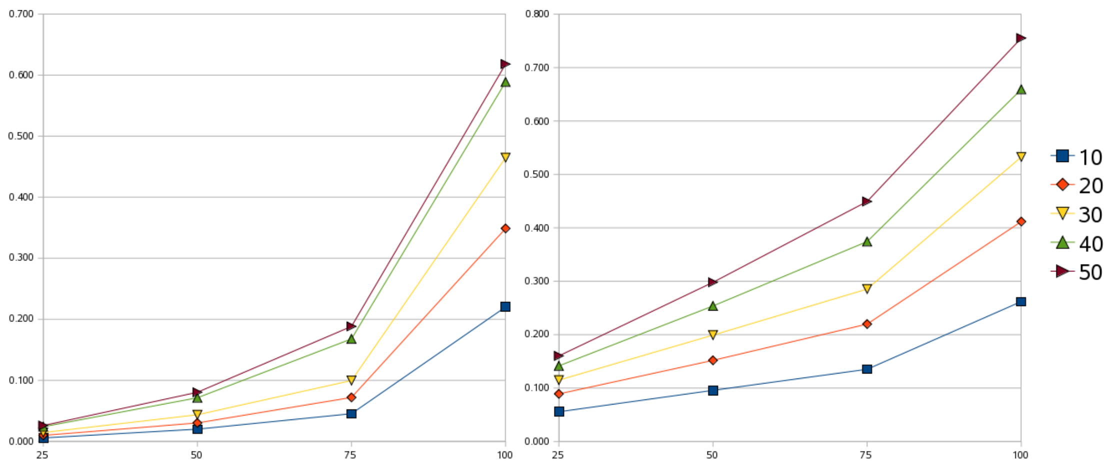

Figure 4.2: Noise identification efficiency per level of NL for the majority class (left) and minority class (right)

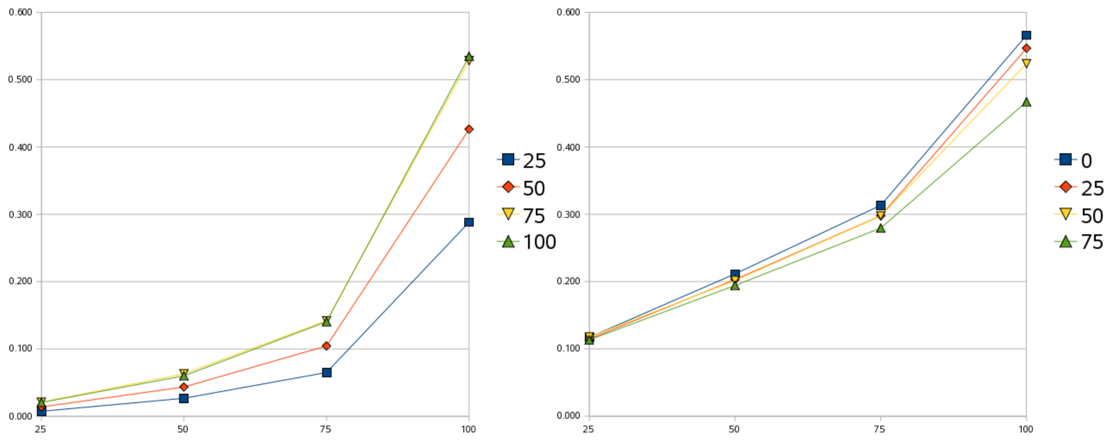

Figure 4.3: Noise identification efficiency per level of ND for the majority class (left) and minority class (right)

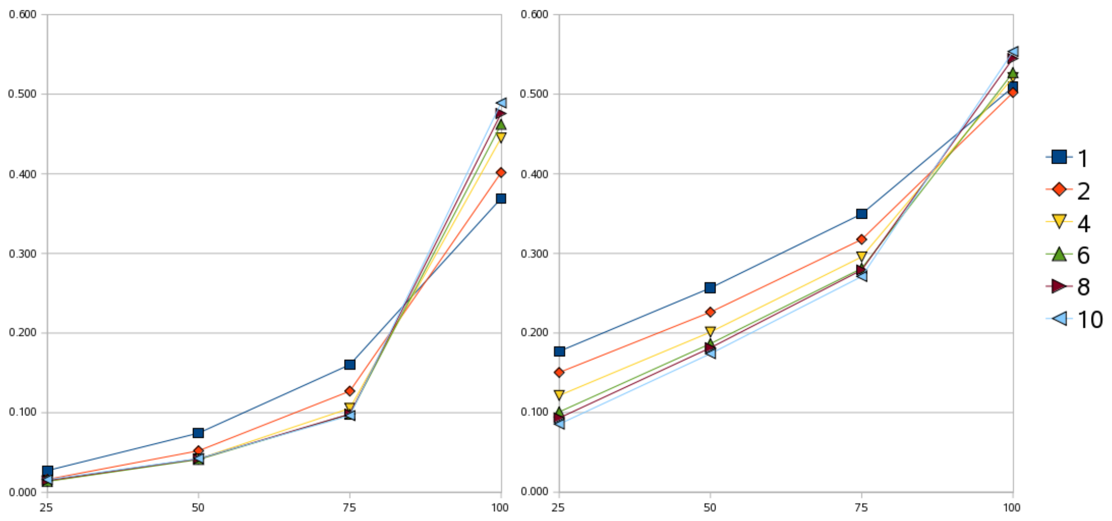

Figure 4.4: Noise identification efficiency per level of CD for the majority class (left) and minority class (right)

The performance decreases as the level of NL increases. Even with perfect accuracy, the proportion of instances removed (y axis value) would necessarily increase, though the slopes would be constant if the accuracy actually was perfect, which of course is not the case. Figure 4.3 illustrates the performance for each level of ND. These results are mostly intuitive as well. For the minority class, as the level of ND increases (meaning the number of noisy instances with this class label is decreasing), performance also increases. For the majority class, as the level of ND increases (meaning the number of noisy instances with this class label is also increasing), the performance decreases. The exception to this is that ND=75% performs slightly worse than ND=100%. This occurs because ND=100% doesn't occur at the highest level of noise (NL=50%, which when combined with ND=100% entails mislabeling all minority class instances as belonging to the majority class, leaves no minority class instances in the dataset). Finally, the class distribution is shown in Figure 4.4. For 25% through 75%, the different factors perform as one would probably expect, with the performance decreasing as the level of imbalance increases. Between 75% and 100%, however, the order roughly reverses, with the performance increasing with the level of imbalance. This is presumably because as the level of CD increases, so too does the number of noisy instances injected into the data. With larger numbers of noisy instances, the chances of including some of those previously discussed difficult to identify instances increases, leading to a larger jump at the end to remove the last few noisy instances. Sure enough (though not displayed for graphical clarity), the order stays consistent up until about 95%, at which point the less imbalanced datasets (with more noisy instances) produce larger jumps than the more imbalanced datasets, leading to the reversal.

		Maj-AUC		Min-AUC	
Factor	DF	F-stat	p-value	F-stat	p-value
Lrn	7	8721.41	<.0001	8814.75	<.0001
CD	5	2941.35	<.0001	4994.92	<.0001
NL	4	12889.3	<.0001	8636.95	<.0001
ND	3	7337.84	<.0001	3637.03	<.0001
CDxLrn	35	189.97	<.0001	306.11	<.0001
NLxLrn	28	293.29	<.0001	220.43	<.0001
NDxLrn	21	232.68	<.0001	164.12	<.0001

Table 4.2: ANOVA models of the AUC values for identifying majority and minority class noisy examples.

From the noise removal efficiency results, we calculated AUC values to summarize the performance across all percentages (the x-axis values in Figures 4.1 through 4.4). Table 4.2 presents the ANOVA table analyzing the impact of the four main

Factor	Maj-AUC		Min-AUC	
Learner				
C4.5N	0.904	G	0.905	G
NB	0.955	A	0.955	B
MLP	0.923	E	0.931	D
5NN	0.933	C	0.942	C
SVM	0.914	F	0.909	F
RF100	0.954	B	0.959	A
RBF	0.840	H	0.846	H
LR	0.931	D	0.920	E
Minority Class Distribution				
1%	0.887	F	0.881	F
2%	0.907	E	0.905	E
4%	0.921	D	0.922	D
6%	0.928	C	0.933	C
8%	0.933	B	0.939	B
10%	0.936	A	0.943	A
Noise Percentage				
10%	0.959	A	0.953	A
20%	0.943	B	0.940	B
30%	0.924	C	0.928	C
40%	0.888	D	0.906	D
50%	0.872	E	0.880	E
Minority Noise Percentage				
0%	–	–	0.938	A
25%	0.946	A	0.929	B
50%	0.924	B	0.916	C
75%	0.900	D	0.900	D
100%	0.906	C	–	–

Table 4.3: Average AUC and HSD grouping for each experimental factor in identifying majority and minority class noisy examples

experimental factors (CD, NL, ND, and Lrn), as well as the two way interactions involving Lrn, on these AUC values for both the minority and majority classes. As shown, each of the main factors, as well as the two way interactions, are statistically significant at the $\alpha = 5\%$ level for both the majority and minority classes. ANOVA analysis was performed using SAS [63].

ANOVA can be used to test the hypotheses that the average performance for each level of the main factors (learner, class distribution, noise level, and noise distribution) are equal against the alternative hypothesis that at least one is different. If the alternative hypothesis (i.e., that at least one average performance is different) is accepted, numerous procedures can be used to determine which of the values are significantly different from the others. This involves a comparison of two average performance values, with the null hypothesis that they are equal. A Type I error occurs when the null hypothesis is incorrectly rejected. In this study, we use Tukey's Honestly Significant Difference (HSD) test [63], which controls the Type I experiment-wise error rate [7].

Since each of the main factors are shown to be statistically significant, we perform Tukey's Honestly Significant Difference (HSD) test, indicating which levels of these factors result in significantly different performances when compared to the other levels of that factor. For example, the first part of Table 4.3 provides the results of the HSD test for the main factor Lrn. This factor has eight levels (the eight different learners used in our experiments), each of which is assigned to a group (indicated by a letter) based on its average performance (across all of the other factors, as well as 100 independent runs of each combination of main factors). Learners in group A performed better than those in group B, which performed better than those in group C, and so forth. For all four main factors, each level was significantly different from all the other levels, hence no two levels for a single main factor have the same letter in the HSD grouping column.

The AUC results in Table 4.3 agree completely with the observations made based on Figures 4.1 through 4.4. For both classes, NB and RF100 performed the best (though they performed significantly different from one another, it was by small amounts), while RBF was a distant last. The rest of the learners fell somewhere in the middle, with 5NN, MLP, and LR performing fairly well and C4.5N and SVM performing worse. Overall, the noise removal performance was directly proportional to the class distribution, i.e., it was easier to cleanse data with a more balanced class distribution. As more noise was injected into the data (factor NL), it became more difficult to remove the noise successfully. NL=10% produced the best results while NL=50% produced the worst. Finally, the performance decreased as more of the injected noise came originally from the minority class (factor ND), with the only exception being when ND=100% performed better than ND=75% for the majority class, due to the inability to include NL=50% when ND=100%.

4.3.2 Interactions involving the learner

In this section we examine the interaction between the factor Lrn and the three other main factors in the experiments. Figures 4.5 through 4.7 graph the different levels of the main factors (other than Lrn) on the x-axis versus the corresponding noise removal AUC values on the y-axis (results for the majority class are on the left and results for the minority class are on the right). Unlike the previous graphs, in these figures, higher curves represent better performance. Each graph contains a separate curve for each of the learners, as it is their relative noise removal performance

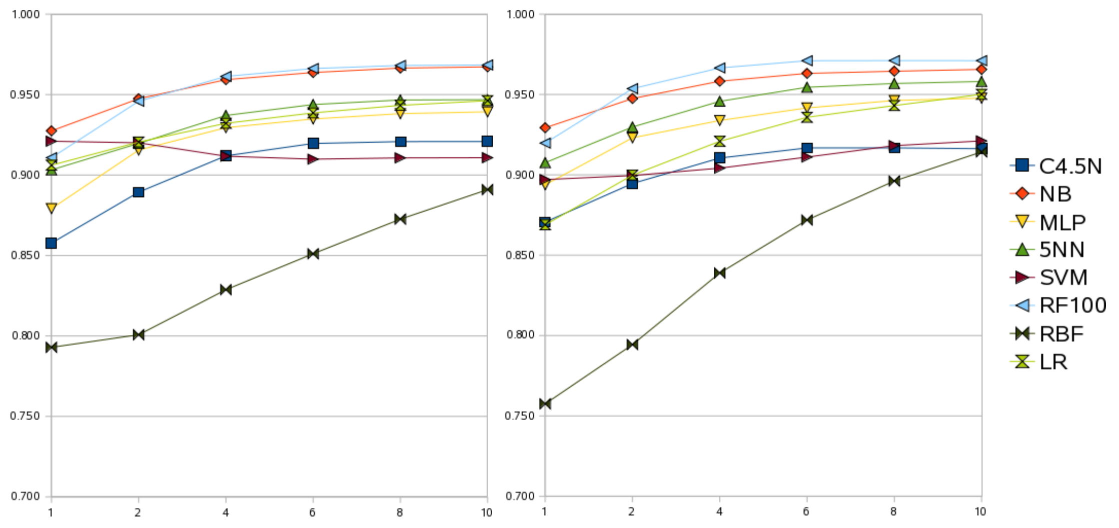

Figure 4.5: Noise identification AUC per learner at each level of CD for the majority class (left) and minority class (right)

that we are most interested in exposing in this study.

The overall noise removal efficiency (the AUC values) for each of the learners at each of the five class distributions is displayed in Figure 4.5. The results follow logically from the single factor results. RBF is the worst performer for every level of main factor CD. Across all distributions, NB and RF100 performed better than the others. For both classes, NB starts out with a performance edge at CD=1%, but RF100 surpasses it at CD=4% (for the majority class) or CD=2% (for the minority class) and stays slightly ahead the rest of the way. Again, 5NN, MLP, and LR generally perform better than every learner but NB and RF100. C4.5N increases the most (other than RBF) as the class distribution increases, but even at its best it produces some of the worst results. SVM behaves interestingly, with relatively little change as the class distribution increases. For the majority class it starts out with

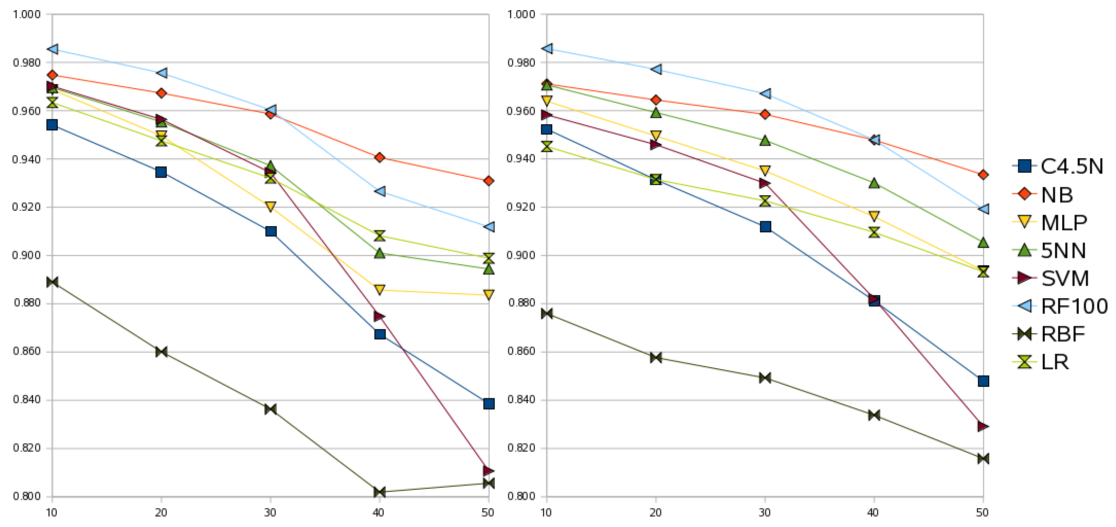

Figure 4.6: Noise identification AUC per learner at each level of NL for the majority class (left) and minority class (right)

the second best performance at CD=1%, but then actually decreases slightly across the subsequent levels of CD. For the minority class, SVM starts out in fourth place, but only slightly improves its performance as the distribution increases, ending up in sixth place at CD=10%. Overall, the curves show a logarithm-like pattern of growth, increasing more between the first several class distributions, but mostly stabilizing by around CD=6%. The exception to this pattern (other than RBF, which doesn't garner much interest due to its overall poor performance) is the logistic regression learner. It maintains a noticeably positive slope, even between CD=8% and CD=10%, leading one to wonder if its performance might continue to increase with even higher class distributions.

Figure 4.6 depicts the noise identification performance for each learner at each level of main factor NL. As before, RBF produces poor results at every level of this main factor, while NB and RF100 produce the best results. As was the case for the class distribution, for most of the levels of NL (10% through 30% for the majority class and 10% through 40% for the minority class), RF100 performed better than NB, while for the more extreme values (in this case more noise as opposed to a more imbalanced class distribution for the previous factor), NB was more robust and produced better results. The other results also follow generally from previous analysis, with 5NN, LR, and MLP performing better than the worst but worse than the best and SVM and C4.5N performing worse, overall, than all but RBF. One noteworthy observation, however, is the performance of SVM as the noise level increases. After NL=30%, the results for SVM drop quite significantly, indicating SVM's poor detection of mislabeled instances when they are present in abundance. Another aspect to note is the gentler slope between the 40% and 50% noise levels for the majority class (RBF's performance actually increased in this interval), attributable to the fact that ND=100% could not be performed at this noise level.

The interaction between Lrn and ND is portrayed in Figure 4.7. Again, RBF performs quite poorly across all levels, while RF100 and NB perform the best. These results again have NB performing more robustly, coming out on top with more extreme levels of the factor (ND=75% and ND = 100%), while RF100's results were superior otherwise. Similar to the situation with main factor NL, at the highest level of ND, the majority class AUC values either stabilize or actually increase due to the fact that the 50% noise level could not be performed at this level of ND. Overall, the

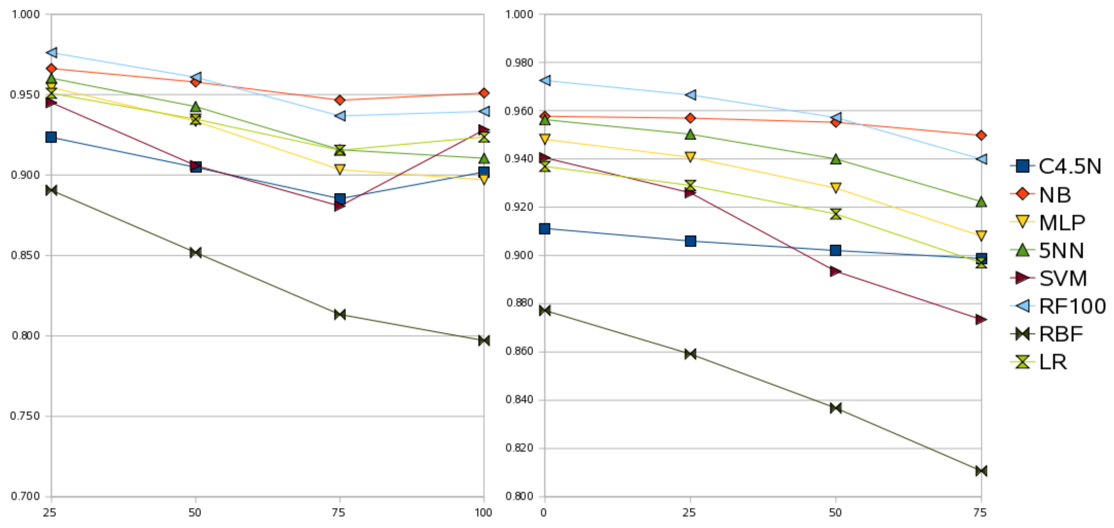

Figure 4.7: Noise identification AUC per learner at each level of ND for the majority class (left) and minority class (right)

AUC results based on the distribution of class noise tend to follow the same patterns identified in the analysis elsewhere in this section.

4.4 Conclusions

Data quality is of the utmost importance, often more so than the choice of learning algorithm, to the results of a classification initiative. Two data quality issues that often afflict many meaningful real world machine learning application areas are class noise and class imbalance. A common method for dealing with class noise is classification filtering, in which a classifier is learned and tested on the dataset and misclassified instances are considered noisy and removed or relabeled. In this paper we presented the results of an extensive set of empirical experiments in which we precisely processed several real world datasets to control their class distribution, noise level, and noise distribution. With this manipulated data we recorded the noise identification performance (accurately discriminating actually noisy instances from clean ones) of eight popular classification algorithms when used as classification filters, and analyzed the effects of differing levels of class imbalance and class noise. All results were found statistically significant (at the $\alpha = 0.05$ level) using ANOVA modeling. Our general conclusions are as follows.

1. Overall, the three experimental factors other than learner, i.e., class distribution, noise level, and noise distribution, affect noise detection results in a predictable way. As the level of class imbalance goes down, classifiers are less biased by the overpopulation of majority instances and produce more accurate classifiers, and thus more accurate classification filters. With increasing levels of class noise comes decreasing ability of the classifiers to efficiently detect the mislabeled instances due to the lower quality data they're trained on. In the combination of the two data quality factors (class imbalance and class noise), as the amount of the mislabeled instances coming from the more vulnerable minority class increases, the classifiers do a poorer job of identifying the noisy instances.

2. The RBF Network learner performs very poorly as a classification filter for imbalanced data. It performed significantly worse (statistically and quantitatively) than all seven other learners tested. This was not only the case on average for all the experiments, but when each level of each other experimental factor was isolated, RBF always produced the worst results.

3. SVM and C4.5N were better than RBF, but worse than the others. Of the two, SVM produced superior results more often, depending on the level of noise and class imbalance, however neither performed well. This is a somewhat dangerous result in that they are both popular classification algorithms, and thus an uninformed practitioner trying to perform classification filtering might be tempted to use them, however our experimental results show this to likely not be a profitable endeavor. One interesting observation, though, is how stable SVM's noise detection performance remained as the level of class imbalance varied.

4. The 5NN, logistic regression, and multilayer perceptron learners performed pretty well, though they were not the best. Of the three, 5NN won out overall, and performed best at most levels of the individual experimental factors. Though they occasionally swapped places at different levels of the main factors, LR was usually somewhat better at identifying noisy instances in the majority class (P$\rightarrow$N noise) than MLP, while MLP was usually superior at identifying noisy instances in the minority class (N$\rightarrow$P).

5. Naive Bayes and random forests were the two best performers. They both produced results that were clearly better than those of the others (both exceeding the other learners at all but one level of one factor) and similar to those of each other. According to statistical comparison, NB was significantly better at identifying P$\rightarrow$N noise and RF100 was significantly better at detecting N$\rightarrow$P noise,

however the actual numerical differences were quite minor. Overall, based on varying levels of the three main factors, NB tended to do slightly better at extreme levels. That is, when the class distribution was highly imbalanced, when there were high levels of noise, or when the level of noise from the minority class (P→N) was high, NB produced the better results, owing to its stable performance. Of all the learners, the results produced by NB varied the least, overall, as the levels of class distribution, noise level, and noise distribution changed. RF100, on the other hand, produced better results when the data was of slightly higher quality, outperforming NB when class distributions were more even, data was less noisy, and less of the noise was of the more dangerous P→N variety.

To conclude, based on the results of our empirical experimentation, we can confidently recommend the Naive Bayes and random forest classification algorithms for use in classifier based noise detection. Both clearly and consistently performed better than all the other learners we tested. Since they perform quite similarly to each other, Naive Bayes is probably most advantageous due to its much lower computational resource requirements compared to random forest. We can also confidently recommend against using the RBF Network learner, as it produced the worst results by a large margin.

CHAPTER 5

FILTERED UNDERSAMPLING

5.1 Introduction

Most classification algorithms are notorious for failing to identify rare cases. Classification performance on the minority class of a dataset with highly imbalanced classes is often quite poor. In addition to class imbalance, noise in the training data is another important data quality issue that has a negative impact on classification performance. Data sampling techniques have been popular for treating both the problems of class label noise and class imbalance. Class label noise is often treated using a classification filter, where the training data is used to build and evaluate a classifier, and training instances that are misclassified are either removed from the training dataset or relabeled. Data preprocessing techniques for dealing with class imbalance are based on undersampling majority class instances, oversampling minority class instances, or both. Both under and oversampling can be performed in a random or intelligent way. Intelligent undersampling techniques often try to remove majority class instances that are considered to be potentially noisy [6], and are thus are very similar to preprocessing techniques intended to deal with data noise.

Past work has shown that simple random undersampling performs better for class imbalance problems than more sophisticated undersampling techniques [78]. Though the intelligent undersampling techniques intended to remove majority class instances that are possibly noisy can work well with noisy data, they often don't remove enough instances to account for the class imbalance. Random undersampling, though easily able to balance a training dataset, indiscriminately eliminates majority class examples and may leave noisy examples which may mislead a classifier. To alleviate both these problems, we propose the *Filtered Undersampling* (FUS) technique, based on classification filtering, as a new data sampling technique to improve classification performance for imbalanced data, especially in the presence of class label noise. Intended to both eliminate noisy or confusing majority instances and sample the data significantly enough to alleviate class imbalance, we test the performance of FUS using 35 real world datasets, including a set of experiments utilizing precise class noise injection. We then compare the performance of this new technique against that of several of the most common data sampling techniques for class imbalance. All results are tested for statistical significance using ANOVA models and multiple pairwise comparison testing.

The rest of this chapter is organized as follows. Section 5.2 describes the Filtered Undersampling procedure and the methodology used to empirically evaluate its performance, including descriptions of the datasets, classification algorithms, comparison sampling techniques, and noise injection procedures utilized. Section 5.3 presents the results of our empirical experimentation on the performance of FUS. Section 5.4 then provides concluding thoughts and analysis.

5.2 Experiments

We evaluated the performance of the Filtered Undersampling technique using the WEKA machine learning tool [87], written in Java. Section 5.2.1 provides information about the datasets used for this empirical experimentation. Section 5.2.2 describes the classification algorithms used. The FUS procedure is described in Section 5.2.3. Section 5.2.4 describes the three common data sampling techniques we compare FUS against. Section 5.2.5 details the overall experimental design, including the data sampling and noise injection procedures.

5.2.1 Datasets

In total, 35 real world datasets were used for these experiments. All of them are imbalanced, ranging from 1.33% to 34.90% minority instances. Some are proprietary datasets, while the others were aquired from the UCI Machine Learning Repository [8]. Many of the datasets originally contained multiple class values, but were transformed into binary class problems by leaving one class as the minority class and combining the rest of the classes into one large majority class. The datasets selected for use in these experiments, as well as information about their size and class distributions, can be found in Table 5.1. In this study we performed two sets of experimentation. First we tested FUS (and compared it against several other common data sampling techniques) on the first 31 datasets from Table 5.1. In the second set of experiments, we evaluated FUS in the presence of controlled class imbalance and class noise using the last four datasets. We selected these four datasets due to their sizes, and more importantly the number of minority class examples. These characteristics facilitiated our experimental design, as decribed in Section 5.2.5. Most importantly, however, is that these datasets are relatively clean. That is, models constructed on these datasets (using ten runs of ten-fold cross validation) resulted in nearly perfect classification performance prior to noise injection.

5.2.2 Learners

All eight classifiers were used as implemented in the WEKA machine learning tool [87]. Unless otherwise noted, the default parameters for the classifier were used.

5.2.2.1 C4.5

C4.5 is the benchmark decision tree learning algorithm proposed by Quinlan [61]. C4.5 is among the most commonly used learning algorithms in data mining research. The decision tree is built using an entropy-based splitting criterion stemming from information theory [2]. C4.5 improves Quinlan's older ID3 [60] decision tree algorithm by adding support for tree pruning and dealing with missing values and numeric attributes. The WEKA version of C4.5 is called J48. Two different versions of the C4.5 classifier were used in this study, denoted C4.5D and C4.5N. C4.5D uses

Dataset	# Positive	# Negative	# Total	% Positive	% Negative
SP3	47	3494	3541	1.33%	98.67%
SP4	92	3886	3978	2.31%	97.69%
MAMMOGRAPHY	260	10923	11183	2.32%	97.68%
SOLAR-FLARE-F	51	1338	1389	3.67%	96.33%
CAR-3	69	1659	1728	3.99%	96.01%
SP2	189	3792	3981	4.75%	95.25%
CCCS-12	16	266	282	5.67%	94.33%
SP1	229	3420	3649	6.28%	93.72%
PC1	76	1031	1107	6.87%	93.13%
MW1	31	372	403	7.69%	92.31%
GLASS-3	17	197	214	7.94%	92.06%
KC3	43	415	458	9.39%	90.61%
CM1	48	457	505	9.50%	90.50%
CCCS-8	27	255	282	9.57%	90.43%
PENDIGITS-5	1055	9937	10992	9.60%	90.40%
SATIMAGE-4	626	5809	6435	9.73%	90.27%
ECOLI-4	35	301	336	10.42%	89.58%
SEGMENT-5	330	1980	2310	14.29%	85.71%
KC1	325	1782	2107	15.42%	84.58%
JM1	1687	7163	8850	19.06%	80.94%
LETTER-VOWEL	3878	16122	20000	19.39%	80.61%
CCCS-4	55	227	282	19.50%	80.50%
KC2	106	414	520	20.38%	79.62%
CONTRACEPTIVE-2	333	1140	1473	22.61%	77.39%
VEHICLE-1	212	634	846	25.06%	74.94%
HABERMAN	81	225	306	26.47%	73.53%
YEAST-2	429	1055	1484	28.91%	71.09%
PHONEME	1586	3818	5404	29.35%	70.65%
CCCS-2	83	199	282	29.43%	70.57%
CREDIT-GERMAN	300	700	1000	30.00%	70.00%
DIABETES	268	500	768	34.90%	65.10%
Datasets for noise injection					
NURSERY-3	328	12632	12960	2.53%	97.47%
LETTER-A	789	19211	20000	3.95%	96.06%
OPTDIGITS-8	554	5066	5620	9.86%	90.14%
SPLICE-2	768	2422	3190	24.08%	75.92%

Table 5.1: Characteristics of experimental datasets

the default parameter settings as in WEKA, while C4.5N uses no pruning and Laplace smoothing [85].

5.2.2.2 Naive Bayes

Naive Bayes [54] (NB) is a quick and simple classifier that utilizes Bayes's rule of conditional probability. It is 'naive' in that it assumes that all predictor variables are independent. Although this assumption rarely holds true in real-world data, Naive Bayes has been shown to often perform well, even in the presence of strong attribute dependencies [19]. We use the default parameters for Naive Bayes in our experiments.

5.2.2.3 Multilayer Perceptron

Multilayer perceptrons [62] (MLP) attempt to artificially mimic the functioning of a biological nervous system. Multiple nodes, or 'neurons,' are connected in layers, with the output of each node being the thresholded weighted sum of its inputs from the previous layer. The network weights are usually learned using a gradient descent algorithm called back-propagation. It has been shown that a multiple hidden layer neural network can approximate any function [38]. We change two parameters in our experiments. The 'hiddenLayers' parameter was changed to 3 to define a network with one hidden layer containing three nodes, and the 'validationSetSize' parameter was changed to 10 so that 10% of the data would be used as a validation set to determine when to stop the iterative training process.

5.2.2.4 RIPPER

RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [16] is a rule-based learner that modifies the IREP algorithm [27] to improve accuracy without sacrificing efficiency. RIPPER has been shown to produce classifier accuracy that compares favorably with both C4.5 and C4.5rules [61], while maintaining an advantage in efficiency for large and noisy datasets. JRip is the WEKA implementation of RIPPER. The default JRip parameter values were used in all experiments.

5.2.2.5 K Nearest Neighbors

K nearest neighbors [3], abbreviated kNN, is a 'lazy learning' technique. The training dataset serves as a *case library* which is used to predict the class of an instance based on it's k 'nearest neighbors' in the case library. The nearest neighbors are those instances in the case library that are closest in the feature-space to the instance being classified based on some distance measure, usually Euclidean distance. Two versions of the kNN classifier were used in our experiments, denoted 2NN and 5NN. 2NN uses two nearest neighbors for classification and 5NN uses five nearest neighbors. Also, the 'distanceWeighting' parameter was set to 'Weight by 1/distance.'

5.2.2.6 Support Vector Machines

A *support vector machine* (SVM) [65] is a linear classifier which attempts to learn the maximum margin hyperplane separating two classes in the data. This maximum margin hyperplane is usually determined by a small subset of instances called *support vectors.* In order to learn nonlinear decision boundaries, the data can be transformed by a kernel function. The linear maximum margin hyperplane then found for the transformed data can represent a nonlinear partitioning of the original feature space. SMO is the implementation of support vector machines in WEKA. Two changes to the default parameters were made: the complexity constant ‘c’ was changed from 1.0 to 5.0, and the ‘buildLogisticModels’ parameter, which allows proper probability estimates to be obtained, was set to true. By default, SMO uses a linear kernel.

5.2.2.7 Random Forests

Random forests [10] (RF100) give accuracy that compares favorably with Adaboost [26] and is relatively robust to noise. This learner utilizes bagging [9] and the ‘random subspace method’ [33] to construct an ensemble (forest) of randomized, unpruned decision trees. The outputs of this forest of decision trees are combined to produce the ultimate prediction. The ‘numTrees’ parameter was changed to 100 to specify using a forest of 100 trees.

5.2.2.8 Radial Basis Function Network

A *radial basis function network* [56] (RBF), like MLP, is a type of artificial neural network. The radial basis function outputs a value based on the radial distance of a feature vector from a ‘center’ in the feature space. RBFs typically have two processing layers. The data is first input to each RBF in the first layer. The second layer then linearly combines the output of the radial basis functions to produce the final output. The only parameter change for RBF (called RBFNetwork in WEKA) was to set the parameter 'numClusters' to 10.

5.2.2.9 Logistic Regression

Logistic regression (LR) is a a type of generalized linear model which can be used to predict a binary dependent variable [31]. While LR does allow categorical or continuous independent variables, categorical variables must be encoded (e.g., 0,1) to facilitate classification modeling. No changes to the default parameter values were made.

5.2.3 Filtered Undersampling Procedure

The Filtered Undersampling procedure we utilize in these experiments is, for the most part, a version of the standard classification filter. Classification filters build and evaluate a chosen classifier on the training data and then any instances that are misclassified are considered to be noisy, and subsequently are removed from the data or have their class label changed. In our application, instead of judging instances to be noisy based on a set decision threshold, we just remove the majority instances in order of their apparent level of 'noisiness' (with noisiness being defined by the difficulty for the classifier to properly identify the instance) until the desired class distribution is reached. This is intended to remove the noisy and/or confusing majority instances as well as balance the class distribution in a more extreme way than is typically achieved by using a standard classification filter such as in Wilson's Editing [6].

We apply the classification filtering procedure described in Section 3.3 and in Figure 3.1 with the following settings. We apply 10-fold cross-validation to the training data using one of the classifiers listed in Section 5.2.2 as the FilterLearner. For the experiments with the 31 unaltered datasets, we used eight total classifiers as FilterLearners for FUS: C4.5N, NB, MLP, 5NN, SVM, RF100, LR, and Ens (an ensemble combining the results of C4.5N, NB, 5NN, and LR). For the noise injection experiments, due to the increased computational complexity of having many more combinations of experimental parameters, we limited this set to C4.5N, NB, 5NN, RF100, LR, and Ens. We use the Maj option so that only majority class instances are identified as noisy, and thus, in each fold of the cross-validation, we include all of the minority instances in the training set. As it is an undersampling technique, the Treatment=Remv option is used, rather than relabeling the instances. The choice of the Threshold option for the classification filter is determined dynamically at runtime. The majority instances with the highest probabilities of membership in the minority class are removed until the desired class distribution is reached. In this work we tested two different sampling percentages (one of the main experimental factors), 35% and 50%, where that percentage represents the proportion of the total instances represented by the minority class after sampling has been performed.

5.2.4 Sampling Techniques

In addition to testing various parameters for FUS, we compare it against three common data sampling techniques utilized for class imbalance: random undersampling, random oversampling, and SMOTE. Each of these three techniques requires a parameter to indicate how much sampling to perform. We use 35% (indicating the post-sampling minority class percentage) in all three cases, as previous work [48] has shown this distribution to generally work well for imbalanced data. All three of these techniques have been shown to be effective at improving classification performance in previous research [78]. In addition, all experiments are performed without sampling (denoted NONE) to serve as a baseline for comparison. Each of the data sampling techniques are described in this section.

5.2.4.1 Random Resampling

The two most common (largely due to their simplicity) data sampling techniques used in this work are *random oversampling* (ROS) and *random undersampling* (RUS). Random oversampling duplicates instances (selected randomly) of the minority class. While this does help to balance the class distribution, no new information is added to the dataset and this may lead to overfitting [20]. Also, the size of the training datasets is increased, which causes longer model training times. Random undersampling randomly discards instances from the majority class. In doing so, the class distribution can be balanced, but important information can be lost when examples are discarded at random.

5.2.4.2 Synthetic Minority Oversampling Technique (SMOTE)

Chawla et al. [14] proposed an intelligent oversampling method called Synthetic Minority Oversampling Technique (SMOTE). SMOTE (denoted SM in this work) adds new, artificial minority examples by extrapolating between preexisting minority instances rather than simply duplicating original examples. The newly created instances cause the minority regions of the feature-space to be fuller and more general. The technique first finds the k nearest neighbors of the minority class for each minority example (the paper recommends $k = 5$). The artificial examples are then generated in the direction of some or all of the nearest neighbors, depending on the amount of oversampling desired. If 200% oversampling is specified, then synthetic examples are generated randomly along the line segments connecting each minority example to two of its five nearest neighbors. If x_i is the minority example being examined, x_j is one of the selected nearest neighbors of x_i, and x_n is the new example being added to the dataset, then $x_n = u(x_j - x_i) + x_i$, where u is a uniform random number between zero and one. This sampling process causes a classifier to learn a larger and more general decision region in the feature-space, ideally alleviating the problem caused by the class imbalance.

5.2.5 Experimental Design

Two separate sets of experiments were performed to evaluate the Filtered Undersampling technique. In the first set of experiments, the FUS preprocessing technique was applied using eight different FilterLearners, in conjunction with 11 classifiers on 31 unaltered real world datasets. 10-fold cross-validation was used, so for each fold, $\frac{9}{10}$ of the data was used as training data and the final $\frac{1}{10}$ of the data was used as the test data. Each time, FUS (or another sampling technique in the case of the others we compared it to) is applied to the training part, the classifier is built on this sampled data, and then the classifier is evaluated on the test fold. To ensure the statistical validity of the results, 10 runs of each 10-fold cross-validation process were performed.

The second set of experiments is performed using subsets of the four datasets described in Section 5.2.1 with controlled class distributions, noise levels, and noise distributions. For each combination of the different levels of these parameters, 100 independent trials (10 runs of 10-fold cross-validation) are performed, so that the set of instances which are included in the subset and the set of those instances which have their class labels switched (from the minority class to the majority class or vice versa) is different each time. Each subset of the original training data is sampled to contain 1500 examples and a class distribution according to our experimental parameters. These subsets are then corrupted to include the desired level and distribution of class noise. It is then, once this final, processed subset of data is constructed, that the FUS procedure (or other sampling technique) is applied. The three experimental factors pertinent to this second set of experiments, class distribution, noise level, and noise distribution, are described in this section.

5.2.5.1 Experimental Factor: Class Distribution

The first experimental factor, class distribution (CD), indicates the percentage of examples in the derived dataset belonging to the minority class. The experiments in this work consider six levels of class distribution, $CD = \{1, 2, 4, 6, 8, 10\}$, where the value of CD indicates the percentage of examples in the training data that belong to the minority class. For example, a derived dataset with $CD = 4$ has 4% of its examples from the minority class and 96% of its examples from the majority class. Note that this is the ratio prior to noise injection (discussed in the following sections). Depending on the noise distribution, the final dataset may contain a different class distribution.

5.2.5.2 Experimental Factor: Noise Level

The second factor, noise level (NL), determines the quantity of noisy examples in the training data. The selected datasets are relatively clean, so NL is varied by artificially injecting noise into the dataset. This is accomplished by swapping the class value of some of the examples. The number of examples with their classes swapped is a function of NL and the number of minority examples in the derived dataset.

While many works involving noise injection often inject noise by simply selecting $x\%$ of the examples and corrupting their class, this technique may be inappropriate when dealing with imbalanced datasets. For example, if a dataset contains only 1% of its examples belonging to the minority class, and as little as 10% of its examples are corrupted (injected with noise), the minority class will become overwhelmed by noisy examples from the majority class. Instead, we corrupt a percentage of the examples based on the number of minority examples in the dataset.

In our experiments, we use five levels of noise, $NL = \{10, 20, 30, 40, 50\}$, where NL determines the number of examples, based on the size of the minority class, that will be injected with noise. The actual number of noisy examples will be:

$$2 \times \frac{NL}{100} \times P$$

where P is the number of positive (minority) examples in the dataset. For example, a dataset with $CD = 10$ and $NL = 20$ will have 150 minority examples (10% of 1500) and $2 \times 0.2 \times 150 = 60$ noisy examples. Note that this does not indicate which examples (minority or majority) will be corrupted. That is determined by the final experimental factor: noise distribution.

5.2.5.3 Experimental Factor: Noise Distribution

The final experimental factor, noise distribution (ND), determines the type of noise that will be injected into the data. When dealing with binary class datasets (the only kind considered in this work), there are two possible noise types: P→N and

N$\rightarrow$P. Noise of type N$\rightarrow$P is when an instance that should be labeled "negative" is incorrectly labeled as "positive." Conversely, P$\rightarrow$N noise is when an example that should be labeled "positive" is instead labeled "negative."

The experiments in this work use five levels of noise distribution, $ND = \{0, 25, 50, 75, 100\}$, where the value of ND indicates the percentage of noisy examples that are of the P$\rightarrow$N variety. For example, if a derived dataset is to contain 60 noisy examples, and $ND = 25$, then 25% (15) of those noisy examples will be minority ("positive") class examples that have their labels changed to "negative" (P$\rightarrow$N), while the remaining 75% (45) of the noisy examples will be of the N$\rightarrow$P variety. Due to the definition of ND, the combination of $ND = 100$ and $NL = 50$ can not be used, since the resulting dataset would have zero minority class examples.

5.3 Results

Here we present the results of the extensive empirical experiments performed to evaluate the performance of the Filtered Undersampling technique. As explained in Section 5.2, two sets of experiments were performed. The first set evaluated FUS using 8 different FilterLearners (as well as three other common sampling techniques and no sampling at all) on 31 imbalanced real world datasets. The second set of experiments used four other real world datasets, this time controlling the dataset size, class distribution, noise level, and noise distribution. For this second set of experiments, FUS was tested using six different FilterLearners and again compared to three other sampling techniques, as well as the baseline of not performing any data preprocessing.

5.3.1 Filtered Undersampling Performance

5.3.1.1 Regular Data

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	10	3532.80	<.0001	5429.77	<.0001
FilterLearner	7	2306.80	<.0001	831.41	<.0001
Percentage	1	4070.99	<.0001	2454.14	<.0001
Learner*FilterLearner	70	45.67	<.0001	17.27	<.0001
Learner*Percentage	10	60.99	<.0001	23.68	<.0001
FilterLearner*Percentage	7	58.86	<.0001	7.42	<.0001
Learner*FilterLearner*Percentage	70	3.91	<.0001	0.98	0.5280

Table 5.2: Regular data: ANOVA models showing the significance of each experimental factor based on both the AROC and APRC values.

Table 5.2 shows the results of the ANOVA model for the performance of FUS using the 31 unaltered datasets. For the AROC response variable, all three main factors (Learner, FilterLearner, and Percentage), as well as their two and three way interactions are all statistically significant at the $\alpha = 0.05$ level. For the APRC response variable, all main factors and two way interactions are significant, however the three way interaction term is not.

Since each of the main factors are shown to be statistically significant, we perform Tukey's Honestly Significant Difference (HSD) test, indicating which levels of these factors result in significantly different performances when compared to the other levels of that factor. For example, the first part of Table 5.3 provides the

AROC			APRC		
Factor	Mean	HSD	Factor	Mean	HSD
Learner					
RF100	0.821	A	MLP	0.502	A
MLP	0.819	A	5NN	0.472	B
SVM	0.803	B	2NN	0.443	C
5NN	0.786	C	SVM	0.441	C
NB	0.783	D	RF100	0.438	C
LR	0.782	D	LR	0.393	D
2NN	0.758	E	NB	0.385	E
C4.5N	0.755	F	RBF	0.348	F
RBF	0.754	F	C4.5N	0.309	G
RIPPER	0.729	G	RIPPER	0.279	H
C4.5D	0.723	H	C4.5D	0.277	H
FilterLearner					
RF100	0.811	A	RF100	0.435	A
MLP	0.793	B	MLP	0.409	B
C4.5N	0.792	B	C4.5N	0.401	C
5NN	0.771	C	SVM	0.389	D
Ens	0.766	D	5NN	0.388	D
SVM	0.759	E	LR	0.381	E
LR	0.759	E	Ens	0.366	F
NB	0.741	F	NB	0.351	G
Percentage					
35	0.785	A	35	0.405	A
50	0.763	B	50	0.374	B

Table 5.3: Regular data: mean values and HSD groupings for each experimental factor for both AROC and APRC.

results of the HSD test for the main factor Learner. This factor has 11 levels (the 11 different learners used in our experiments for classification), each of which is assigned to a group (indicated by a letter) based on its average performance (across all of the other factors, as well as 100 independent runs of each combination of main factors). Learners in group A performed better than those in group B, which performed better than those in group C, and so forth. If multiple levels (Learners) are assigned to the same group (i.e., they have the same letter in a column in this table) then their average performances were not significantly different based on the performance metric indicated in that column.

As the intent of this work is to describe FUS, discussion of unrelated single factors (e.g. Learner in this case), will be skipped. Table 5.3 presents the performance of the different levels of the two factors related to FUS, namely FilterLearner and Percentage. For the main factor FilterLearner, both response variables show relatively similar results. RF100 performs significantly better than all seven other classifiers tested as a FilterLearner. MLP and C4.5N follow in second and third place, while Naive Bayes is shown to be outperformed by all other FilterLearners by a statistically significant margin. The two levels of the Percentage main factor perform signficantly differently, with 35% outperforming 50% in both AROC and APRC. Table 5.4 shows the mean AROC and APRC values for the interaction of the main factors Percentage and FilterLearner. For both response variables, 35% results in better performance than 50% for every FilterLearner. In each case, the difference is statistically significant at the $\alpha = 0.05$ level, indicating the clear superiority of 35% as the sampling percentage.

AROC								
Percentage	C4.5N	NB	MLP	5NN	SVM	RF100	LR	Ens
35	0.799	0.754	0.804	0.780	0.772	0.815	0.774	0.779
50	0.785	0.728	0.781	0.761	0.746	0.807	0.744	0.753
APRC								
Percentage	C4.5N	NB	MLP	5NN	SVM	RF100	LR	Ens
35	0.414	0.367	0.427	0.402	0.405	0.446	0.399	0.383
50	0.387	0.334	0.391	0.374	0.372	0.423	0.362	0.349

Table 5.4: Regular data: mean AROC and APRC values produced by each sampling percentage for each FilterLearner. All differences are statistically significant.

AROC											
FilterLearner	2NN	5NN	C4.5D	C4.5N	LR	MLP	NB	RBF	RF100	RIPPER	SVM
C4.5N	A	A	C	C	B	BC	AB	B	B	C	B
NB	F	F	G	G	D	F	E	F	D	G	D
MLP	B	B	B	B	B	B	B	C	B	B	B
5NN	C	C	D	D	C	CD	E	D	C	D	C
SVM	DE	E	EF	F	C	EF	D	D	D	EF	C
RF100	A	A	A	A	A	A	A	A	A	A	A
LR	EF	E	F	F	C	EF	B	E	D	F	C
Ens	D	D	E	E	C	DE	C	DE	C	E	C
APRC											
FilterLearner	2NN	5NN	C4.5D	C4.5N	LR	MLP	NB	RBF	RF100	RIPPER	SVM
C4.5N	A	A	D	CD	BC	C	B	B	B	D	C
NB	C	D	F	E	E	D	D	F	E	F	E
MLP	B	B	B	B	B	B	B	B	B	B	B
5NN	B	B	C	B	D	C	C	C	B	C	D
SVM	C	C	CD	C	C	BC	B	C	C	D	C
RF100	A	A	A	A	A	A	A	A	A	A	A
LR	C	CD	E	D	D	C	A	D	C	E	C
Ens	C	CD	E	E	D	C	C	E	D	EF	D

Table 5.5: Regular data: HSD groupings showing the relative performance of each level of FilterLearner for each of the learners.

Table 5.5 displays the HSD groupings of the FilterLearners for each level of Learner. Looking down the columns, the HSD groupings for the eight FilterLearners for a single Learner are shown. Looking across the rows, one sees the different HSD groupings that FilterLearner achieved for all 11 different levels of Learner. As was the case when comparing the FilterLearners overall, when looking at each Learner individually, RF100 is always in group A for both AROC and APRC. MLP was also consistent and successful, falling into group B for every Learner based on APRC, and all but one according to AROC. The bottom performer is also the same for the overall results and those for each individual Learner. NB was in the lowest group for every Learner according to both response variables, though sometimes there were other FilterLearners that did not perform significantly better than NB.

5.3.1.2 Data With Injected Noise

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	10	558802	<.0001	573137	<.0001
Percentage	1	155965	<.0001	95117	<.0001
FilterLearner	5	76023.7	<.0001	34938.3	<.0001
ClassDistribution	5	232686	<.0001	112342	<.0001
NoiseLevel	4	158966	<.0001	70974.7	<.0001
NoiseDistribution	4	40285.3	<.0001	20720.2	<.0001
Learner*Percentage	10	4619.38	<.0001	273.38	<.0001
Learner*FilterLearner	50	5525.64	<.0001	3780.4	<.0001
Percentage*FilterLearner	5	805.44	<.0001	354.55	<.0001
Learner*ClassDistribution	50	3846.93	<.0001	3026.23	<.0001
Percentage*ClassDistribution	5	4264.75	<.0001	4339.67	<.0001
FilterLearner*ClassDistribution	25	4246.25	<.0001	1101.69	<.0001
Learner*NoiseLevel	40	567	<.0001	3258.55	<.0001
Percentage*NoiseLevel	4	1270.5	<.0001	271.52	<.0001
FilterLearner*NoiseLevel	20	680.92	<.0001	528.85	<.0001
ClassDistribution*NoiseLevel	20	4695.58	<.0001	1224.3	<.0001
Learner*NoiseDistribution	40	2541.72	<.0001	1759.7	<.0001
Percentage*NoiseDistribution	4	3798.92	<.0001	2773.35	<.0001
FilterLearner*NoiseDistribution	20	2136.11	<.0001	954.25	<.0001
ClassDistribution*NoiseDistribution	20	4893.2	<.0001	3049.81	<.0001
NoiseLevel*NoiseDistribution	15	12865.6	<.0001	4957.45	<.0001

Table 5.6: Noise injection: ANOVA models showing the significance of each experimental factor based on both the AROC and APRC values.

After testing FUS on 31 unaltered real world datasets of varying sizes and levels of class imbalance, we performed some more controlled experiments using four other real world datasets. Here we control the dataset size, class distribution, noise level, and noise distribution as described in Section 5.2.5. The three previous main factors (Learner, Percentage, and FilterLearner) are now joined by three new ones, ClassDistribution (CD), NoiseLevel (NL), and NoiseDistribution (ND). The ANOVA results for these six main factors and all of their two way interactions are given in Table 5.6. The table shows that all main factors, as well as their two way interactions are statistically significant at the $\alpha = 0.05$ level for both response variables.

Table 5.7 displays the mean response variable values and the associated HSD groupings for each level of all six main factors. Again, discussion of the performance of each Learner is outside the scope of this paper. The three new factors (CD, NL, and ND) by themselves also warrant little discussion here. The results for CD and NL are intuitive. The level of class imbalance or class label noise is inversely proportional to the performance. For the most part, the proportion of noisy instances that originally came from the minority class ($P \rightarrow N$ noise) is also inversely proportional to the performance. The exception is that the largest ND value, 100%, actually performs the best overall. This has to do with the fact that the highest NoiseLevel, $NL = 50$, isn't possible for this level of ND.

As with the 31 unaltered datasets, a sampling percentage of 35% clearly results in better FUS performance than 50% according to both response variables. The relative performance of the FilterLearners did shuffle somewhat from the first set of experiments, however. RF100 still performs significantly better than all other FilterLearners, however the other levels of this factor rank differently. The second and third place performers in this case are LR and Ens, respectively, even though they performed rather poorly for the regular data. C4.5N, which ranked third for the previous 31 datasets, is now second to last when artificially injecting noise and controlling the class distribution. NB, while still in the bottom half of the rankings, now occupies the top spot in the bottom half, rather than ranking dead last as before.

AROC			APRC		
Factor	Mean	HSD	Factor	Mean	HSD
Learner					
5NN	0.939	A	5NN	0.698	A
RF100	0.930	B	RF100	0.669	B
2NN	0.918	C	2NN	0.589	C
MLP	0.896	D	NB	0.560	D
NB	0.893	E	MLP	0.521	E
SVM	0.861	F	SVM	0.424	F
LR	0.815	G	LR	0.327	G
RBF	0.812	H	RBF	0.309	H
C4.5N	0.810	I	C4.5N	0.284	I
RIPPER	0.745	J	RIPPER	0.208	J
C4.5D	0.740	K	C4.5D	0.204	K
FilterLearner					
RF100	0.886	A	RF100	0.495	A
LR	0.854	B	LR	0.447	B
Ens	0.853	C	Ens	0.430	C
NB	0.843	D	NB	0.430	C
C4.5N	0.835	E	C4.5N	0.411	D
5NN	0.835	F	5NN	0.402	E
Percentage					
35	0.862	A	35	0.458	A
50	0.840	B	50	0.414	B
ClassDistribution					
10	0.891	A	10	0.510	A
8	0.879	B	8	0.486	B
6	0.863	C	6	0.454	C
4	0.845	D	4	0.423	D
2	0.823	E	2	0.386	E
1	0.803	F	1	0.356	F
NoiseLevel					
10	0.880	A	10	0.486	A
20	0.866	B	20	0.461	B
30	0.853	C	30	0.439	C
40	0.833	D	40	0.407	D
50	0.814	E	50	0.374	E
NoiseDistribution					
100	0.874	A	100	0.477	A
0	0.858	B	0	0.451	B
25	0.851	C	25	0.436	C
50	0.842	D	50	0.418	D
75	0.833	E	75	0.405	E

Table 5.7: Noise injection: mean values and HSD groupings for each experimental factor for both AROC and APRC.

Learner												
	AROC						APRC					
Learner	C4.5N	NB	5NN	RF100	LR	Ens	C4.5N	NB	5NN	RF100	LR	Ens
2NN	E	D	F	A	C	B	C	F	B	A	E	D
5NN	E	D	F	A	B	C	E	F	B	A	D	C
C4.5D	F	D	B	A	C	E	F	D	C	A	B	E
C4.5N	E	F	D	A	C	B	D	D	B	A	C	E
LR	D	E	C	A	C	B	E	D	F	A	C	B
MLP	F	D	E	A	B	C	E	B	F	D	A	C
NB	E	D	F	A	B	C	E	C	F	A	B	D
RBF	B	F	E	A	D	C	B	F	E	A	D	C
RF100	E	D	F	A	B	C	F	D	E	A	B	C
RIPPER	F	D	C	A	B	E	F	C	D	A	B	E
SVM	F	D	E	A	C	B	E	C	F	A	B	D

Percentage												
	AROC						APRC					
Percentage	C4.5N	NB	5NN	RF100	LR	Ens	C4.5N	NB	5NN	RF100	LR	Ens
35	F	D	E	A	B	C	E	C	F	A	B	D
50	E	D	F	A	C	B	E	D	F	A	B	C

ClassDistribution												
	AROC						APRC					
Class Dist.	C4.5N	NB	5NN	RF100	LR	Ens	C4.5N	NB	5NN	RF100	LR	Ens
1	B	D	E	A	C	C	B	D	E	A	B	C
2	C	D	E	A	B	B	C	C	D	A	B	C
4	E	C	D	A	B	B	D	C	E	A	B	C
6	F	E	D	A	B	C	E	C	D	A	B	C
8	F	E	D	A	B	C	E	C	D	A	B	C
10	E	D	C	A	B	C	E	C	D	A	B	C

NoiseLevel												
	AROC						APRC					
Noise Level	C4.5N	NB	5NN	RF100	LR	Ens	C4.5N	NB	5NN	RF100	LR	Ens
10	E	C	D	A	B	C	E	C	E	A	B	D
20	F	D	E	A	B	C	E	C	F	A	B	D
30	D	C	E	A	B	B	E	D	F	A	B	C
40	E	D	F	A	C	B	E	D	F	A	B	C
50	D	C	D	A	B	B	E	D	F	A	B	C

NoiseDistribution												
	AROC						APRC					
Noise Dist.	C4.5N	NB	5NN	RF100	LR	Ens	C4.5N	NB	5NN	RF100	LR	Ens
0	E	D	D	A	B	C	F	D	E	A	B	C
25	F	D	E	A	B	C	F	D	E	A	B	C
50	F	D	E	A	B	C	E	C	D	A	B	C
75	E	D	F	A	B	C	E	C	F	A	B	D
100	B	D	E	A	C	C	B	C	E	A	C	D

Table 5.8: Noise injection: HSD groupings (layed out horizontally) showing the relative performance of each FilterLearner for each level of all the other main factors.

The HSD groupings of the different FilterLearners for each level of all five other main factors are presented in Table 5.8. Here, the groupings are layed out horizontally. Each column represents one FilterLearner, and each row is one of the levels of one of the other main factors. For example, the first row of results depicts the HSD groupings for the six FilterLearners for the 2NN Learner (using both AROC and APRC). Looking down a column shows how the ranking of a FilterLearner varies as the level of a main factor changes. For example, based on AROC, RF100 again is in group A for every level of Learner. This is also almost the case according to APRC, however, one Learner, MLP, had the RF100 FilterLearner in group D. As in the overall results, LR and Ens were mostly in the B and C groups, though this varies more for the APRC response variable. 5NN and C4.5N, on the other hand, tend to be in the lowest ranks for most of the Learners. This trend continues for all of the main factors. RF100 is in group A for every level of every other main factor for both response variables (except, as previously mentioned, for the MLP Learner using APRC). LR and Ens continue to mostly occupy the B and C groups, while 5NN and C4.5N are most often the lowest ranked.

5.3.2 Comparison of FUS and Other Sampling Techniques

In this section, we compare FUS to three common sampling techniques for class imbalance: random undersampling (RUS), random oversampling (ROS), and SMOTE (SM), as well as the baseline case of not performing any data preprocessing (labeled None). For the two sets of experiments, we compare FUS using the three best performing FilterLearners to these four other sampling schemes. As the 35% sampling percentage clearly outperformed 50% for FUS, we use only the 35% results. The same sampling percentage is used for the three other data sampling techniques, which has been found to work well for them in previous work [48]. Here, the data sampling technique (RUS, ROS, SMOTE, or None) and the three versions of FUS are all described by the main factor called Technique. Within Technique, the different versions of FUS are labeled by their FilterLearners. For the 31 regular datasets, RF100, MLP, and C4.5N performed the best and are included, while for the noise injection experiments, RF100, LR, and Ens performed best and are used for comparison.

5.3.2.1 Regular Data

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	10	1673.34	<.0001	1401.82	<.0001
Technique	6	133.63	<.0001	1284.16	<.0001
Learner*Technique	60	50.51	<.0001	23.34	<.0001

Table 5.9: Regular data: ANOVA models showing the significance of each experimental factor based on both the AROC and APRC values.

The ANOVA results comparing FUS and the other data sampling techniques are given in Table 5.9. Since only the 35% value for Percentage is used, there are only two main factors, Learner and Technique. The analysis of variance model shows that both factors and their interaction are statistically significant. The means for both response variables and their HSD groupings for both main factors are presented in

AROC			APRC		
Factor	Mean	HSD	Factor	Mean	HSD
Learner					
RF100	0.854	A	MLP	0.555	A
MLP	0.837	B	RF100	0.554	A
5NN	0.835	B	5NN	0.543	B
SVM	0.822	C	2NN	0.509	C
LR	0.819	C	SVM	0.508	C
2NN	0.814	D	LR	0.492	D
NB	0.812	DE	RBF	0.463	E
C4.5N	0.810	E	NB	0.457	E
RBF	0.793	F	C4.5N	0.440	F
C4.5D	0.749	G	C4.5D	0.380	G
RIPPER	0.744	H	RIPPER	0.373	H
Technique					
RUS	0.817	A	SM35	0.526	A
RF100	0.815	AB	ROS35	0.524	AB
SM	0.814	B	None	0.520	B
ROS	0.809	C	RUS35	0.499	C
MLP	0.804	D	RF100	0.446	D
None	0.799	E	MLP	0.427	E
C4.5N	0.799	E	C4.5N	0.414	F

Table 5.10: Regular data: mean values and HSD groupings for each experimental factor for both AROC and APRC.

Table 5.10. Based on AUC, RUS produces the highest average AROC, however FUS with RF100 is also in group A and is thus not significantly outperformed. RF100 is also in group B with SMOTE, so it doesn't perform significantly better than SMOTE, but RUS does. ROS follows those three in group C, followed by MLP in group D. Finally, None and C4.5N are at the bottom in group E. The results based on APRC are much less encouraging for FUS. The three FUS techniques occupy the bottom three spots for this response variable, all being significantly outperformed by each of the four other levels of Technique. SMOTE and ROS share the top spot in group A, with ROS also falling into group B with None.

The HSD grouping of each level of Technique for each of the Learners is depicted in Table 5.11. Each column shows how the different Techniques rank for that particular Learner. These results, again, are quite similar to the overall results. Based on AROC, FUS with MLP produces good results for the C4.5D and RIPPER Learners, but overall, it produces relatively unimpressive results. FUS with C4.5N is even worse, often ranked in one of the lowest groups. FUS using RF100 as the FilterLearner is often in group A, though it falls into groups B and C for some of the learners. Using the APRC response variable, the three levels of Technique using FUS perform quite poorly. All three perform significantly worse than all four other levels of Technique for every learner except SVM, where RF100 and None are both in group C.

AROC											
Technique	2NN	5NN	C4.5D	C4.5N	LR	MLP	NB	RBF	RF100	RIPPER	SVM
None	AB	A	C	BC	A	CD	A	BC	A	E	D
RUS	AB	A	A	BC	BC	A	A	AB	A	B	AB
ROS	AB	AB	C	AB	A	AB	A	C	A	D	A
SM	A	A	B	A	AB	ABC	A	BC	A	D	AB
C4.5N	B	BC	B	E	D	D	B	BC	B	C	C
MLP	C	C	A	D	D	BCD	B	BC	B	AB	C
RF100	AB	AB	A	C	C	A	B	A	B	A	B
APRC											
Technique	2NN	5NN	C4.5D	C4.5N	LR	MLP	NB	RBF	RF100	RIPPER	SVM
None	A	A	B	A	A	AB	A	A	A	B	BC
RUS	B	AB	C	B	B	B	A	B	A	B	AB
ROS	AB	B	B	A	A	A	A	A	A	A	A
SM	B	B	A	A	A	A	A	A	A	A	A
C4.5N	C	C	E	E	D	D	B	D	C	D	E
MLP	C	D	D	D	D	C	B	D	C	C	D
RF100	C	C	D	C	C	C	B	C	B	C	C

Table 5.11: Regular data: HSD groupings showing the relative performance of each level of Technique for each of the learners.

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	10	313046	<.0001	287270	<.0001
Technique	6	26233.1	<.0001	148465	<.0001
ClassDistribution	5	164620	<.0001	107721	<.0001
NoiseLevel	4	109906	<.0001	99937.6	<.0001
NoiseDistribution	4	45675.3	<.0001	31969.9	<.0001
Learner*Technique	60	8650.84	<.0001	6399.91	<.0001
Learner*ClassDistribution	50	3381.67	<.0001	1061.93	<.0001
Technique*ClassDistribution	30	495.79	<.0001	870.18	<.0001
Learner*NoiseLevel	40	773.53	<.0001	797.48	<.0001
Technique*NoiseLevel	24	649.39	<.0001	1729.95	<.0001
ClassDistribution*NoiseLevel	20	981.92	<.0001	219.66	<.0001
Learner*NoiseDistribution	40	1643.38	<.0001	504.64	<.0001
Technique*NoiseDistribution	24	2860.14	<.0001	1470.59	<.0001
ClassDistribution*NoiseDistribution	20	1133.69	<.0001	1295.53	<.0001
NoiseLevel*NoiseDistribution	15	11638.5	<.0001	7524.52	<.0001

Table 5.12: Noise injection: ANOVA models showing the significance of each experimental factor based on both the AROC and APRC values.

5.3.2.2 Data With Injected Noise

The ANOVA results for the noise injection experiments comparing FUS with other common sampling techniques are presented in Table 5.12. All of the main factors, as well as all of their two way interaction terms, were statistically significant at the $\alpha = 0.05$ level for both response variables. To determine which levels are significantly different from each other, the means and HSD groupings for each level of all five main factors are given in Table 5.13. Main factors ClassDistribution, NoiseLevel, and NoiseDistribution follow the now familiar trends. Levels with more class imbalance or more class noise cause reduced performance. Higher levels of noise of the $P \rightarrow N$ variety also harm performance, except for the case of $ND = 100\%$, which is helped by the fact that it doesn't include the highest level of noise ($NL = 50$). For

AROC			APRC		
Factor	Mean	HSD	Factor	Mean	HSD
Learner					
RF100	0.952	A	RF100	0.761	A
5NN	0.942	B	5NN	0.725	B
NB	0.935	C	NB	0.702	C
MLP	0.928	D	MLP	0.665	D
2NN	0.912	E	2NN	0.628	E
SVM	0.906	F	SVM	0.598	F
LR	0.881	G	LR	0.533	G
C4.5N	0.866	H	RBF	0.472	H
RBF	0.840	I	C4.5N	0.456	I
C4.5D	0.772	J	C4.5D	0.336	J
RIPPER	0.768	K	RIPPER	0.332	K
Technique					
RUS	0.899	A	None	0.656	A
RF100	0.896	B	SM35	0.649	B
SM	0.893	C	ROS35	0.631	C
ROS	0.884	D	RUS35	0.579	D
None	0.875	E	RF100	0.520	E
LR	0.865	F	LR	0.469	F
Ens	0.862	G	Ens	0.449	G
ClassDistribution					
10	0.917	A	10	0.636	A
8	0.910	B	8	0.620	B
6	0.899	C	6	0.596	C
4	0.883	D	4	0.566	D
2	0.855	E	2	0.510	E
1	0.827	F	1	0.460	F
NoiseLevel					
10	0.912	A	10	0.634	A
20	0.899	B	20	0.603	B
30	0.884	C	30	0.568	C
40	0.861	D	40	0.517	D
50	0.848	E	50	0.484	E
NoiseDistribution					
0	0.901	A	0	0.605	A
25	0.892	B	25	0.581	B
100	0.880	C	100	0.571	C
50	0.877	D	50	0.549	D
75	0.860	E	75	0.518	E

Table 5.13: Noise injection: mean values and HSD groupings for each experimental factor for both AROC and APRC.

AROC, the ranking of the techniques is similar to that produced with the 31 unaltered datasets. RUS performs the best, followed closely by FUS with RF100, which is followed closely by SMOTE. In this case, however, the differences are all statistically significant. The two versions of FUS using LR and Ens produce the lowest AROC, not even providing any improvement over doing nothing at all (None). According to the APRC performance measure, FUS again produced poor results. All three versions were significantly outperformed by every other level of main factor Technique. In fact, no data sampling technique was able to produce a higher average APRC value than not doing any preprocessing at all, as None is alone in group A.

Tables 5.14 and 5.15 display the HSD groupings of each level of Technique for every level of all the other main factors based on AROC and APRC, respectively. The groupings are arrayed horizontally, so, for example, the first row shows the groupings of the sampling techniques in each column for the 2NN learner only. The first part of the tables shows that there is a lot of variability in the relative performance of each level of Technique depending on what learner is being used. Based on AROC, every level of Technique other than Ens is in group A for at least one Learner. The technique with the lowest variability, SMOTE, ranges from group A (RF100 and SVM) to group D (2NN, 5NN, RBF, and RIPPER). The three FUS techniques rank well for the 2NN, 5NN, and RIPPER learners. For all other learners, FUS with LR and Ens perform relatively poorly. FUS with RF100 also does well with C4.5D and RBF. According to response variable APRC, FUS performs poorly with almost all

Learner	None	RF100	LR	Ens	RUS	ROS	SM
				Learner			
2NN	E	A	B	B	C	F	D
5NN	E	A	A	B	C	F	D
C4.5D	G	B	D	F	A	E	C
C4.5N	A	D	E	F	C	A	B
LR	A	D	G	F	E	C	B
MLP	A	E	F	G	D	B	C
NB	A	C	D	E	B	A	B
RBF	C	B	G	E	A	F	D
RF100	C	E	F	G	D	B	A
RIPPER	E	A	B	C	A	D	D
SVM	C	D	F	E	C	B	A
				ClassDistribution			
1	E	A	F	G	B	D	C
2	E	B	F	G	A	D	C
4	D	B	E	F	A	C	B
6	D	B	E	F	A	C	B
8	E	B	F	G	A	D	C
10	E	B	F	G	A	D	C
				NoiseLevel			
10	D	B	E	F	A	C	B
20	D	B	E	F	A	C	B
30	E	B	F	G	A	D	C
40	E	A	D	D	A	C	B
50	D	AB	E	F	A	C	B
				NoiseDistribution			
0	C	D	E	F	B	B	A
25	D	D	E	F	B	C	A
50	D	B	E	F	A	C	B
75	E	A	E	F	B	D	C
100	F	A	CD	D	B	E	C

Table 5.14: Noise injection: AROC HSD groupings (layed out horizontally) showing the relative performance of each Technique for each level of all the other main factors.

Learner	None	RF100	LR	Ens	RUS	ROS	SM
Learner							
2NN	B	B	E	D	A	D	C
5NN	A	B	D	D	C	F	E
C4.5D	C	E	F	G	D	B	A
C4.5N	AB	D	E	F	C	B	A
LR	A	E	F	F	D	C	B
MLP	A	E	D	E	C	B	B
NB	A	D	D	E	C	A	B
RBF	A	D	E	E	C	B	A
RF100	B	D	E	F	C	A	B
RIPPER	C	E	F	G	D	B	A
SVM	B	D	E	F	C	A	A
ClassDistribution							
1	B	E	F	G	D	C	A
2	B	E	F	G	D	C	A
4	A	D	E	F	C	B	A
6	A	E	F	G	D	C	B
8	A	E	F	G	D	C	B
10	A	E	F	G	D	C	B
NoiseLevel							
10	A	E	F	G	D	C	B
20	A	E	F	G	D	C	B
30	A	E	F	G	D	C	B
40	A	D	E	F	C	B	A
50	A	E	F	G	D	C	B
NoiseDistribution							
0	A	E	F	G	D	C	B
25	A	E	F	G	D	C	B
50	A	E	F	G	D	C	B
75	B	E	F	G	D	C	A
100	C	E	F	G	D	B	A

Table 5.15: Noise injection: APRC HSD groupings (layed out horizontally) showing the relative performance of each Technique for each level of all the other main factors.

learners. All three versions fall into group D or lower for every learner, except for the cases of 2NN and 5NN where FUS with RF100 is in group B. As was indicated by the overall performance displayed in Table 5.13, None, SMOTE, and ROS occupy most of the A and B groups for most of the learners when judging on APRC.

The performance ranking of the different Techniques are more stable for the levels of the other main factors. Based on AROC, None and FUS with LR and Ens are in group D or worse for every level of ClassDistribution. ROS and SM are above those three, with ROS joining groups C and D, and SMOTE in groups B and C. RUS and FUS with RF100 are the only techniques in group A for any class distribution. RUS is in group B for $CD = 1$ and group A for all the others, while FUS is in group A for $CD = 1$ and group B for the rest. As usual, FUS does not perform well as measured by the APRC metric, always falling into group D or worse for all levels of ClassDistribution. The best performers based on APRC are SMOTE in group A for the more imbalanced distributions, and group B for the others, and None with the inverse behavior of group B for the more imbalanced distributions and group A for the rest.

Partitioned by NoiseLevel and NoiseDistribution, similar trends are shown. FUS invariably performs poorly based on APRC with None and SMOTE producing the best results. Based on AUC, FUS with LR and Ens both perform relatively poorly also, though Fus using RF100 performs well most of the time. As with ClassDistribution, this is particularly true at more extreme values of these two factors (i.e. higher levels of noise and more of it being from the minority class). At $NL = \{10, 20, 30\}$, FUS with RF100 is in group B and in group A for the two levels with even more noise. FUS with RF100 is in group D for $ND = \{0, 25\}$, but improves to group B for $ND = 50$ and group A for $ND = \{75, 100\}$. Overall though, RUS tends to more consistently rank at the top, and SMOTE performs well also, though it fades as the level of $P \rightarrow N$ noise increases.

5.4 Conclusions

Most classification algorithms tend to have low accuracy in identifying instances of rare classes. Class label noise is another recurring data quality issue that negatively affects classification performance. Classification accuracy degradation is further magnified when both issues are present. For class imbalance, data sampling techniques tend to remove instances from the majority class or add instances to the minority class to bring their numbers closer to parity. Preprocessing techniques for addressing class noise typically involve removing training instances that are misclassified by some classification scheme. Some sampling techniques combine both strategies, attempting to remove potentially noisy instances only from the majority class so that the imbalance is reduced. With an appreciable level of class imbalance, however, the amount of majority instances identified as potentially noisy tends to be too small to significantly ameliorate the performance reduction caused by the class imbalance.

To address this shortcoming, in this chapter we described and tested a majority class undersampling technique, Filtered Undersampling (FUS), based on noise filtering, that is intended for situations of class imbalance. Rather than using a static strategy to remove the majority instances likely to be noisy, FUS removes majority instances in their order of apparent likelihood of noisiness until the desired class distribution is achieved. This allows noisy instances to be removed, while also balancing the classes enough to improve the classification performance in the presence of extreme class imbalance. We performed thorough empirical experimentation to test this technique using both 31 real world datasets as well as four more datasets with noise injection in which the class distribution, noise level, and noise distribution among classes was tightly controlled. The results of FUS were then compared to those from several other common data sampling techniques. The following points summarize our general conclusions based on the experimental results.

1. RF100 was shown definitively to be the best classification algorithm to use for performing the filtering in Filtered Undersampling. This holds true whether it was tested on 31 different unaltered real world datasets or the four datasets that were artificially corrupted with injected noise. RF100 produced results that were either significantly superior to or statistically indistinguishable from the results of all seven other classification algorithms used in conjunction with FUS in these experiments. This was consistent using both the AROC and APRC performance metrics and found to be statistically significant at the $\alpha = 0.05$ significance level,

and remained the case at every level of every other experimental factor utilized.

2. A 35% after-sampling class distribution performs much better than a perfectly balanced class distribution (50%). This was found to be true even more consistently than in the case of other, more common data sampling techniques for class imbalance [48]. 35% produced statistically significantly superior AROC and APRC results overall, and for each classifier used for filtering with FUS.

3. Despite the success of ensemble classification techniques and the frequent use of classifier ensembles to perform classification filtering [49], there was little or no benefit in these experiments to using a classifier ensemble to perform the filtering for FUS. The ensemble of C4.5N, NB, 5NN, and LR didn't produce an overall improvement in performance (based on average AROC or APRC), and rarely wasn't outperformed by at least one of the single classifiers composing it.

4. Based on the APRC performance measure, FUS was significantly outperformed by all of the other data sampling techniques tested. Even the best version of FUS (that using RF100 as the FilterLearner) was unable to produce even comparable performance to any of the simpler sampling techniques or to the performance achieved using the unaltered training data. SMOTE, ROS, and None tended to produce the best APRC scores in all the various experimental scenarios.

5. Based on AROC, FUS was still largely outperformed. Other than when RF100 was used as the FilterLearner, the other sampling techniques produced superior results. FUS with RF100, however, was one of the best performing sampling techniques. For most levels of the other experimental factors, FUS with RF100, along with RUS, was one of the top two techniques. Further, FUS with RF100 seemed to rank even better as the data quality decreased. It tended to rank highest when the class distribution was most imbalanced, when the level of noise was most extreme, or when the amount of the noise inserted into the more sensitive minority class was highest.

Though it's rather sensitive to the choice of FilterLearner, the Filtered Undersampling technique used with the RF100 classifier can be quite successful as measured by the area under the ROC curve. It performed roughly on par with the best performing popular data sampling technique, random undersampling, and better than several others. However, taking into account the greatly increased complexity of implementation and computation compared to simple random undersampling and lack of a clear performance improvement, its use can not be generally recommended. It did, however, seem to improve in relative performance as the severity of class noise and class imbalance increased, so its use might be beneficial in situations where there are extreme data quality issues.

CHAPTER 6

BOOSTING TECHNIQUES FOR NOISY AND IMBALANCED DATA

6.1 Introduction

In real world machine learning applications, it is rare to find a situation where a target concept is fully and unerringly represented by the example data points used for induction. Data quality is a pervasive issue in machine learning, and walking the line between fitting the data accurately and precisely, and avoiding overfitting to generalize successfully and be robust to noisy data, is a difficult task. Data preprocessing techniques have probably been the most popular methods for addressing data quality issues. For class imbalance, schemes to oversampling the minority class [14] or undersample the majority class [50] have been popular. For alleviating class noise, classification filtering has been a popular method, whereby a learner is trained and evaluated on the training data (typically using cross-validation) and misclassified instances are considered noisy and either removed from the final training set [49] or relabeled [74]. For class imbalance, cost sensitive learning is another common solution.

Another popular method for addressing both class imbalance and class noise is to use ensemble techniques. For class imbalance, variants of Bagging [9] that adjust the class distribution and modified boosting algorithms that adjust the weight update formulas [43] or sample the training datasets at each iteration [72] can successfully improve performance. Similarly, several boosting methods have been developed for being robust to class noise, either by ignoring potentially noisy instances [45] or by preventing extreme increases in instance weights [58]. To our knowledge, however, these successful ensemble techniques have thus far focused solely on one issue or the other (class imbalance or class noise). To fill that void, in this work we test six new hybrid boosting techniques which combine solution concepts for both class imbalance and class noise on data that is both imbalanced and noisy. We perform a comprehensive battery of empirical experiments in which dataset size, class distribution, noise level, and noise distribution (among classes) is tightly controlled in order to assess the performance of these six new boosting algorithms in conjunction with four different base classifiers. We also test the boosting methods on 15 real world datasets that haven't been altered to assess their general performance without artificially injected class noise. These results are also compared to two boosting techniques each aimed at the issues of class imbalance and class noise, and all results are tested for statistical significance using analysis of variance models.

The rest of the Chapter is organized as follows. Descriptions of our experimental methodology, including details about the datasets, learning algorithms, boosting algorithms, and noise injection procedures are given in Section 6.2. Section 6.3 presents the findings of our exhaustive empirical experimentation. Lastly, Section 6.4 offers a summary of the findings.

6.2 Experiments

We evaluated the performance of 10 different boosting techniques using the WEKA machine learning tool [87], written in Java. Information about the datasets used for experimentation is given in Section 6.2.1. Section 6.2.2 describes the classification algorithms used. Section 6.2.3 provides details about the 10 boosting algorithms tested. Finally, Section 6.2.4 describes the overall experimental design, including the data sampling and noise injection procedures.

6.2.1 Datasets

In total, 19 real world datasets were used for these experiments. All of them are imbalanced, ranging from 1.33% to 30.00% minority instances. Some are proprietary datasets, while the others were aquired from the UCI Machine Learning Repository [8]. Many of the datasets originally contained multiple class values, but were transformed into binary class problems by leaving one class as the minority class and combining the rest of the classes into one large majority class. The datasets selected for use in these experiments, as well as information about their size and class

Dataset	# Positive	# Negative	# Total	% Positive	% Negative
SP3	47	3494	3541	1.33%	98.67%
MAMMOGRAPHY	260	10923	11183	2.32%	97.68%
SOLAR-FLARE-F	51	1338	1389	3.67%	96.33%
SP1	229	3420	3649	6.28%	93.72%
PC1	76	1031	1107	6.87%	93.13%
MW1	31	372	403	7.69%	92.31%
GLASS-3	17	197	214	7.94%	92.06%
CM1	48	457	505	9.50%	90.50%
PENDIGITS-5	1055	9937	10992	9.60%	90.40%
SATIMAGE-4	626	5809	6435	9.73%	90.27%
JM1	1687	7163	8850	19.06%	80.94%
CONTRACEPTIVE-2	333	1140	1473	22.61%	77.39%
VEHICLE-1	212	634	846	25.06%	74.94%
HABERMAN	81	225	306	26.47%	73.53%
CREDIT-GERMAN	300	700	1000	30.00%	70.00%
Datasets for noise injection					
NURSERY-3	328	12632	12960	2.53%	97.47%
LETTER-A	789	19211	20000	3.95%	96.06%
OPTDIGITS-8	554	5066	5620	9.86%	90.14%
SPLICE-2	768	2422	3190	24.08%	75.92%

Table 6.1: Characteristics of experimental datasets

distributions, can be found in Table 6.1. In this study we performed two sets of experimentation. First we tested the 10 boosting algorithms on the first 15 datasets from Table 6.1. In the second set of experiments, we evaluated these boosting techniques in the presence of controlled class imbalance and class noise using the last four datasets. We selected these four datasets due to their sizes, and more importantly the number of minority class examples. These characteristics facilitiated our experimental design, as decribed in Section 6.2.4. Most importantly, however, is that these datasets are relatively clean. That is, models constructed on these datasets (using ten runs of ten-fold cross validation) resulted in nearly perfect classification performance prior to noise injection.

6.2.2 Learners

All four classifiers were used as implemented in the WEKA machine learning tool [87]. Unless otherwise noted, the default parameters for the classifier were used.

6.2.2.1 C4.5

C4.5 is the benchmark decision tree learning algorithm proposed by Quinlan [61]. C4.5 is among the most commonly used learning algorithms in data mining research. The decision tree is built using an entropy-based splitting criterion stemming from information theory [2]. C4.5 improves Quinlan's older ID3 [60] decision tree algorithm by adding support for tree pruning and dealing with missing values and numeric attributes. The WEKA version of C4.5 is called J48. Two different versions of the C4.5 classifier were used in this study, denoted C4.5D and C4.5N. C4.5D uses the default parameter settings as in WEKA, while C4.5N uses no pruning and Laplace smoothing [85].

6.2.2.2 Naive Bayes

Naive Bayes [54] (NB) is a quick and simple classifier that utilizes Bayes's rule of conditional probability. It is 'naive' in that it assumes that all predictor variables are independent. Although this assumption rarely holds true in real-world data, Naive Bayes has been shown to often perform well, even in the presence of strong attribute dependencies [19]. We use the default parameters for Naive Bayes in our experiments.

6.2.2.3 RIPPER

RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [16] is a rule-based learner that modifies the IREP algorithm [27] to improve accuracy without sacrificing efficiency. RIPPER has been shown to produce classifier accuracy that compares favorably with both C4.5 and C4.5rules [61], while maintaining an advantage in efficiency for large and noisy datasets. JRip is the WEKA implementation of RIPPER. The default JRip parameter values were used in all experiments.

6.2.3 Boosting Techniques

Boosting is a meta learning technique that iteratively builds an ensemble of models, adjusting example weights after each iteration. Examples that were misclassified in the previous iteration have their weights increased, while correctly classified examples have their weights decreased. In doing so, examples that were misclassified in the current iteration are more likely to be correctly classified in subsequent iterations. We present 10 versions of boosting in this work, which are all based on the popular AdaBoost algorithm [26]. This section provides descriptions of the boosting algorithms utilized in this study. AveBoost2 is presented in Section 6.2.3.1; ORBoost in Section 6.2.3.2; SMOTEBoost in Section 6.2.3.3; RUSBoost in Section 6.2.3.4; and finally, the six hybrid boosting algorithms are described in section 6.2.3.5.

One boosting implementation detail that is often not addressed in the literature is how the example weights are accounted for when building the weak hypothesis.

The simplest way, when possible, is for the classifier to directly incorporate the example weights into whatever calculations it performs in building a model, known as boosting by reweighting. When the classifier is incapable of directly handling weighted instances (or if the practitioner simply prefers), boosting by resampling may be used. Boosting by resampling provides a universal way to incorporate these weights, which is to create a temporary training dataset (often of the same size as the original) using sampling (we'll refer to this as S'_t). The examples in the original training dataset are sampled with replacement with probabilities of being chosen proportional to their weights. In this work, we present results using only boosting by resampling, since it can be applied to any base learner, and previous research has shown this option to perform as well or better than boosting by reweighting for most boosting algorithms [69].

6.2.3.1 AveBoost2

Oza developed the *AveBoost2* algorithm [58] for improved generalization performance, especially in the presence of noisy data. AveBoost2 is a revision of the earlier AveBoost algorithm, altered to provide theoretical bounds on training and generalization error. While training error is greater than that for AdaBoost, generalization error is decreased. AveBoost2 improves performance on noisy examples by averaging the example weights produced at each iteration with those from the previous iterations. This averaging process prevents the weights of potentially mislabeled

```
Algorithm AveBoost2
Given: Set $S$ of examples $(x_1, y_1), ..., (x_m, y_m)$
Weak learner $WeakLearn$
Number of iterations $T$

  1. Initialize $D_1(i) = \frac{1}{m}$ for all $i$.
  2. Do for $t = 1, 2, ..., T$
     (a) Create temporary training dataset $S'_t$ with distribution $D_t$ by
         sampling with replacement:
     (b) Call $WeakLearn$, providing it with examples $S'_t$.
     (c) Get back a hypothesis $h_t : X \to Y$.
     (d) Calculate the error (for $S$ and $D_t$): $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$.
     (e) Calculate the weight update parameters:
         i. $\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$
         ii. $\gamma_t = \frac{2(1 - \epsilon_t)t + 1}{2\epsilon_t t + 1}$
     (f) Update $D_t$:
         i. $w_i = D_t(i) \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$
         ii. $c_t(i) = \frac{w_i}{\sum_{i=1}^{m} w_i}$
         iii. $D_{t+1}(i) = \frac{tD_t(i) + c_t(i)}{t + 1}$
  3. Output the final hypothesis: $H(x) = \operatorname{argmax}_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t \gamma_t}$.
```

Figure 6.1: The AveBoost2 algorithm

instances from quickly increasing, avoiding the overfitting which might occur using the original AdaBoost algorithm. Oza tested AveBoost2 versus AdaBoost with three base learners on nine UCI datasets, both with and without noise injected into each class, and found AveBoost2 to improve performance, especially on the noisy datasets. Pseudocode for the AveBoost2 algorithm is given in Figure 6.1.

6.2.3.2 ORBoost

The *Outlier Removal Boosting* algorithm (ORBoost) [45], developed by Karmaker and Kwek, takes a different approach to handling noise. Also utilizing the framework of the AdaBoost algorithm, ORBoost effectively eliminates instances with exceptionally high weights rather than moderating those weights via an averaging process like AveBoost2. Once an instance's weight exceeds the given threshold, it is determined to be an outlier and its weight is set to zero on the next iteration, equivalent to removing it from the training data. The determination of the weight threshold is important, as too high a value will result in little to no outliers being removed, whereas too low a threshold leads to removing too many 'clean' examples and thus a possible decrease in classification accuracy. They set the weight threshold based on an integer parameter, d, ranging from 3 to 20. 20% of the training instances are set aside as a validation set, and the remaining 80% are used to create ORBoost classifiers with each of the values for d between 3 and 20. In the boosting process, training instances were discarded if their weight grew to exceed $\frac{d}{m'}$, where m' is the number of remaining training instances. The d value that produces the lowest error on the validation set is then assumed to be optimal and the model built with that threshold is the final classifier. Pseudocode for the ORBoost algorithm is given in Figure 6.2.

```
Algorithm ORBoost
Given: Set $S$ of examples $(x_1, y_1), ..., (x_m, y_m), y \in Y = [-1, +1]$
Weak learner $WeakLearn$
Number of iterations $T$
Threshold $d$

  1. Initialize $D_1(i) = \frac{1}{m}$ for all $i$.
  2. Do for $t = 1, 2, ..., T$
     (a) Create temporary training dataset $S'_t$ with distribution $D_t$ by
         sampling with replacement:
     (b) Call $WeakLearn$, providing it with examples $S'_t$.
     (c) Get back a hypothesis $h_t : X \to \mathbb{R}$.
     (d) Calculate the error (for $S$ and $D_t$): $\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
     (e) Update $D_t$:
         i. $w_{t+1}(i) = w_t(i) \times \begin{cases} \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} & \text{if } h_t(x_i) \neq y_i \\ \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} & \text{otherwise} \end{cases}$
         ii. If $w_{t+1}(i) \geq d$, then $w_{t+1}(i) = 0$
         iii. $D_{t+1}(i) = D_t(i)(w_{t+1}(i))^{(y_i h_t(x_i))}$
     (f) Normalize $D_{t+1}$: Let $Z_t = \sum_i D_{t+1}(i)$. Set $D_{t+1}(i) = \frac{D_{t+1}(i)}{Z_t}$.
  3. Output the final hypothesis: $H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$ where
     $\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
```

Figure 6.2: The ORBoost algorithm

6.2.3.3 SMOTEBoost

Another boosting method aimed at improving performance on imbalanced data is the *SMOTEBoost* technique proposed by Chawla et al. [15]. Based on the AdaBoost.M2 algorithm [26], instead of theoretically or heuristically deriving new update rules as some previous methods aimed at class imbalance have done, SMOTEBoost combines boosting with intelligent data oversampling, a data level technique for improving classification performance on imbalanced data. The AdaBoost.M2 method can accommodate multiple classes (though in this work we focus on the binary classification problem) and utilizes confidence rated predictions between 0 and 1 for each class ($h_t(x, y) : X \times Y \rightarrow [0, 1]$) instead of a single real-valued confidence rating whose sign determines which of two classes to predict as in RareBoost or AdaCost. Their Synthetic Minority Oversampling Technique (SMOTE) [14] is used to add synthetic minority class examples to the dataset. In SMOTEBoost, this sampling technique is applied to the training data before it is sent to the weak learner to produce a classification model at each iteration of the boosting procedure. This requires another parameter to be specified to the boosting algorithm: the amount of oversampling to apply. Figure 6.3 shows the entire SMOTEBoost algorithm.

Chawla et al. provide a parameter, N, to the SMOTEBoost algorithm which specifies how many synthetic minority instances to add to the data (as a percentage of the original number of minority instances) each time SMOTE is applied. Due to varying levels of class imbalance in different datasets, and thus varying levels of effect caused by adding constant multiples of the number of minority class examples, we choose to let N represent the percentage of the total number of instances that should be represented by the minority class and calculate the number of examples to add dynamically based on this percentage and the numbers of instances in each class in the original dataset. After the weights are determined at each iteration and the temporary training dataset S'_t has been created by sampling with replacement (Figure 6.3, step 2ai), SMOTE is applied to it so that the minority class becomes $N\%$ of the resulting S'_t (Figure 6.3, step 2aii). The random number seed for SMOTE is set differently at each iteration (e.g., to t) to increase the diversity of the ensemble produced. If the minority class already represents greater than or equal to $N\%$ of S'_t, then no sampling is applied. After this altered training dataset is created, the AdaBoost.M2 procedure continues as usual.

6.2.3.4 RUSBoost

Our previous research on data sampling techniques for improving classification of imbalanced data has shown SMOTE to perform well [78], though it was consistently outperformed by a much simpler data sampling technique: random undersampling (RUS) of majority class instances. If adding SMOTE to boosting can produce such good results, then shouldn't adding a sampling technique that performs even better produce even better results? In addition to superior classification performance, RUS is very simple to implement and decreases the time needed to train a classifier since

```
Algorithm SMOTEBoost
Given: Set $S$ of examples $(x_1, y_1), ..., (x_m, y_m)$ with minority class $y^r$, $|Y| = 2$
Weak learner $WeakLearn$
Number of iterations $T$
Desired percentage of total instances to be represented by the minority class $N$

1. Initialize $D_1(i) = \frac{1}{m}$ for all $i$.
2. Do for $t = 1, 2, ..., T$
   (a) Create temporary training dataset $S'_t$ with distribution $D'_t$:
       i. Create $S'_t$ by sampling with replacement.
       ii. If $\frac{\sum_{i:y_i=y^r} 1}{m} < N\%$, add synthetic minority class examples
           using the SMOTE technique until $\frac{\sum_{i:y_i=y^r} 1}{\sum_{i:i\in S'_t} 1} = N\%$.
   (b) Call $WeakLearn$, providing it with examples $S'_t$ and their weights $D'_t$.
   (c) Get back a hypothesis $h_t : X \times Y \to [0, 1]$.
   (d) Calculate the pseudo-loss (for $S$ and $D_t$): $\epsilon_t = \sum_{(i,y):y_i\neq y} D_t(i)(1 - h_t(x_i, y_i) + h_t(x_i, y))$.
   (e) Calculate the weight update parameter: $\alpha_t = \frac{\epsilon_t}{1-\epsilon_t}$.
   (f) Update $D_t$: $D_{t+1}(i) = D_t(i)\alpha_t^{\frac{1}{2}(1+h_t(x_i,y_i)-h_t(x_i,y:y\neq y_i))}$.
   (g) Normalize $D_{t+1}$: Let $Z_t = \sum_i D_{t+1}(i)$. Set $D_{t+1}(i) = \frac{D_{t+1}(i)}{Z_t}$.
3. Output the final hypothesis: $H(x) = \underset{y\in Y}{\operatorname{argmax}} \sum_{t=1}^{T} h_t(x, y) \log \frac{1}{\alpha_t}$.
```

Figure 6.3: The SMOTEBoost algorithm for a binary class

it decreases the size of a dataset. To this end we have introduced the *RUSBoost* technique [70], based on the successful SMOTEBoost algorithm.

RUSBoost works very similarly to SMOTEBoost, with the only difference being the sampling technique applied to the training dataset sent to the weak learner at each iteration. Instead of applying SMOTE to oversample the minority class, RUS is applied to randomly remove examples from the majority class (step 2aii). As in SMOTEBoost, a different random number seed is provided to the RUS procedure at each iteration to increase ensemble diversity. By repetitively undersampling the training data (rather than undersampling it only once), RUSBoost mitigates the main drawback of random undersampling (loss of information). Information that is not present during the construction of one iteration's model is likely to be included in the training of subsequent iteration's models, especially if that information is important (the example will be given a higher weight by boosting if it is misclassified by the current iteration's hypothesis). Figure 6.4 provides the RUSBoost algorithm.

6.2.3.5 Hybrid Boosting Algorithms

AveBoost2 and ORBoost are intended for dealing with class label noise, but they don't incorporate any means to combat class imbalance. Conversely, SMOTEBoost and RUSBoost aim to improve boosting performance in the presence of class imbalance, however they lack any mechanisms for accounting for noisy data. Both class noise and class imbalance are common data quality issues that affect many real

```
Algorithm RUSBoost
Given: Set $S$ of examples $(x_1, y_1), ..., (x_m, y_m)$ with minority class $y^r \in Y$, $|Y| = 2$
Weak learner $WeakLearn$
Number of iterations $T$
Desired percentage of total instances to be represented by the minority class $N$

1. Initialize $D_1(i) = \frac{1}{m}$ for all $i$.
2. Do for $t = 1, 2, ..., T$
    (a) Create temporary training dataset $S'_t$ with distribution $D'_t$:
        i. Create $S'_t$ by sampling with replacement.
        ii. If $\frac{\sum_{i:y_i=y^r} 1}{m} < N\%$, remove majority class examples using
            random undersampling until $\frac{\sum_{i:y_i=y^r} 1}{\sum_{i:i \in S'_t} 1} = N\%$.
    (b) Call $WeakLearn$, providing it with examples $S'_t$ and their weights $D'_t$.
    (c) Get back a hypothesis $h_t : X \times Y \rightarrow [0, 1]$.
    (d) Calculate the pseudo-loss (for $S$ and $D_t$): $\epsilon_t = \sum_{(i,y):y_i \neq y} D_t(i)(1 - h_t(x_i, y_i) + h_t(x_i, y))$.
    (e) Calculate the weight update parameter: $\alpha_t = \frac{\epsilon_t}{1 - \epsilon_t}$.
    (f) Update $D_t$: $D_{t+1}(i) = D_t(i)\alpha_t^{\frac{1}{2}(1+h_t(x_i,y_i)-h_t(x_i,y:y \neq y_i))}$.
    (g) Normalize $D_{t+1}$: Let $Z_t = \sum_i D_{t+1}(i)$. Set $D_{t+1}(i) = \frac{D_{t+1}(i)}{Z_t}$.
3. Output the final hypothesis: $H(x) = \underset{y \in Y}{\text{argmax}} \sum_{t=1}^{T} h_t(x, y) \log \frac{1}{\alpha_t}$.
```

Figure 6.4: The RUSBoost algorithm for a binary class

world classification problems, and can jointly occur. To deal with the combination of these hurdles, we examined four combinations of the concepts used in the four aforementioned boosting algorithms to evaluate their performance compared to each other and to the four originals. The two pertinent concepts used for class noise are the weight averaging of AveBoost2 and the weight thresholding of ORBoost, while the two driving influences for class imbalance are oversampling the training data at each iteration using SMOTE in SMOTEBoost and undersampling each training set using random undersampling in RUSBoost.

AveSMOTEBoost (A-SB) and *AveRUSBoost* (A-RB) add the sampling step of SMOTEBoost (SB) and RUSBoost (RB) to the framework of the AveBoost2 (AB) algorithm. Right before the training dataset for each iteration is passed to the weak learner (between steps 2a and 2b in Figure 6.1), either SMOTE or RUS is applied to create a modified dataset to train the learner. This balances the class distribution, so that the inherent robustness to noise that AveBoost2 possesses is augmented to improve the classification performance for imbalanced data. *ORSMOTEBoost* (O-SB) and *ORRUSBoost* (O-RB) are similar hybrids. For these two algorithms, the weight thresholding step of ORBoost (ORB) is added to the framework of SMOTE-Boost/RUSBoost, with weights being set to zero between steps 2f and 2g in Figures 6.3 and 6.4 if they exceed the threshold. Also, as in regular ORBoost, the same iterative threshold discovery process using the 20% validation set is incorporated into both O-SB and O-RB.

Though not originating from AveBoost2 or ORBoost, we investigated another pair of SMOTEBoost/RUSBoost hybrids. In *CleanseSMOTEBoost* (C-SB) and *CleanseRUSBoost* (C-RB), a classification filtering procedure is applied right before applying either SMOTE or RUS (between steps 2ai and 2aii in the pseudocode for the two algorithms) to remove potentially noisy instances before sampling the data and building the weak learner. The classification filter was applied by performing 10-fold cross-validation on the training data S'_t using Naive Bayes. Any majority instances that were misclassified (and thus potentially noisy), were removed from the training set. Only majority instances are removed to avoid further reducing the already sparse population of minority instances. For each of the 10 folds in the filtering procedure, all the minority instances were included in each training set since they weren't going to be removed and accordingly didn't need to be evaluated in the test sets. After this noise cleansing process, SMOTE or RUS was applied as in the original SB and RB algorithms, and all the other steps proceeded unaltered.

6.2.4 Experimental Design

Two separate sets of experiments were performed to evaluate the 10 boosting techniques. In the first set of experiments, the boosting techniques were applied in conjunction with four classifiers on 15 unaltered real world datasets. 10-fold cross-validation was used, so for each fold, $\frac{9}{10}$ of the data was used as training data and the final $\frac{1}{10}$ of the data was used as the test data. To ensure the statistical validity of the results, 10 independent runs of each 10-fold cross-validation process were performed.

The second set of experiments is performed using subsets of the four datasets described in Section 6.2.1 with controlled class distributions, noise levels, and noise distributions. For each combination of the different levels of these parameters, 100 independent trials (10 runs of 10-fold cross-validation) are performed, so that the set of instances which are included in the subset and the set of those instances which have their class labels switched (from the minority class to the majority class or vice versa) is different each time. Each subset of the original training data is sampled to contain 1500 examples and a class distribution according to our experimental parameters. These subsets are then corrupted to include the desired level and distribution of class noise. It is then, once this final, processed subset of data is constructed, that the boosting technique is applied. The three experimental factors pertinent to this second set of experiments, class distribution, noise level, and noise distribution, are described in this section.

6.2.4.1 Experimental Factor: Class Distribution

The first experimental factor, class distribution (CD), indicates the percentage of examples in the derived dataset belonging to the minority class. The experiments in this work consider six levels of class distribution, $CD = \{1, 2, 4, 6, 8, 10\}$, where the value of CD indicates the percentage of examples in the training data that belong to the minority class. For example, a derived dataset with $CD = 4$ has 4% of its examples from the minority class and 96% of its examples from the majority class. Note that this is the ratio prior to noise injection (discussed in the following sections). Depending on the noise distribution, the final dataset may contain a different class distribution.

6.2.4.2 Experimental Factor: Noise Level

The second factor, noise level (NL), determines the quantity of noisy examples in the training data. The selected datasets are relatively clean, so NL is varied by artificially injecting noise into the dataset. This is accomplished by swapping the class value of some of the examples. The number of examples with their classes swapped is a function of NL and the number of minority examples in the derived dataset.

While many works involving noise injection often inject noise by simply selecting x% of the examples and corrupting their class, this technique may be inappropriate when dealing with imbalanced datasets. For example, if a dataset contains only 1% of its examples belonging to the minority class, and as little as 10% of its examples are corrupted (injected with noise), the minority class will become overwhelmed by noisy examples from the majority class. Instead, we corrupt a percentage of the examples based on the number of minority examples in the dataset.

In our experiments, we use five levels of noise, $NL = \{10, 20, 30, 40, 50\}$, where NL determines the number of examples, based on the size of the minority class, that will be injected with noise. The actual number of noisy examples will be:

$$2 \times \frac{NL}{100} \times P$$

where P is the number of positive (minority) examples in the dataset. For example, a dataset with $CD = 10$ and $NL = 20$ will have 150 minority examples (10% of 1500) and $2 \times 0.2 \times 150 = 60$ noisy examples. Note that this does not indicate which examples (minority or majority) will be corrupted. That is determined by the final experimental factor: noise distribution.

6.2.4.3 Experimental Factor: Noise Distribution

The final experimental factor, noise distribution (ND), determines the type of noise that will be injected into the data. When dealing with binary class datasets (the only kind considered in this work), there are two possible noise types: P→N and N→P. Noise of type N→P is when an instance that should be labeled "negative" is incorrectly labeled as "positive." Conversely, P→N noise is when an example that should be labeled "positive" is instead labeled "negative."

The experiments in this work use five levels of noise distribution, $ND = \{0, 25, 50, 75, 100\}$, where the value of ND indicates the percentage of noisy examples that are of the P→N variety. For example, if a derived dataset is to contain 60 noisy examples, and $ND = 25$, then 25% (15) of those noisy examples will be minority ("positive") class examples that have their labels changed to "negative" (P→N), while the remaining 75% (45) of the noisy examples will be of the N→P variety. Due to the definition of ND, the combination of $ND = 100$ and $NL = 50$ can not be used, since the resulting dataset would have zero minority class examples.

6.3 Results

This section presents the results of the thorough empirical experimentation performed to assess the classification performance of 10 boosting techniques (described in Section 6.2.3) in the presence of imbalanced and noisy data. Two separate sets of experiments were run. First, the 10 boosting techniques were tested on 15 imbalanced real world datasets without any modification to determine how each technique performs for regular, potentially noiseless data. Subsequently, we performed a more comprehensive set of experiments using four other imbalanced datasets in which we precisely controlled the dataset size, class distribution, amount of class noise, and distribution of that noise among the two classes, as described in Section 6.2.4.

6.3.1 Sampling Percentage

Eight of the 10 boosting techniques considered in this work add data sampling to the boosting process in order to improve performance on imbalanced data. Though a balanced class distribution (50% positive class instances) is often targeted for data sampling for imbalanced data, our previous work [48] has shown a 2:1 class distribution (35% positive class instances) to work well based on ROC curves, especially for severely imbalanced data. As two of the boosting techniques, AveBoost2 (AB) and ORBoost (ORB), are not intended specifically to deal with class noise and don't add data sampling to the boosting process, they require no sampling percentage parameter. Accordingly, in this section we present some preliminary analysis to determine which percentage to use for the eight boosting techniques which do require the parameter when comparing all 10 techniques together.

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	3	209.80	<.0001	187.50	<.0001
Percentage	1	5.22	0.0223	33.10	<.0001
Type	7	24.01	<.0001	24.59	<.0001
Learner*Percentage	3	11.15	<.0001	0.33	0.8005
Learner*Type	21	4.23	<.0001	1.19	0.2516
Percentage*Type	7	3.08	0.0030	6.80	<.0001
Learner*Percent*Type	21	0.83	0.6870	2.00	0.0041

Table 6.2: Regular data: ANOVA models showing the significance of each experimental factor based on both the AROC and APRC values.

The ANOVA results analyzing the impact of the three main factors (Learner, Percentage, and Type) in the experiments with 15 unaltered datasets, and their two and three way interaction terms, for the AROC and APRC response variables are given in Table 6.2. As shown, all three main factors are statistically significant at the $\alpha = 0.05$ level for both AROC and APRC. All three two way interaction terms were found to be significant based on AROC, but only the interaction between Percentage and Type is significant according the APRC metric. The three way interaction term is statistically significant only for the APRC performance measure. ANOVA analysis was performed using SAS [63].

Since each of the main factors are shown to be statistically significant, we perform Tukey's Honestly Significant Difference (HSD) test, indicating which levels of these factors result in significantly different performances when compared to the other levels of that factor. For example, the first part of Table 6.3 provides the results of the HSD test for the main factor Learner on the 15 unaltered datasets. This factor has four levels (the four different base classifiers used by the boosting techniques in our experiments), each of which is assigned to a group (indicated by a letter) based on its average performance (across all of the other factors, as well as 100 independent runs of each combination of main factors). Learners in group A performed better than those in group B, which performed better than those in group C, and so forth. If multiple levels (Learners) are assigned to the same group (i.e., they have the same letter in a column in this table) then their average performances were not significantly different based on the performance metric indicated in that column.

AROC			APRC		
Factor	Mean	HSD	Factor	Mean	HSD
Learner					
C4.5N	0.808	A	C4.5N	0.483	A
RIPPER	0.796	B	C4.5D	0.471	B
C4.5D	0.794	B	RIPPER	0.470	B
NB	0.778	C	NB	0.433	C
Percentage					
50	0.795	A	35	0.469	A
35	0.793	B	50	0.460	B
Type					
A-RB	0.801	A	A-SB	0.481	A
A-SB	0.800	AB	SB	0.474	AB
RB	0.798	ABC	O-SB	0.467	BC
O-RB	0.795	BC	RB	0.466	BC
SB	0.794	C	A-RB	0.466	BC
C-RB	0.794	C	O-RB	0.461	CD
O-SB	0.786	D	C-SB	0.454	DE
C-SB	0.785	D	C-RB	0.445	E

Table 6.3: Regular data: mean values and HSD groupings for each experimental factor for both AROC and APRC.

The four levels of the Learner main factor performed similarly, relative to each other, for both AROC and APRC. C4.5N significantly outperforms the other three. RIPPER and C4.5D are both in group B and thus their results are statistically indistinguishable from each other, and they both produced better results than NB which is alone in group C. The focus of this section is the second main factor, Percentage. Averaged over the different levels of the other two factors, there is not a large difference between the two levels of Percentage for the 15 plain datasets, though the differences are statistically significant. For AROC, 50% produces better average results than 35%, while 35% outperforms 50% for the APRC performance metric. Averaged over the four base learners and two sampling percentages, the relative performances of the different boosting types show some variance depending on the performance metric. For AROC, AveRusBoost (A-RB), AveSMOTEBoost (A-SB), and RUSBoost (RB) all fall into group A. ORSMOTEBoost (O-SB) and CleanseSMOTEBoost (C-SB) achieve the lowest AROC results in group D, while the results for ORRUSBoost (O-RB), SMOTEBoost (SB), and CleanseRUSBoost(C-RB) lie in between. A-SB also performs well based on APRC, falling into group A again, this time with SB. C-SB and C-RB produce the lowest values, falling into group E, while the rest again fall into the middle in groups B, C, and D.

Table 6.4 presents the mean AROC and APRC values and the corresponding HSD groupings for the two levels of Percentage for each level of main factors Learner

	AROC				APRC			
	35%		50%		35%		50%	
Factor	Mean	HSD	Mean	HSD	Mean	HSD	Mean	HSD
Learner								
C4.5D	0.794	A	0.795	A	0.476	A	0.465	B
C4.5N	0.808	A	0.808	A	0.487	A	0.479	B
NB	0.779	A	0.776	A	0.437	A	0.430	B
RIPPER	0.791	B	0.801	A	0.475	A	0.465	B
Type								
A-RB	0.805	A	0.797	B	0.479	A	0.452	B
A-SB	0.797	B	0.803	A	0.479	A	0.483	A
C-RB	0.792	A	0.795	A	0.452	A	0.439	B
C-SB	0.784	A	0.786	A	0.455	A	0.453	A
O-RB	0.795	A	0.795	A	0.472	A	0.451	B
O-SB	0.784	A	0.788	A	0.467	A	0.467	A
RB	0.796	A	0.800	A	0.473	A	0.459	B
SB	0.791	B	0.796	A	0.473	A	0.475	A

Table 6.4: Regular data: mean values and HSD groupings of the two percentages for each level of experimental factors Learner and Type for both AROC and APRC.

and Type. Using the AROC performance measure, both the results for both percentages are not significantly different for most of the four base learners and eight boosting techniques. For the RIPPER learner, 50% produces significantly higher performance, as it also does for A-SB and SB, while for A-RB, 35% achieves higher AROC values. For the APRC response variable, 35% is in group A for every level of Learner and Type. Using this metric, 50% is significantly outperformed for each of the four base learners, and in half of the boosting techniques it produces statistically significantly lower values, while in the other half the difference between the two percentages is not significant.

Table 6.5 presents the ANOVA results for each of the six main factors (Learner, Percentage, ClassDistribution, NoiseLevel, NoiseDistribution, and Type), as well as

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	3	77542.00	<.0001	43958.80	<.0001
Percentage	1	5537.31	<.0001	4582.04	<.0001
ClassDistribution	5	344586.00	<.0001	315281.00	<.0001
NoiseLevel	4	203711.00	<.0001	312438.00	<.0001
NoiseDistribution	4	86954.30	<.0001	112803.00	<.0001
Type	7	29230.80	<.0001	7720.98	<.0001
Learner*Percentage	3	621.71	<.0001	2901.76	<.0001
Learner*ClassDistribution	15	4215.23	<.0001	978.29	<.0001
Percentage*ClassDistribution	5	6078.50	<.0001	7713.09	<.0001
Learner*NoiseLevel	12	2755.35	<.0001	2500.22	<.0001
Percentage*NoiseLevel	4	159.39	<.0001	39.46	<.0001
ClassDistribution*NoiseLevel	20	2189.88	<.0001	592.86	<.0001
Learner*NoiseDistribution	12	3765.56	<.0001	4795.31	<.0001
Percentage*NoiseDistribution	4	209.30	<.0001	221.44	<.0001
ClassDistribution*NoiseDistribution	20	2524.41	<.0001	832.27	<.0001
NoiseLevel*NoiseDistribution	15	23467.20	<.0001	22851.10	<.0001
Learner*Type	21	2194.24	<.0001	1419.04	<.0001
Percentage*Type	7	3425.82	<.0001	4454.28	<.0001
ClassDistribution*Type	35	6740.56	<.0001	4119.95	<.0001
NoiseLevel*Type	28	215.26	<.0001	313.52	<.0001
NoiseDistribution*Type	28	2699.44	<.0001	1459.55	<.0001

Table 6.5: Noise injection: ANOVA models showing the significance of each experimental factor based on both the AROC and APRC values.

all of their two way interactions, using both performance metrics from the noise injection experiments with the other four imbalanced datasets. All main factors and all of their two way interactions were statistically significant at the $\alpha = 0.05$ level for both response variables. The mean values and HSD groupings of each level of each factor for both metrics are presented in Table 6.6. C4.5N was again the best learner, while NB improved for this set of experiments, falling into group B, followed by C4.5D and then RIPPER. For this set of experiments, where the class imbalance and class noise were tightly controlled, Percentage=35% performed significantly better than 50% for both performance metrics. The ClassDistribution, NoiseLevel, and

NoiseDistribution factors affect results predictably, with performance decreasing as the level of class imbalance, class noise, or amount of the noise being injected into the minority class increases. One exception to this pattern is with NoiseDistribution=100 not producing the worst performance for either AROC or APRC, which is due to the fact that NoiseLevel=50 cannot be performed for this level of NoiseDistribution. Among the eight levels of Type, A-SB significantly outperforms all other boosting techniques for both performance metrics. As was the case for the 15 regular datasets, A-RB performed well (group B this time) based on AROC, while SB produced good APRC results (also group B). For both response variables, O-SB is in the last group, being significantly outperformed by every other boosting technique on the data with injected noise.

The mean values and HSD groupings of the two levels of Percentage based on both AROC and APRC are given in Table 6.7. Among the four levels of Learner, 35% produces significantly better results than 50% for all cases except with RIPPER using the APRC metric, where 50% performs significantly better. Agreeing with previous results [48], 35% provides better classification performance at the more severely imbalanced datasets (ClassDistribution={1,2} for both AROC and APRC and ClassDistribution=4 for APRC), while 50% produces significantly better values for three less imbalanced levels of ClassDistribution. For both noise related factors, NoiseLevel and NoiseDistribution, 35% significantly outperforms 50% at all levels. The

AROC			APRC		
Factor	Mean	HSD	Factor	Mean	HSD
Learner					
C4.5N	0.934	A	C4.5N	0.725	A
NB	0.932	B	NB	0.715	B
C4.5D	0.917	C	C4.5D	0.692	C
RIPPER	0.897	D	RIPPER	0.656	D
Percentage					
35	0.922	A	35	0.702	A
50	0.918	B	50	0.692	B
ClassDistribution					
10	0.954	A	10	0.777	A
8	0.951	B	8	0.768	B
6	0.945	C	6	0.755	C
4	0.934	D	4	0.724	D
2	0.896	E	2	0.636	E
1	0.840	F	1	0.522	F
NoiseLevel					
10	0.954	A	10	0.803	A
20	0.941	B	20	0.755	B
30	0.924	C	30	0.703	C
40	0.893	D	40	0.623	D
50	0.878	E	50	0.577	E
NoiseDistribution					
0	0.943	A	0	0.765	A
25	0.933	B	25	0.725	B
50	0.916	C	100	0.684	C
100	0.909	D	50	0.676	D
75	0.896	E	75	0.632	E
Type					
A-SB	0.929	A	A-SB	0.722	A
A-RB	0.926	B	SB	0.712	B
SB	0.926	C	C-SB	0.706	C
C-SB	0.926	D	A-RB	0.702	D
C-RB	0.925	E	RB	0.692	E
RB	0.925	E	O-RB	0.691	F
O-RB	0.921	F	C-RB	0.689	G
O-SB	0.883	G	O-SB	0.662	H

Table 6.6: Noise injection: mean values and HSD groupings for each experimental factor for both AROC and APRC.

best sampling percentage varied according to the sampling type used in the boosting techniques. For those techniques based on random undersampling (RUS), 35% always was significantly better than 50%, while those techniques based on SMOTE performed significantly better using a sampling percentage of 50%. As more of the levels of the other factors had 35% outperforming 50%, and since the difference between the two percentages was more in the cases where 35% produced better results, we utilize 35% as the sampling percentage in the next section for comparing the eight boosting techniques which require the percentage with the other two that do not.

6.3.2 Boosting Technique Comparison

In this section we focus on the relative performance among the 10 boosting techniques. As before, there are two sets of experiments. We first apply the boosting techniques to 15 regular datasets to determine performance in normal circumstances without significant (known) class noise. We then test the 10 boosting algorithms in the context of our noise injection experiments in which we vary the levels of class distribution, class noise, and distribution of noise among the classes to gauge the effect of these factors on the boosting performance. As established in the previous section, for the eight boosting techniques which require a sampling percentage parameter, 35% is utilized, where the percentage represents the percentage of the post sampling training dataset that is represented by the positive (minority) class.

	AROC				APRC			
	35%		50%		35%		50%	
Factor	Mean	HSD	Mean	HSD	Mean	HSD	Mean	HSD
Learner								
C4.5D	0.920	A	0.914	B	0.704	A	0.681	B
C4.5N	0.938	A	0.930	B	0.736	A	0.714	B
NB	0.932	A	0.931	B	0.717	A	0.713	B
RIPPER	0.898	A	0.895	B	0.651	B	0.661	A
ClassDistribution								
1	0.853	A	0.826	B	0.554	A	0.490	B
2	0.901	A	0.892	B	0.651	A	0.621	B
4	0.934	A	0.934	A	0.725	A	0.722	B
6	0.944	B	0.947	A	0.751	B	0.759	A
8	0.949	B	0.952	A	0.761	B	0.775	A
10	0.952	B	0.955	A	0.768	B	0.785	A
NoiseLevel								
10	0.956	A	0.953	B	0.808	A	0.798	B
20	0.943	A	0.939	B	0.760	A	0.750	B
30	0.926	A	0.922	B	0.708	A	0.698	B
40	0.897	A	0.890	B	0.629	A	0.617	B
50	0.881	A	0.876	B	0.581	A	0.574	B
NoiseDistribution								
0	0.945	A	0.941	B	0.771	A	0.759	B
25	0.934	A	0.931	B	0.728	A	0.722	B
50	0.918	A	0.915	B	0.679	A	0.674	B
75	0.899	A	0.893	B	0.637	A	0.627	B
100	0.913	A	0.905	B	0.694	A	0.675	B
Type								
A-RB	0.935	A	0.918	B	0.720	A	0.684	B
A-SB	0.927	B	0.931	A	0.715	B	0.730	A
C-RB	0.930	A	0.919	B	0.703	A	0.676	B
C-SB	0.923	B	0.928	A	0.698	B	0.713	A
O-RB	0.930	A	0.911	B	0.717	A	0.665	B
O-SB	0.880	B	0.886	A	0.655	B	0.670	A
RB	0.930	A	0.920	B	0.703	A	0.681	B
SB	0.924	B	0.928	A	0.704	B	0.719	A

Table 6.7: Noise injection: mean values and HSD groupings of the two percentages for each level of the other experimental factors for both AROC and APRC.

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	3	85.79	<.0001	170.92	<.0001
Type	9	357.46	<.0001	106.86	<.0001
Learner*Type	27	7.10	<.0001	5.11	<.0001

Table 6.8: Regular data: ANOVA models showing the significance of each experimental factor based on both the AROC and APRC values.

The ANOVA table for the AROC and APRC response variables from the experiments with 15 regular datasets is displayed in Table 6.8, showing the effects of the two main factors (Learner and Type) and their interaction. Both main factors and their lone interaction term are found to be statistically significant at the $\alpha = 0.05$ level for both performance measures, so we perform Tukey's Honestly Significant Difference test for each level of the two main factors. The mean values and the corresponding HSD groupings for both response variables are given in Table 6.9. The addition of regular AveBoost2 and ORBoost, as well as the removal of the 50% sampling percentage, don't shake up the rankings for the base learners on the 15 regular datasets. As before, for both metrics, C4.5N is in group A, C4.5D and RIPPER are in group B, and NB is significantly outperformed by them all, falling into group C. There is also little change in the top ranking levels of the Type factor for either performance metric. A-RB and A-SB again produce the best AROC values, both in group A, followed by RB and O-RB behind them. For APRC, A-SB and SB are still on top in group A, though without the 50% sampling percentage, A-RB and RB join them also in group A, as do O-RB and O-SB. The results clearly show that AveBoost2 and ORBoost, due to a lack of mechanism for addressing class imbalance, suffer from poor performance on these imbalanced datasets.

The HSD groupings for AROC and APRC of each level of the Type main factor on the 15 regular datasets are listed horizontally in Table 6.10 for each of the four base learners. According to the AROC performance measure, for C4.5N and

AROC			APRC		
Factor	Mean	HSD	Factor	Mean	HSD
Learner					
C4.5N	0.792	A	C4.5N	0.476	A
C4.5D	0.777	B	C4.5D	0.464	B
RIPPER	0.773	B	RIPPER	0.462	B
NB	0.768	C	NB	0.416	C
Type					
A-RB	0.805	A	A-RB	0.479	A
A-SB	0.797	AB	A-SB	0.479	A
RB	0.796	B	SB	0.473	A
O-RB	0.795	B	RB	0.473	A
C-RB	0.792	B	O-RB	0.472	A
SB	0.791	BC	O-SB	0.467	AB
O-SB	0.784	C	C-SB	0.455	BC
C-SB	0.784	C	C-RB	0.452	C
ORB	0.726	D	ORB	0.411	D
AB	0.705	E	AB	0.381	E

Table 6.9: Regular data: mean values and HSD groupings for each experimental factor for both AROC and APRC.

NB, all eight boosting techniques aimed at class imbalance are in group A, while only AB and ORB fall into groups B or C. The C4.5D and RIPPER learners were more effected by the boosting algorithm used. In both cases, A-RB and O-RB are in group A, and A-SB, C-RB, and RB are in group A for one of the two. Regardless of learner, AB and ORB end up in the lowest HSD groupings, outperformed by all the other boosting methods. As with AROC, C4.5N has AB and ORB in group B and the eight other techniques in group A, as does the RIPPER base learner. For C4.5D, AB and ORB are in group D, while all other boosting methods are in group A except for C-RB and C-SB. For Learner=NB, AB is significantly outperformed by ORB, which gets significantly worse results than all the other techniques, all of which are in group A except, again, for C-RB and C-SB. Overall, A-RB and O-RB are in group A for all four levels of Learner according to both performance metrics, while A-SB and RB are in group A for all but one learner based on AROC.

AROC										
Learner	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
C4.5D	A	AB	BC	C	AB	BC	D	D	BC	BC
C4.5N	A	A	A	A	A	A	B	B	A	A
NB	A	A	A	A	A	A	C	B	A	A
RIPPER	A	BCD	ABC	DE	ABCD	E	G	F	AB	CD
APRC										
Learner	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
C4.5D	A	A	C	BC	ABC	ABC	D	D	ABC	AB
C4.5N	A	A	A	A	A	A	B	B	A	A
NB	AB	A	C	BC	ABC	ABC	E	D	ABC	ABC
RIPPER	A	A	A	A	A	A	B	B	A	A

Table 6.10: Regular data: HSD groupings of the 10 boosting techniques for each level of experimental factor Learner for both AROC and APRC.

Table 6.11 shows the ANOVA results for each of the five main factors in the noise injection experiments (Learner, ClassDistribution, NoiseLevel, NoiseDistribution, and Type) and all of their two way interactions, all of which are statistically significant at the $\alpha = 0.05$ level for both response variables, AROC and APRC. The mean and HSD groupings for each of these performance measures are listed for each level of all five main factors in Table 6.12. C4.5N remains the best base learner based on both performance metrics. It is again (for these noise injection experiments) followed in order by NB, C4.5D, and RIPPER, all producing statistically distinct results. The different levels of ClassDistribution, NoiseLevel, and NoiseDistribution affect performance in the same way as usual and bear no further discussion. A-RB

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	3	43819.40	<.0001	28862.60	<.0001
ClassDistribution	5	207245.00	<.0001	150871.00	<.0001
NoiseLevel	4	105699.00	<.0001	156156.00	<.0001
NoiseDistribution	4	77012.00	<.0001	75011.90	<.0001
Type	9	125146.00	<.0001	37918.10	<.0001
Learner*ClassDistribution	15	1175.66	<.0001	214.54	<.0001
Learner*NoiseLevel	12	1524.55	<.0001	1260.94	<.0001
ClassDistribution*NoiseLevel	20	208.46	<.0001	321.84	<.0001
Learner*NoiseDistribution	12	1993.19	<.0001	2670.53	<.0001
ClassDistribution*NoiseDistribution	20	268.85	<.0001	431.90	<.0001
NoiseLevel*NoiseDistribution	15	14721.00	<.0001	13031.10	<.0001
Learner*Type	27	4541.79	<.0001	2079.99	<.0001
ClassDistribution*Type	45	7867.76	<.0001	2198.32	<.0001
NoiseLevel*Type	36	540.41	<.0001	257.76	<.0001
NoiseDistribution*Type	36	4459.89	<.0001	1554.03	<.0001

Table 6.11: Noise injection: ANOVA models showing the significance of each experimental factor based on both the AROC and APRC values.

produced significantly better performance than the other techniques for both performance metrics, though not by a large amount. It is followed in AROC performance by C-RB, O-RB, and RB in group B, A-SB in group C, and SB and C-SB in group D, none of which perform differently by a large margin. The three that do perform noticeably worse are O-SB, ORB, and AB, in groups E, F, and G, respectively. These three also perform distinctly worse than the other techniques and are in the last three HSD groups according to the APRC metric. For this response variable, O-RB and A-SB trail A-RB in groups B and C, respectively, though the difference is again small.

The HSD groupings based on AROC and APRC for the 10 boosting techniques at each level of main factors Learner, ClassDistribution, NoiseLevel, and NoiseDistribution are layed out horizontally in Tables 6.13 and 6.13, respectively. In the noise injection experiments, the performance of the boosting techniques for each level of

AROC			APRC		
Factor	Mean	HSD	Factor	Mean	HSD
Learner					
C4.5N	0.920	A	C4.5N	0.711	A
NB	0.910	B	NB	0.688	B
C4.5D	0.888	C	C4.5D	0.665	C
RIPPER	0.879	D	RIPPER	0.631	D
ClassDistribution					
10	0.941	A	10	0.752	A
8	0.936	B	8	0.744	B
6	0.929	C	6	0.730	C
4	0.911	D	4	0.697	D
2	0.866	E	2	0.609	E
1	0.813	F	1	0.510	F
NoiseLevel					
10	0.936	A	10	0.778	A
20	0.922	B	20	0.732	B
30	0.902	C	30	0.678	C
40	0.869	D	40	0.599	D
50	0.858	E	50	0.560	E
NoiseDistribution					
0	0.931	A	0	0.750	A
25	0.918	B	25	0.708	B
50	0.896	C	50	0.655	C
100	0.877	D	100	0.646	D
75	0.871	E	75	0.605	E
Type					
A-RB	0.935	A	A-RB	0.720	A
C-RB	0.930	B	O-RB	0.717	B
O-RB	0.930	B	A-SB	0.715	C
RB	0.930	B	SB	0.704	D
A-SB	0.927	C	RB	0.703	D
SB	0.924	D	C-RB	0.703	D
C-SB	0.923	D	C-SB	0.698	E
O-SB	0.880	E	O-SB	0.655	F
ORB	0.809	F	ORB	0.562	G
AB	0.807	G	AB	0.560	H

Table 6.12: Noise injection: mean values and HSD groupings for each experimental factor for both AROC and APRC.

Learner	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
C4.5D	A	BC	C	D	B	E	G	F	CD	CD
C4.5N	D	A	BC	B	E	F	G	H	C	A
NB	A	D	D	F	B	G	I	H	C	E
RIPPER	A	E	B	D	C	G	H	I	C	F
ClassDistribution	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
1	B	E	AB	C	D	F	G	H	A	C
2	A	D	B	D	C	E	F	G	B	D
4	A	D	C	E	B	F	H	G	CD	E
6	A	C	C	D	B	E	G	F	C	D
8	A	C	C	D	B	E	G	F	C	D
10	A	BC	CD	E	B	F	H	G	D	E
NoiseLevel	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
10	A	DE	BC	E	CD	F	G	G	AB	DE
20	A	C	B	D	B	E	G	F	B	D
30	A	C	B	D	B	E	G	F	B	D
40	A	C	B	D	B	E	F	F	B	D
50	A	B	B	C	B	D	E	E	B	C
NoiseDistribution	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
0	A	A	C	C	BC	D	F	E	B	B
25	A	B	C	D	B	E	G	F	C	C
50	A	C	B	D	BC	E	G	F	BC	D
75	A	D	B	DE	BC	F	G	H	C	E
100	A	C	A	C	B	D	E	F	A	C

Table 6.13: Noise injection: AROC HSD groupings of the 10 boosting techniques for each level of the other experimental factors.

learner was more variable than when applied to the 15 unaltered datasets. As has been the case throughout, AB and ORB performed the worst. They are in the lowest HSD groups for all levels of the other factors and both performance metrics, except for RIPPER based on APRC, where O-SB produced significantly worse results than AB (but significantly better than ORB). O-SB, in turn, is most often significantly outperformed by all other levels of Type. A-RB managed to be in group A for all learners but C4.5N based on AROC, and for C4.5D and RIPPER according to APRC. A-SB and SB ranked highest second most frequently (based on the combination of four levels of Learner and two performance metrics), both falling into group A twice out

Learner	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
C4.5D	A	A	E	D	C	F	H	G	E	B
C4.5N	BC	B	A	CD	D	E	F	G	AB	AB
NB	B	C	F	F	A	DE	H	G	D	E
RIPPER	A	C	B	E	BC	H	G	I	D	F
ClassDistribution	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
1	D	B	C	AB	E	F	G	H	C	A
2	AB	A	D	BCD	BC	E	F	G	DC	ABC
4	A	B	C	E	A	F	H	G	C	D
6	A	B	C	E	A	D	G	F	C	D
8	A	C	D	F	B	D	H	G	D	E
10	A	C	DE	G	B	D	I	H	E	F
NoiseLevel	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
10	C	A	AB	AB	A	D	F	E	A	A
20	B	B	CD	D	A	E	G	F	CD	C
30	A	B	C	D	A	E	F	F	C	C
40	A	B	C	E	B	F	G	H	CD	D
50	A	B	D	E	C	E	F	G	D	D
NoiseDistribution	A-RB	A-SB	C-RB	C-SB	O-RB	O-SB	AB	ORB	RB	SB
0	C	A	E	D	D	E	G	F	D	B
25	A	A	C	C	A	D	F	E	C	B
50	A	C	D	E	B	F	G	G	DE	D
75	A	B	B	D	A	E	F	G	C	D
100	B	C	A	D	B	E	F	G	B	D

Table 6.14: Noise injection: APRC HSD groupings of the 10 boosting techniques for each level of the other experimental factors.

of eight possibilities. Throughout the rest of the other factors, the patterns remain the same. A-RB tends to perform well more consistently than the other techniques, often being placed in HSD group A, and is trailed closely by A-SB and O-RB for AROC and APRC, and by C-RB and RB as well for AROC alone.

6.4 Conclusions

Class imbalance and class noise are prevalent hinderances to successful classification tasks. Numerous techniques for addressing each problem have been proposed, including data sampling, cost-sensitive learning, and ensemble classification methods.

Ensemble techniques, especially, have received serious attention in recent research. For dealing with class noise, the AveBoost2 algorithm averages the instance weights across each iteration of boosting, moderating the growth in weight (and thus potential disruptive influence) of potentially noisy instances. ORBoost is another boosting variant intended for situations involving class noise. ORBoost applies a threshold to the amount of weight an instance can achieve, ignoring any instances whose weights exceed the threshold in all further iterations. Several modified boosting techniques have also been proposed for handling imbalanced data. Two of the most successful are SMOTEBoost and RUSBoost, which improve performance on imbalanced data by applying SMOTE and RUS, respectively, to the training dataset at each iteration before building the base classifier.

Neither SMOTEBoost nor RUSBoost, however, contain any mechanism to account for noisy data. None of the other class imbalance inspired boosting techniques we are aware of make any such considerations either. Similarly, both AveBoost2 and ORBoost focus solely on accounting for class noise and make no attempt to improve classification performance in the presence of imbalanced data. In fact, since minority instances could easily be confused for noisy data by a classifier, these techniques may actually perform even worse than standard boosting in the presence of class imbalance. This lack of coverage of the union of the two data quality issues motivated the investigation of six new hybrid boosting techniques. AveSMOTEBoost and AveRUSBoost combine the robustness to noise of AveBoost2 with the improved minority class identification of SMOTEBoost and RUSBoost, respectively. Similarly, ORSMOTEBoost and ORRUSBoost add the weight-thresholding-based outlier removal of ORBoost to SMOTEBoost and RUSBoost, respectively. Additionally, CleanseSMOTEBoost and CleanseRUSBoost add a noise filtering component to SMOTEBoost and RUSBoost, so that the training datasets at each iteration are sampled only after potentially noisy majority instances have been removed by a classification filter using Naive Bayes. These six hybrid boosting algorithms, along with AveBoost2, ORBoost, SMOTEBoost, and RUSBoost, were put through a gauntlet of empirical experiments using four different base classifiers. First they were applied to 15 unaltered real world datasets to examine their performance on typical data which may or may not contain class noise. Later they were utilized in the framework of an extensive set of noise injection experiments in which four other real world datasets were sampled and had noise injected to precisely control their size, class distribution, noise level, and noise distribution. These experiments allowed us to compare these 10 boosting techniques, as well as examine the effects of the base learner, class distribution, noise level, and noise distribution on their performance. Our major conclusions based on these experimental results are provided below.

1. Accounting for class imbalance is paramount. Of the 10 boosting techniques tested, only two of them, AveBoost2 and ORBoost, didn't include mechanisms for addressing the problem of imbalanced data. These two techniques almost

invariably resulted in worse performance than all eight other boosting methods (based on both performance metrics presented), regardless of the values for the other experimental factors such as base learner, class distribution, noise level, and noise distribution.

2. Though AveBoost2 generally resulted in inferior performance to ORBoost in these experiments, when combined with boosting techniques directed at class imbalance (i.e. RUSBoost and SMOTEBoost), the noise handling mechanism in AveBoost2 seemed to outperform that of ORBoost. AveRUSBoost and AveSMOTEBoost were consistently some of the top performing boosting algorithms, outperforming ORRUSBoost and ORSMOTEBoost, respectively. In addition to producing better classification performance, AveBoost2's weight averaging mechanism is computationally less intensive then ORBoost's thresholding mechanism by virtue of lacking the demanding iterative search for the best weight threshold that ORBoost utilizes.

3. The noise handling aspect of the boosting process didn't produce a tremendous benefit much of the time, at least in the context of noisy data. Even when the differences between, for example, AveRUSBoost and regular RUSBoost or AveSMOTEBoost and regular SMOTEBoost, were statistically significant, they were often fairly small. This seems to suggest, either, that the class imbalance is much more detrimental to the learning process than class noise is, or that the

mechanisms utilized to address class noise don't work very well for imbalanced data. Identifying noise in imbalanced data is a more difficult problem than when classes are evenly distributed, as the rarely occurring minority instances can seem like outliers themselves and could be mistaken for noise in the data.

The problem of class noise requires special treatment when present in concert with class imbalance. Because the minority class instances can already seem like noise due to their rarity, standard noise handling procedures can be too blunt to avoid exacerbating the class imbalance problem. These experiments have shown that applying techniques aimed solely at handling noise are easily outperformed on noisy imbalanced data by techniques which specifically address class imbalance, even if they don't also address noise. Combining the strengths of techniques aimed separately at class imbalance and class noise proved a fruitful endeavor. Three of the best performing boosting techniques throughout these experiments were AveRUSBoost (combining AveBoost2 and RUSBoost), AveSMOTEBoost (combining AveBoost2 and SMOTEBoost), and ORRUSBoost (combining ORBoost and RUSBoost). It's unclear whether typical noise handling techniques, when applied alone, don't seem to perform well for noisy imbalanced data because the effects of class noise pale in comparison to those of class imbalance or because the standard noise handling techniques aren't suited to accurately detecting noise when many of the minority class instances may already seem spurious. Further study is needed to determine this. It would also be interesting to see if the performance of AveBoost2 and ORBoost on imbalanced data could be improved without direct modification of the boosting algorithm, perhaps by performing data sampling before the boosting or using cost sensitive classifiers as base learners.

CHAPTER 7

EMPIRICAL ANALYSIS OF CLASSIFICATION FILTERING FOR NOISY IMBALANCED DATA

7.1 Introduction

Accurate data is essential for machine learning algorithms to produce reliable predictive models. Low quality input leads to low quality models and poor predictions. The most frequent approach to identifying noisy data tends to be classification filtering [49, 86], in which a classifier is built and evaluated on the training data (usually using cross-validation) and the instances that are misclassified are considered to be noisy. Different aspects of classification filtering have been explored in many different studies. Different classifiers have been used for filtering [86, 47], while others have used ensembles of classifiers to avoid being limited by the induction bias of a single learner [28, 49]. Some implementations removed the instances identified as noisy [11], while others 'corrected' their class label [74]. In most instances of classification filtering, instances from all classes are identified as noisy, however, in cases of class imbalance, researchers have investigated only removing instances from the majority class [6]. Another facet of classification filtering is the decision threshold used to determine the classification. Many learning algorithms can output a numeric probability indicating the level confidence that the instance belongs to a given class. With imbalanced data, the default threshold on this posterior probability is often quite suboptimal, and thus is inappropriate also for identifying potentially noisy instances. The decision threshold used in classification filtering hasn't received any direct attention, to our knowledge. The only indirect treatment of this topic is, when using classifier ensembles for noise filtering, the issue of whether to require a majority vote [49] or a consensus (unanimous) vote [11] for identifying noisy instances, which essentially equates to two different decision thresholds on the output of the classifier ensemble.

While various details of the classification filtering procedure have been independently explored, they have not received thorough, integrated treatment. This void in the research, combined with the dearth of attention to the conjunction of class noise and class imbalance, stimulated the current study. In this work, 35 imbalanced real world datasets are used to perform an extensive battery of experiments to empirically test the performance of classification filtering options, including the learner used for filtering, whether filtered instances are removed or relabeled, whether instances are filtered from both classes or just the majority class, and how the decision threshold for identifying noisy instances is determined. Two sets of experiments are performed. First, each of the combinations of noise filtering parameters are applied to 31 of the datasets, unaltered, to measure the performance on typical data with ambiguous noise levels. Then, the remaining four datasets, having been determined relatively free of noise, are aritifically manipulated using a number of experimental factors, precisely controlling the datasets' size, class distribution, noise level, and distribution of noise among the classes, and each of the parameterizations of the classification filter are tested under all these conditions to gauge their effects on filter performance. In all cases, after applying the classification filters as a data preprocessing technique, 11 different classification algorithms are used to build the actual predictive models to determine the interaction between filter parameters and predictive learner. All results are tested for statistical significance using ANOVA models and multiple pairwise comparison tests.

Details of our empirical experiments are given in Section 7.2. This section lists the datasets used, descriptions of the classification algorithms and their parameters, details of the classification filtering process, and specifics of the noise injection process. The results of the exhaustive experimentation are presented in Section 7.3. Finally, concluding remarks are presented in Section 7.4.

7.2 Experiments

To explore, in detail, the various options for classification filtering and their effect on noisy, imbalanced data, the WEKA machine learning tool [87], written in Java, was used. Section 7.2.1 provides information about the datasets used for experimentation. Section 7.2.2 describes the 11 classification algorithms used. Section

7.2.3 provides the specific classification filtering procedure. Section 7.2.4 explains the overall experimental design, including the data sampling and noise injection procedures.

7.2.1 Datasets

In total, 35 real world datasets were used for these experiments. All of them are imbalanced, ranging from 1.33% to 34.90% minority instances. Some are proprietary datasets, while the others were aquired from the UCI Machine Learning Repository [8]. Many of the datasets originally contained multiple class values, but were transformed into binary class problems by leaving one class as the minority class and combining the rest of the classes into one large majority class. The datasets selected for use in these experiments, as well as information about their size and class distributions, can be found in Table 7.1. In this study we performed two sets of experimentation. First we tested the various incarnations of classification filtering on the first 31 datasets from Table 7.1. In the second set of experiments, we evaluated these preprocessing techniques in the presence of controlled class imbalance and class noise using the last four datasets. We selected these four datasets due to their sizes, and more importantly the number of minority class examples. These characteristics facilitiated our experimental design, as decribed in Section 7.2.4. Most importantly, however, is that these datasets are relatively clean. That is, models constructed on these datasets (using ten runs of ten-fold cross validation) resulted in nearly perfect classification performance prior to noise injection.

7.2.2 Learners

All 11 classifiers were used as implemented in the WEKA machine learning tool [87]. Unless otherwise noted, the default parameters for the classifier were used.

7.2.2.1 C4.5

C4.5 is the benchmark decision tree learning algorithm proposed by Quinlan [61]. C4.5 is among the most commonly used learning algorithms in data mining research. The decision tree is built using an entropy-based splitting criterion stemming from information theory [2]. C4.5 improves Quinlan's older ID3 [60] decision tree algorithm by adding support for tree pruning and dealing with missing values and numeric attributes. The WEKA version of C4.5 is called J48. Two different versions of the C4.5 classifier were used in this study, denoted C4.5D and C4.5N. C4.5D uses the default parameter settings as in WEKA, while C4.5N uses no pruning and Laplace smoothing [85].

7.2.2.2 Naive Bayes

Naive Bayes [54] (NB) is a quick and simple classifier that utilizes Bayes's rule of conditional probability. It is 'naive' in that it assumes that all predictor variables are independent. Although this assumption rarely holds true in real-world data, Naive

Dataset	# Positive	# Negative	# Total	% Positive	% Negative
SP3	47	3494	3541	1.33%	98.67%
SP4	92	3886	3978	2.31%	97.69%
MAMMOGRAPHY	260	10923	11183	2.32%	97.68%
SOLAR-FLARE-F	51	1338	1389	3.67%	96.33%
CAR-3	69	1659	1728	3.99%	96.01%
SP2	189	3792	3981	4.75%	95.25%
CCCS-12	16	266	282	5.67%	94.33%
SP1	229	3420	3649	6.28%	93.72%
PC1	76	1031	1107	6.87%	93.13%
MW1	31	372	403	7.69%	92.31%
GLASS-3	17	197	214	7.94%	92.06%
KC3	43	415	458	9.39%	90.61%
CM1	48	457	505	9.50%	90.50%
CCCS-8	27	255	282	9.57%	90.43%
PENDIGITS-5	1055	9937	10992	9.60%	90.40%
SATIMAGE-4	626	5809	6435	9.73%	90.27%
ECOLI-4	35	301	336	10.42%	89.58%
SEGMENT-5	330	1980	2310	14.29%	85.71%
KC1	325	1782	2107	15.42%	84.58%
JM1	1687	7163	8850	19.06%	80.94%
LETTER-VOWEL	3878	16122	20000	19.39%	80.61%
CCCS-4	55	227	282	19.50%	80.50%
KC2	106	414	520	20.38%	79.62%
CONTRACEPTIVE-2	333	1140	1473	22.61%	77.39%
VEHICLE-1	212	634	846	25.06%	74.94%
HABERMAN	81	225	306	26.47%	73.53%
YEAST-2	429	1055	1484	28.91%	71.09%
PHONEME	1586	3818	5404	29.35%	70.65%
CCCS-2	83	199	282	29.43%	70.57%
CREDIT-GERMAN	300	700	1000	30.00%	70.00%
DIABETES	268	500	768	34.90%	65.10%
Datasets for noise injection					
NURSERY-3	328	12632	12960	2.53%	97.47%
LETTER-A	789	19211	20000	3.95%	96.06%
OPTDIGITS-8	554	5066	5620	9.86%	90.14%
SPLICE-2	768	2422	3190	24.08%	75.92%

Table 7.1: Characteristics of experimental datasets

Bayes has been shown to often perform well, even in the presence of strong attribute dependencies [19]. We use the default parameters for Naive Bayes in our experiments.

7.2.2.3 Multilayer Perceptron

Multilayer perceptrons [62] (MLP) attempt to artificially mimic the functioning of a biological nervous system. Multiple nodes, or 'neurons,' are connected in layers, with the output of each node being the thresholded weighted sum of its inputs from the previous layer. The network weights are usually learned using a gradient descent algorithm called back-propagation. It has been shown that a multiple hidden layer neural network can approximate any function [38]. We change two parameters in our experiments. The 'hiddenLayers' parameter was changed to 3 to define a network with one hidden layer containing three nodes, and the 'validationSetSize' parameter was changed to 10 so that 10% of the data would be used as a validation set to determine when to stop the iterative training process.

7.2.2.4 RIPPER

RIPPER (Repeated Incremental Pruning to Produce Error Reduction) [16] is a rule-based learner that modifies the IREP algorithm [27] to improve accuracy without sacrificing efficiency. RIPPER has been shown to produce classifier accuracy that compares favorably with both C4.5 and C4.5rules [61], while maintaining an advantage in efficiency for large and noisy datasets. JRip is the WEKA implementation of RIPPER. The default JRip parameter values were used in all experiments.

7.2.2.5 K Nearest Neighbors

K nearest neighbors [3], abbreviated kNN, is a 'lazy learning' technique. The training dataset serves as a *case library* which is used to predict the class of an instance based on it's *k* 'nearest neighbors' in the case library. The nearest neighbors are those instances in the case library that are closest in the feature-space to the instance being classified based on some distance measure, usually Euclidean distance. Two versions of the kNN classifier were used in our experiments, denoted 2NN and 5NN. 2NN uses two nearest neighbors for classification and 5NN uses five nearest neighbors. Also, the 'distanceWeighting' parameter was set to 'Weight by 1/distance.'

7.2.2.6 Support Vector Machines

A *support vector machine* (SVM) [65] is a linear classifier which attempts to learn the maximum margin hyperplane separating two classes in the data. This maximum margin hyperplane is usually determined by a small subset of instances called *support vectors*. In order to learn nonlinear decision boundaries, the data can be transformed by a kernel function. The linear maximum margin hyperplane then found for the transformed data can represent a nonlinear partitioning of the original feature space. SMO is the implementation of support vector machines in WEKA. Two changes to the default parameters were made: the complexity constant 'c' was changed from 1.0 to 5.0, and the 'buildLogisticModels' parameter, which allows proper probability estimates to be obtained, was set to true. By default, SMO uses a linear kernel.

7.2.2.7 Random Forests

Random forests [10] (RF100) give accuracy that compares favorably with Adaboost [26] and is relatively robust to noise. This learner utilizes bagging [9] and the 'random subspace method' [33] to construct an ensemble (forest) of randomized, unpruned decision trees. The outputs of this forest of decision trees are combined to produce the ultimate prediction. The 'numTrees' parameter was changed to 100 to specify using a forest of 100 trees.

7.2.2.8 Radial Basis Function Network

A *radial basis function network* [56] (RBF), like MLP, is a type of artificial neural network. The radial basis function outputs a value based on the radial distance of a feature vector from a 'center' in the feature space. RBFs typically have two processing layers. The data is first input to each RBF in the first layer. The second layer then linearly combines the output of the radial basis functions to produce the final output. The only parameter change for RBF (called RBFNetwork in WEKA) was to set the parameter 'numClusters' to 10.

7.2.2.9 Logistic Regression

Logistic regression (LR) is a a type of generalized linear model which can be used to predict a binary dependent variable [31]. While LR does allow categorical or continuous independent variables, categorical variables must be encoded (e.g., 0,1) to facilitate classification modeling. No changes to the default parameter values were made.

7.2.3 Classification Filtering Procedure

In this work we used the classification filtering procedure described in Section 3.3. 10 was used for the number of folds for cross-validation. We tested two different classifiers (previously described in Section 7.2.2) for performing the noise filtering, RF100 and NB. We chose these classifiers because, based on our recent work (presented in Chapter 4), we found them most efficient at identifying the actually noisy instances injected into imbalanced datasets. We tested both removing the instances found to be noisy ('Remv'), as well as relabeling them ('Rlbl'). We also tested identifying noise in just the majority class ('Maj') as well as both the minority and majority classes ('Both'). We tested nine different parameters for the decision threshold. Five of them are percentages, {2%,5%,10%,15%,25%}, indicating that we identified as potentially noisy that percentage of the instances (separately for each class) with the highest probability of membership in the opposite class. For example, if we are removing the noisy instances from both classes, there are 20 minority instances and 100 majority instances, and we are using a threshold parameter of 10%, the two minority instances with the highest probability of membership in the majority class will be removed and the 10 majority instances with the highest probability of membership in the minority class will be removed. Three other thresholds are based on the statistics of the probabilities produced. Labeled $\{1.0S,1.5S,2.0S\}$, these thresholds correspond to a number of standard deviations from the mean the probability must be to be removed. For example, take again removing instances from both classes with 20 and 100 instances, respectively. If the average probability of membership in the majority class for the minority instances is 0.1 and the standard deviation is 0.2, then a threshold of 1.5S indicates that any minority instances whose probabilities of membership in the majority class are greater than or equal to 0.4 will be removed. In the case of both the percentage-based and the standard-deviation-based threshold parameters, the actual probability thresholds will vary depending on the results of the cross-validation, so they are calculated dynamically at runtime. The final threshold parameter we used, 0.5, was static. With binary data, this is the default decision threshold (the class with the highest predicted probability of membership is the one that the instance is assigned to) and is the threshold most often used in reports of classification filtering.

7.2.4 Experimental Design

Two separate sets of experiments were performed to evaluate the different options for classification filtering. In the first set of experiments, the boosting techniques were applied in conjunction with four classifiers on 15 unaltered real world datasets. 10-fold cross-validation was used, so for each fold, $\frac{9}{10}$ of the data was used as training data and the final $\frac{1}{10}$ of the data was used as the test data. To ensure the statistical validity of the results, 10 independent runs of each 10-fold cross-validation process were performed.

The second set of experiments is performed using subsets of the four datasets described in Section 7.2.1 with controlled class distributions, noise levels, and noise distributions. For each combination of the different levels of these parameters, 100 independent trials (10 runs of 10-fold cross-validation) are performed, so that the set of instances which are included in the subset and the set of those instances which have their class labels switched (from the minority class to the majority class or vice versa) is different each time. Each subset of the original training data is sampled to contain 1500 examples and a class distribution according to our experimental parameters. These subsets are then corrupted to include the desired level and distribution of class noise. It is then, once this final, processed subset of data is constructed, that the classification filter is applied. The three experimental factors pertinent to this second set of experiments, class distribution, noise level, and noise distribution, are described in this section.

7.2.4.1 Experimental Factor: Class Distribution

The first experimental factor, class distribution (CD), indicates the percentage of examples in the derived dataset belonging to the minority class. The experiments in this work consider six levels of class distribution, $CD = \{1, 2, 4, 6, 8, 10\}$, where the value of CD indicates the percentage of examples in the training data that belong to the minority class. For example, a derived dataset with $CD = 4$ has 4% of its examples from the minority class and 96% of its examples from the majority class. Note that this is the ratio prior to noise injection (discussed in the following sections). Depending on the noise distribution, the final dataset may contain a different class distribution.

7.2.4.2 Experimental Factor: Noise Level

The second factor, noise level (NL), determines the quantity of noisy examples in the training data. The selected datasets are relatively clean, so NL is varied by artificially injecting noise into the dataset. This is accomplished by swapping the class value of some of the examples. The number of examples with their classes swapped is a function of NL and the number of minority examples in the derived dataset.

While many works involving noise injection often inject noise by simply selecting x% of the examples and corrupting their class, this technique may be inappropriate when dealing with imbalanced datasets. For example, if a dataset contains only 1% of its examples belonging to the minority class, and as little as 10% of its examples are corrupted (injected with noise), the minority class will become overwhelmed by noisy examples from the majority class. Instead, we corrupt a percentage of the examples based on the number of minority examples in the dataset.

In our experiments, we use five levels of noise, $NL = \{10, 20, 30, 40, 50\}$, where NL determines the number of examples, based on the size of the minority class, that will be injected with noise. The actual number of noisy examples will be:

$$2 \times \tfrac{NL}{100} \times P$$

where P is the number of positive (minority) examples in the dataset. For example, a dataset with $CD = 10$ and $NL = 20$ will have 150 minority examples (10% of 1500) and $2 \times 0.2 \times 150 = 60$ noisy examples. Note that this does not indicate which examples (minority or majority) will be corrupted. That is determined by the final experimental factor: noise distribution.

7.2.4.3 Experimental Factor: Noise Distribution

The final experimental factor, noise distribution (ND), determines the type of noise that will be injected into the data. When dealing with binary class datasets (the only kind considered in this work), there are two possible noise types: P→N and N→P. Noise of type N→P is when an instance that should be labeled "negative" is incorrectly labeled as "positive." Conversely, P→N noise is when an example that should be labeled "positive" is instead labeled "negative."

The experiments in this work use five levels of noise distribution, $ND = \{0, 25, 50, 75, 100\}$, where the value of ND indicates the percentage of noisy examples that are of the P→N variety. For example, if a derived dataset is to contain 60 noisy examples, and $ND = 25$, then 25% (15) of those noisy examples will be minority ("positive") class examples that have their labels changed to "negative" (P→N), while the remaining 75% (45) of the noisy examples will be of the N→P variety. Due to the definition of ND, the combination of $ND = 100$ and $NL = 50$ can not be used, since the resulting dataset would have zero minority class examples.

7.3 Results

Here we present the results of the extensive empirical experiments performed to evaluate the performance of the different parameters involved in classification filtering. As explained in Section 7.2, two sets of experiments were performed. The first set evaluated the effect of the classification filtering options on 31 unaltered, imbalanced real world datasets, the results of which are given in Section 7.3.1. Section 7.3.2 shows the results for the second set of experiments, which used four other real world datasets, this time controlling the dataset size, class distribution, noise level, and noise distribution.

7.3.1 Experiments With Regular Data

Table 7.2 presents the ANOVA table analyzing the impact of the five main experimental factors (Learner, FilterLearner, BothClasses, Treatment, and Threshold), as well as all of their two way interactions, for both the AROC and APRC performance measures. As shown, all five main factors are statistically significant at the $\alpha = 0.05$ level for APRC, and all but Treatment are significant based on AROC. All of the two way factor interactions were found to be significant for AROC, while all but the interaction between BothClasses and Treatment were statistically significant

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	10	15503.50	<.0001	12594.30	<.0001
FilterLearner	1	1590.37	<.0001	10629.30	<.0001
BothClasses	1	1428.86	<.0001	243.32	<.0001
Treatment	1	1.22	0.2700	5692.81	<.0001
Threshold	8	1334.16	<.0001	3115.03	<.0001
Learner*FilterLearner	10	74.61	<.0001	109.55	<.0001
Learner*BothClasses	10	51.68	<.0001	8.49	<.0001
FilterLearner*BothClasses	1	632.12	<.0001	304.37	<.0001
Learner*Treatment	10	291.53	<.0001	418.79	<.0001
FilterLearner*Treatment	1	861.28	<.0001	1320.30	<.0001
BothClasses*Treatment	1	14.79	0.0001	1.22	0.2700
Learner*Threshold	80	89.87	<.0001	30.91	<.0001
FilterLearner*Threshold	8	436.66	<.0001	363.28	<.0001
BothClasses*Threshold	8	195.93	<.0001	66.10	<.0001
Treatment*Threshold	8	182.81	<.0001	190.20	<.0001

Table 7.2: ANOVA models showing the significance of each experimental factor based on both the AROC and APRC values for the 31 regular datasets

using APRC as the response variable. ANOVA analysis was performed using SAS [63].

Since each of the main factors are shown to be statistically significant for at least one of the response variables, we perform Tukey's Honestly Significant Difference (HSD) test, indicating which levels of these factors result in significantly different performances when compared to the other levels of that factor. For example, the first part of Table 7.3 provides the results of the HSD test for the main factor Learner on the 31 unaltered datasets. This factor has 11 levels (the 11 different classifiers used in our experiments), each of which is assigned to a group (indicated by a letter) based on its average performance (across all of the other factors, as well as 100 independent runs of each combination of main factors). Learners in group A performed better than those in group B, which performed better than those in group C, and so forth.

If multiple levels (Learners) are assigned to the same group (i.e., they have the same letter in a column in this table) then their average performances were not significantly different based on the performance metric indicated in that column.

AROC			APRC		
Factor	Mean	HSD	Factor	APRC	HSD
Learner					
RF100	0.849	A	RF100	0.553	A
5NN	0.837	B	5NN	0.551	A
LR	0.832	C	MLP	0.549	B
MLP	0.830	D	LR	0.524	C
2NN	0.824	E	2NN	0.523	C
SVM	0.821	F	SVM	0.521	D
C4.5N	0.818	G	RBF	0.498	E
NB	0.810	H	C4.5N	0.468	F
RBF	0.806	I	NB	0.456	G
C4.5D	0.760	J	C4.5D	0.409	H
RIPPER	0.739	K	RIPPER	0.389	I
FilterLearner					
RF100	0.815	A	RF100	0.510	A
NB	0.808	B	NB	0.479	B
BothClasses					
Maj	0.814	A	Maj	0.497	A
Both	0.808	B	Both	0.492	B
Treatment					
Rlbl	0.811	A	Remv	0.506	A
Remv	0.811	A	Rlbl	0.483	B
Threshold					
10%	0.819	A	2%	0.520	A
15%	0.818	A	5%	0.515	B
2.0S	0.817	B	2.0S	0.514	B
1.5S	0.816	B	1.5S	0.505	C
5%	0.815	C	10%	0.499	D
1.0S	0.814	D	1.0S	0.493	E
2%	0.810	E	0.5	0.491	F
0.5	0.797	F	15%	0.481	G
25%	0.796	G	25%	0.435	H

Table 7.3: Mean AROC and APRC values and HSD groupings of each level of each of the main factors for the classification filtering experiments with 31 regular datasets

As the focus of this work is on classification filtering, we refrain from discussion of Learner as a single factor. Averaged across all the levels of the other main factors, FilterLearner RF100 was shown to produce significantly better performance than NB for AROC and, by a larger margin, for APRC. For the BothClasses factor, treating noisy instances from only the majority classes worked significantly better than treating both classes for both performance measures. As indicated in Table 7.2, the methods of treating the noisy instances were not significantly different based on AROC, but removing the noisy instances was found to produce a higher APRC value than relabeling them for these 31 unaltered datasets. The ideal Threshold parameter was shown to depend strongly on the performance metric, though the 0.5 and 25% thresholds fared poorly for both. For AROC, 10% and 15% produced the best noise handling performance, while more conservative thresholds of 2% and 5% produced the highest APRC values. Threshold=2.0S, though, performed consistently regardless of performance metric, falling into HSD group B for both measures.

Table 7.4 displays the mean AROC and APRC values and their HSD groupings of the two levels of factor FilterLearner for all levels of all the other factors. The HSD groupings are listed horizontally, so, for example, for Learner=2NN, FilterLearner=NB was in group B for both AROC and APRC, while FilterLearner=RF100 was in group A for both measures, and thus had statistically significantly better performance. In fact, looking down the APRC columns, RF100 was the superior level of FilterLearner for every level of every other factor. The situation is similar for AROC as well. Threshold=25% is the only time that NB produced significantly better performance than RF100, though for Learner={C4.5D,RBF} and Threshold=2.0S, the two levels of FilterLearner produced results that were statistically indistinguishable at the $\alpha = 0.05$ significance level.

	AROC				APRC			
Factor	NB		RF100		NB		RF100	
	Mean	HSD	Mean	HSD	Mean	HSD	Mean	HSD
Learner								
2NN	0.821	B	0.826	A	0.512	B	0.535	A
5NN	0.832	B	0.842	A	0.534	B	0.568	A
C4.5D	0.760	A	0.759	A	0.395	B	0.423	A
C4.5N	0.811	B	0.824	A	0.445	B	0.491	A
LR	0.829	B	0.835	A	0.513	B	0.536	A
MLP	0.826	B	0.833	A	0.534	B	0.563	A
NB	0.805	B	0.814	A	0.444	B	0.469	A
RBF	0.806	A	0.806	A	0.479	B	0.516	A
RF100	0.842	B	0.855	A	0.526	B	0.579	A
RIPPER	0.735	B	0.742	A	0.373	B	0.405	A
SVM	0.821	B	0.822	A	0.512	B	0.530	A
BothClasses								
Maj	0.813	B	0.816	A	0.484	B	0.510	A
Both	0.803	B	0.814	A	0.474	B	0.511	A
Treatment								
Remv	0.810	B	0.812	A	0.496	B	0.516	A
Rlbl	0.806	B	0.817	A	0.462	B	0.504	A
Threshold								
2%	0.809	B	0.810	A	0.514	B	0.526	A
5%	0.813	B	0.817	A	0.506	B	0.523	A
10%	0.815	B	0.822	A	0.485	B	0.512	A
15%	0.813	B	0.824	A	0.462	B	0.500	A
25%	0.802	A	0.790	B	0.424	B	0.446	A
1.0S	0.804	B	0.823	A	0.470	B	0.516	A
1.5S	0.813	B	0.820	A	0.488	B	0.521	A
2.0S	0.817	A	0.817	A	0.504	B	0.524	A
0.5	0.786	B	0.808	A	0.456	B	0.525	A

Table 7.4: Mean AROC and APRC values and HSD groupings of the two levels of FilterLearner for each level of the other main factors for the classification filtering experiments with 31 regular datasets

The mean AROC and APRC values and their HSD groupings of the two levels of factor BothClasses for all levels of all the other factors are depicted in Table 7.5. Again, the HSD groupings are displayed horizontally comparing the two levels of BothClasses, Maj and Both, for all the levels of all the other factors. Unsurprisingly, because all the datasets were unbalanced, Maj generally produced superior performance to Both as measure by both performance metrics. For the AROC response variable, almost all results for Maj are significantly higher than those produced by Both. In only two cases, Threshold={2%,5%}, the performances of Maj and Both are not different by a statistically significant margin. Based on APRC, Maj is still never significantly outperformed by Both. However, for two different learners (2NN and NB), FilterLearner=RF100, and four levels of Threshold (2%, 5%, 1.5S, and 2.0S), a difference cannot be statistically discerned at a significance level of 0.05.

Similar to Tables 7.4 and 7.5, Table 7.6 shows the mean AROC and APRC results and corresponding HSD groupings for the two levels of Treatment (Remv and Rlbl) for each level of every other main factor. As has tended to be the case so far, one of the two options (in this case, removing the potentially noisy instances rather than correcting their class label), is clearly superior for the APRC performance measure. For Learner=RBF, the two levels of Treatment produce statistically identical results, but Remv outperforms Rlbl for all other levels of all other main factors. Based on the AROC response variable, however, there is not clear overall winner. For five of the learners, Remv achieves higher AROC values, for four other learners, Rlbl

	AROC				APRC			
Factor	Maj		Both		Maj		Both	
	Mean	HSD	Mean	HSD	Mean	HSD	Mean	HSD
Learner								
2NN	0.825	A	0.823	B	0.524	A	0.523	A
5NN	0.839	A	0.835	B	0.553	A	0.548	B
C4.5D	0.764	A	0.755	B	0.412	A	0.407	B
C4.5N	0.823	A	0.813	B	0.472	A	0.464	B
LR	0.834	A	0.830	B	0.528	A	0.521	B
MLP	0.831	A	0.828	B	0.550	A	0.547	B
NB	0.811	A	0.809	B	0.456	A	0.457	A
RBF	0.812	A	0.801	B	0.501	A	0.495	B
RF100	0.855	A	0.843	B	0.557	A	0.549	B
RIPPER	0.743	A	0.735	B	0.392	A	0.386	B
SVM	0.823	A	0.820	B	0.523	A	0.519	B
FilterLearner								
NB	0.813	A	0.803	B	0.484	A	0.474	B
RF100	0.816	A	0.814	B	0.510	A	0.511	A
Treatment								
Remv	0.814	A	0.809	B	0.509	A	0.504	B
Rlbl	0.815	A	0.808	B	0.485	A	0.481	B
Threshold								
2%	0.810	A	0.809	A	0.520	A	0.520	A
5%	0.816	A	0.815	A	0.515	A	0.514	A
10%	0.820	A	0.817	B	0.500	A	0.498	B
15%	0.821	A	0.816	B	0.483	A	0.479	B
25%	0.802	A	0.790	B	0.436	A	0.434	B
1.0S	0.818	A	0.809	B	0.497	A	0.489	B
1.5S	0.818	A	0.815	B	0.506	A	0.504	A
2.0S	0.817	A	0.816	B	0.514	A	0.513	A
0.5	0.808	A	0.787	B	0.502	A	0.479	B

Table 7.5: Mean AROC and APRC values and HSD groupings of the two levels of BothClasses for each level of the other main factors for the classification filtering experiments with 31 regular datasets

outperforms Remv, and for the remaining two learners, they perform similarly. Using NB as the learner for filtering, Remv significantly outperforms Rlbl, while when using RF100, Rlbl delivers higher AROC values. For the BothClasses factor, both levels of Treatment perform very similarly, though for Maj, the slight margin by which Remv's AROC values exceed those of Rlbl is statistically significant. Based on Threshold, Rlbl produces significantly higher AROC performance than Remv for all levels except 25% and 0.5 (incidentally the worst performing levels of Threshold).

The HSD groupings of each of the nine levels of Threshold for each level of all the other main factors are given in Table 7.7. As before, the HSD groupings are listed horizontally, comparing each level of the Threshold main factor. Looking vertically down a column, one can see the general level of performance of a single level of Threshold as compared to the other nine levels. For example, 25% and 0.5 are almost always in the lowest group for both AROC and APRC, indicating they are usually outperformed by the other levels of the Threshold main factor. Even more consistent than the poor performance of 25% and 0.5, though, is the constant superiority of Threshold=2% based on the APRC metric, which is in group A for every level of every factor. 2% is followed by 5% and 2.0S for this perfomance metric, always falling into either group A or B. Based on AROC, 10% and 15% perform the best, ending up in group A or B in every row.

Tables 7.3, 7.4, 7.5, 7.6, and 7.7 depict the performance of various combinations of the classification filtering parameters, but they are averaged over several levels of

	AROC				APRC			
Factor	Remv		Rlbl		Remv		Rlbl	
	Mean	HSD	Mean	HSD	Mean	HSD	Mean	HSD
Learner								
2NN	0.831	A	0.817	B	0.554	A	0.493	B
5NN	0.842	A	0.832	B	0.577	A	0.525	B
C4.5D	0.755	B	0.764	A	0.420	A	0.399	B
C4.5N	0.818	A	0.818	A	0.478	A	0.458	B
LR	0.833	A	0.831	B	0.530	A	0.519	B
MLP	0.829	B	0.830	A	0.556	A	0.541	B
NB	0.812	A	0.808	B	0.460	A	0.453	B
RBF	0.800	B	0.812	A	0.498	A	0.497	A
RF100	0.852	A	0.845	B	0.577	A	0.529	B
RIPPER	0.731	B	0.747	A	0.393	A	0.385	B
SVM	0.821	A	0.821	A	0.526	A	0.516	B
FilterLearner								
NB	0.810	A	0.806	B	0.496	A	0.462	B
RF100	0.812	B	0.817	A	0.516	A	0.504	B
BothClasses								
Maj	0.814	B	0.815	A	0.509	A	0.485	B
Both	0.809	A	0.808	A	0.504	A	0.481	B
Threshold								
2%	0.808	B	0.811	A	0.524	A	0.517	B
5%	0.813	B	0.817	A	0.522	A	0.508	B
10%	0.817	B	0.821	A	0.512	A	0.485	B
15%	0.818	B	0.819	A	0.499	A	0.463	B
25%	0.804	A	0.788	B	0.459	A	0.411	B
1.0S	0.813	B	0.814	A	0.506	A	0.480	B
1.5S	0.815	B	0.818	A	0.515	A	0.495	B
2.0S	0.815	B	0.819	A	0.521	A	0.507	B
0.5	0.799	A	0.796	B	0.499	A	0.482	B

Table 7.6: Mean AROC and APRC values and HSD groupings of the two levels of Treatment for each level of the other main factors for the classification filtering experiments with 31 regular datasets

AROC									
Factor	2%	5%	10%	15%	25%	1.0S	1.5S	2.0S	0.5
Learner									
2NN	CD	AB	A	BC	E	ABC	AB	AB	D
5NN	AB	A	A	B	D	B	AB	A	C
C4.5D	E	D	B	A	D	C	C	C	F
C4.5N	AB	AB	AB	B	E	C	AB	A	D
LR	AB	A	AB	BC	E	C	AB	A	D
MLP	D	ABC	A	AB	F	CD	BCD	ABCD	E
NB	A	AB	BC	D	E	CD	ABC	ABC	D
RBF	C	B	AB	A	D	C	B	B	E
RF100	AB	AB	BC	D	F	D	C	A	E
RIPPER	F	E	B	A	B	C	C	D	F
SVM	D	C	AB	A	E	C	BC	C	F
FilterLearner									
NB	C	B	A	B	E	D	B	A	F
RF100	D	C	A	A	F	A	B	C	E
BothClasses									
Maj	D	C	A	A	F	B	B	BC	E
Both	C	B	A	AB	D	C	B	AB	E
Treatment									
Remv	E	CD	A	A	F	D	B	BC	G
Rlbl	D	B	A	B	F	C	B	B	E

APRC									
Factor	2%	5%	10%	15%	25%	1.0S	1.5S	2.0S	0.5
Learner									
2NN	A	AB	D	E	F	D	BC	A	CD
5NN	A	A	C	D	E	C	B	A	C
C4.5D	A	B	D	F	G	DE	C	B	E
C4.5N	A	B	D	F	G	E	C	B	DE
LR	A	AB	CD	EF	G	DE	BC	AB	F
MLP	A	A	B	CD	E	CD	BC	A	D
NB	A	B	E	F	G	DE	CD	BC	CD
RBF	A	AB	C	E	F	D	C	B	DE
RF100	A	B	C	F	G	D	C	B	E
RIPPER	AB	AB	CD	E	F	D	BC	A	E
SVM	AB	A	BC	D	F	CD	AB	A	E
FilterLearner									
NB	A	B	D	F	H	E	C	B	G
RF100	A	BC	E	F	G	D	C	ABC	AB
BothClasses									
Maj	A	B	DE	F	G	E	C	B	D
Both	A	B	D	F	G	E	C	B	F
Treatment									
Remv	A	A	B	D	E	C	B	A	D
Rlbl	A	B	D	F	G	E	C	B	E

Table 7.7: AROC and APRC HSD groupings of the nine levels of Threshold for each level of the other main factors for the classification filtering experiments with 31 regular datasets

the other parameters. Tables 7.8 and 7.9 show the top three parameter schemes for each of the 11 learners based on AROC and APRC, respectively. For AROC, RF100 is almost unanimously a better option for FilterLearner than NB. NB only appears once, in the third ranking spot for 2NN, in which it's used to remove potentially noisy instances from both classes using Threshold=10%. For optimizing AROC, Rlbl is the best level of Treatment for most of the learners. With the exception of all three top spots for 2NN and RF100 and the third spot for SVM, Rlbl produced the top three AROC values for all cases. The BothClasses factor didn't produce a clearly optimal choice. For 2NN, the three top AROC values are all produced using Both. On the other hand, for C4.5N, RF100, and SVM, Maj produced the three top values. For the other seven learners, though, both levels of BothClasses were included in the top three. The Threshold factor was least consistent, with no level getting all three top spots for any of the 11 learners, though 10%, 15%, and 1.0S occurred frequently in the top three.

When optimizing APRC on the 31 unaltered datasets, RF100 was unanimous as the choice for FilterLearner, producing the top three performing filtering schemes for all 11 learners. Both levels of the BothClasses option appeared in the top three for all learners except for 2NN and LR, for which Both occupied all three spots, and MLP, for which Maj produced the three best results. Treatments Remv and Rlbl appeared in the top three APRC achieving schemes for each learner with almost equal frequency. Remv was best all three times for 2NN, and RF100, while Rlbl took

Learner	Rank	FilterLearner	BothClasses	Treatment	Threshold	AROC	APRC
2NN	1	RF100	Both	Remv	15%	0.838	0.564
	2	RF100	Both	Remv	1.0S	0.836	0.567
	3	NB	Both	Remv	10%	0.836	0.559
5NN	1	RF100	Maj	Rlbl	1.0S	0.849	0.559
	2	RF100	Maj	Rlbl	10%	0.849	0.549
	3	RF100	Both	Rlbl	1.5S	0.848	0.569
C4.5D	1	RF100	Maj	Rlbl	15%	0.792	0.382
	2	RF100	Both	Rlbl	15%	0.789	0.390
	3	RF100	Maj	Rlbl	1.0S	0.788	0.415
C4.5N	1	RF100	Maj	Rlbl	1.0S	0.838	0.490
	2	RF100	Maj	Rlbl	10%	0.837	0.486
	3	RF100	Maj	Rlbl	15%	0.837	0.462
LR	1	RF100	Both	Rlbl	1.5S	0.843	0.546
	2	RF100	Both	Rlbl	1.0S	0.842	0.543
	3	RF100	Maj	Rlbl	1.5S	0.842	0.544
MLP	1	RF100	Maj	Rlbl	10%	0.849	0.565
	2	RF100	Maj	Rlbl	15%	0.849	0.557
	3	RF100	Both	Rlbl	10%	0.848	0.568
NB	1	RF100	Maj	Rlbl	5%	0.820	0.483
	2	RF100	Both	Rlbl	5%	0.820	0.483
	3	RF100	Maj	Rlbl	2%	0.820	0.488
RBF	1	RF100	Maj	Rlbl	15%	0.834	0.515
	2	RF100	Both	Rlbl	15%	0.833	0.523
	3	RF100	Both	Rlbl	1.0S	0.830	0.531
RF100	1	RF100	Maj	Remv	1.0S	0.863	0.600
	2	RF100	Maj	Remv	10%	0.863	0.599
	3	RF100	Maj	Remv	15%	0.862	0.588
RIPPER	1	RF100	Maj	Rlbl	15%	0.787	0.396
	2	RF100	Both	Rlbl	15%	0.783	0.395
	3	RF100	Maj	Rlbl	1.0S	0.778	0.421
SVM	1	RF100	Maj	Rlbl	1.0S	0.834	0.538
	2	RF100	Maj	Rlbl	10%	0.834	0.530
	3	RF100	Maj	Remv	15%	0.833	0.538

Table 7.8: Top three classification filtering schemes based on AROC for each Learner from the experiments with 31 regular datasets

all three for RIPPER. As with AROC, there was a lot of variance in the choice of Threshold that produced the top three values for APRC, though 2% and 2.0S show up most often.

Learner	Rank	FilterLearner	BothClasses	Treatment	Threshold	AROC	APRC
2NN	1	RF100	Both	Remv	1.0S	0.836	0.567
	2	RF100	Both	Remv	1.5S	0.835	0.565
	3	RF100	Both	Remv	15%	0.838	0.564
5NN	1	RF100	Both	Remv	2.0S	0.845	0.587
	2	RF100	Maj	Remv	1.5S	0.845	0.586
	3	RF100	Maj	Remv	10%	0.846	0.586
C4.5D	1	RF100	Maj	Rlbl	2%	0.749	0.455
	2	RF100	Both	Rlbl	2%	0.748	0.454
	3	RF100	Both	Remv	2%	0.741	0.454
C4.5N	1	RF100	Maj	Remv	0.5	0.826	0.523
	2	RF100	Both	Remv	2%	0.824	0.522
	3	RF100	Maj	Rlbl	0.5	0.829	0.522
LR	1	RF100	Both	Rlbl	2%	0.840	0.548
	2	RF100	Both	Rlbl	2.0S	0.842	0.547
	3	RF100	Both	Remv	2.0S	0.838	0.547
MLP	1	RF100	Maj	Remv	2%	0.831	0.576
	2	RF100	Maj	Remv	2.0S	0.832	0.575
	3	RF100	Maj	Rlbl	5%	0.846	0.575
NB	1	RF100	Both	Rlbl	2%	0.820	0.489
	2	RF100	Both	Remv	2%	0.819	0.489
	3	RF100	Maj	Rlbl	2%	0.820	0.488
RBF	1	RF100	Maj	Remv	0.5	0.797	0.532
	2	RF100	Both	Rlbl	1.0S	0.830	0.531
	3	RF100	Maj	Rlbl	0.5	0.799	0.530
RF100	1	RF100	Maj	Remv	2%	0.860	0.607
	2	RF100	Maj	Remv	2.0S	0.861	0.605
	3	RF100	Both	Remv	0.5	0.859	0.604
RIPPER	1	RF100	Maj	Rlbl	1.5S	0.762	0.424
	2	RF100	Maj	Rlbl	2.0S	0.749	0.423
	3	RF100	Both	Rlbl	1.5S	0.759	0.423
SVM	1	RF100	Maj	Remv	1.0S	0.831	0.544
	2	RF100	Both	Rlbl	2.0S	0.830	0.544
	3	RF100	Maj	Rlbl	2.0S	0.831	0.543

Table 7.9: Top three classification filtering schemes based on APRC for each Learner from the experiments with 31 regular datasets

		AROC		APRC	
Factor	DF	F-stat	p-value	F-stat	p-value
Learner	10	1929500	<.0001	2028089	<.0001
ClassDistribution	5	1876376	<.0001	2019471	<.0001
NoiseLevel	4	471235	<.0001	665744	<.0001
NoiseDistribution	4	388252	<.0001	169834	<.0001
FilterLearner	1	15160.1	<.0001	38637.8	<.0001
BothClasses	1	19152.2	<.0001	3515.51	<.0001
Treatment	1	24439.5	<.0001	1389080	<.0001
Threshold	8	60647.1	<.0001	173029	<.0001

Table 7.10: ANOVA results showing the significance of each experimental factor based on both the AROC and APRC values for the noise injection experiments

7.3.2 Experiments With Noise Injection

Section 7.3.1 presented the performance results using classification filtering on 31 imbalanced real world datasets with unknown levels of class noise. In this section we present the outcome of another set of experiments in which we use four different imbalanced real world datasets. These four datasets are presumed to be effectively free of noise, and we use sampling and noise injection to control three new experimental factors: percentage of minority instances (ClassDistribution), amount of class noise (NoiseLevel), and percentage of the noisy instances that come from the minority class (NoiseDistribution). Controlling these factors allows us to elucidate the effects of such dataset characteristics on the performance of the various classification filtering options. Table 7.10 presents the ANOVA results for the eight experimental factors for this set of experiments. All eight main factors were found to be statistically significant at the $\alpha = 0.05$ level based on both response variables, AROC and APRC.

Table 7.11 displays the mean AROC and APRC values and corresponding

HSD groupings achieved by each level of every main factor for the noise injection experiments. As before, the results for the single factor Learner are not relevant to the primary issue this study is concerned with, the performance of different combinations of options for classification filtering, and thus are not addressed. The three dataset characteristic factors, ClassDistribution, NoiseLevel, and NoiseDistribution, by themselves are also not particularly germane. Unsurprisingly, higher levels of class imbalance, class noise, and proportion of class noise injected into the more sensitive minority class result in worse classification performance. This pattern is entirely consistent except for in the case of NoiseDistribution=100, which results in better results than NoiseDistribution=75 for APRC (and almost for AROC) because NoiseLevel=50 could not be performed for that level of NoiseDistribution. According to the AROC performance measure, NB performs better than RF100 for the FilterLearner factor, while the opposite is the case for APRC. For both measures, Maj results in better performance than Both for the BothClasses main factor. For Treatment, Rlbl has a slight edge based on AROC, while, based on APRC, Remv performs significantly better. The two best levels of Threshold for the AROC performance metric are 10% and 1.0S, while 25% and 0.5 are the two worst. Based on APRC, 2% and 2.0S achieve the highest average values, while 15% and 25% produce the lowest. Except for the two worst performing learners according to the AROC metric, all differences were statistically significant at the $\alpha = 0.05$ level.

The mean values and HSD groupings, based on both AROC and APRC, of

AROC			APRC		
Factor	Mean	HSD	Factor	APRC	HSD
Learner					
RF100	0.950	A	MLP	0.756	A
MLP	0.946	B	RF100	0.753	B
SVM	0.940	C	5NN	0.752	C
5NN	0.936	D	SVM	0.741	D
LR	0.935	E	LR	0.716	E
NB	0.932	F	NB	0.698	F
2NN	0.906	G	2NN	0.658	G
C4.5N	0.894	H	RBF	0.617	H
RBF	0.869	I	C4.5N	0.565	I
C4.5D	0.783	J	C4.5D	0.436	J
RIPPER	0.783	J	RIPPER	0.430	K
ClassDistribution					
10	0.937	A	10	0.729	A
8	0.932	B	8	0.717	B
6	0.923	C	6	0.697	C
4	0.905	D	4	0.661	D
2	0.866	E	2	0.581	E
1	0.823	F	1	0.499	F
NoiseLevel					
10	0.920	A	10	0.697	A
20	0.912	B	20	0.680	B
30	0.900	C	30	0.656	C
40	0.878	D	40	0.609	D
50	0.872	E	50	0.581	E
NoiseDistribution					
0	0.916	A	0	0.668	A
25	0.910	B	25	0.665	B
50	0.899	C	50	0.648	C
75	0.880	D	100	0.635	D
100	0.879	E	75	0.619	E
FilterLearner					
NB	0.899	A	RF100	0.652	A
RF100	0.896	B	NB	0.642	B
BothClasses					
Maj	0.899	A	Maj	0.649	A
Both	0.896	B	Both	0.646	B
Treatment					
Rlbl	0.900	A	Remv	0.678	A
Remv	0.896	B	Rlbl	0.617	B
Threshold					
10%	0.907	A	2%	0.677	A
1.0S	0.905	B	2.0S	0.671	B
5%	0.904	C	5%	0.670	C
15%	0.903	D	1.5S	0.664	D
1.5S	0.902	E	1.0S	0.654	E
2.0S	0.900	F	0.5	0.647	F
2%	0.894	G	10%	0.646	G
25%	0.886	H	15%	0.623	H
0.5	0.878	I	25%	0.574	I

Table 7.11: Mean AROC and APRC values and HSD groupings of each level of each of the main factors for the classification filtering experiments with noise injection

the two levels of FilterLearner for each level of every other main factor are given in Table 7.12. The HSD results comparing NB and RF100 for each level of every other factor are presented horizontally for each metric. For example, based on AROC, NB was the better choice for FilterLearner, while RF100 performed better than NB using the APRC metric. For every level of Learner except MLP and SVM, according to the APRC response variable, RF100 produces significantly better performance than NB. Based on AROC, the differences for each learner are not as large, and, in this case, NB proves to be the better FilterLearner option for all learners except NB and RF100, as well as RBF, where the two are statistically indistinguishable. According to both AROC and APRC, NB is shown to be the superior level of FilterLearner with more imbalanced class distributions, while RF100 performs better as the imbalance is reduced. RF100 outperforms NB for all five levels of the NoiseLevel factor when measuring APRC, but for the other performance metric, NB produces the better results for all but the lowest level of noise. As with ClassDistribution and NoiseLevel, based on the NoiseDistribution experimental factor, NB performs better as a FilterLearner when the data quality issues are more extreme. For the higher levels of NoiseDistribution (100% and 75% for both AROC and APRC, as well as 50% for AROC alone), NB provides better results, while RF100 performs better for the lower levels. For both levels of the BothClasses factor, FilterLearner=NB results in the higher AROC values, while FilterLearner=RF100 produces better outcomes according to APRC. Based on AROC, NB is the better FilterLearner for both Remv and Rlbl, as well as outperforming RF100 for Remv using APRC, though it is bested for the Rlbl option using this performance measure. The trend of FilterLearner=NB tending to produce better AROC values and RF100 working better based on APRC continues for the Threshold main factor. NB results in superior AROC values for all but 1.0S, 1.5S, and 0.5, while RF100 results in the better APRC values for all levels of Threshold except 25% and 2% where they perform identically.

Table 7.13 presents the mean AROC and APRC values and HSD groupings of both levels of the BothClasses main factor for each level of all the other factors. As in Table 7.12, the HSD groupings comparing Maj and Both are displayed horizontally for each metric. As with the results on the 31 unaltered datasets, BothClasses=Maj results in superior performance in almost every scenario. Though the differences are not very large, the best AROC performance is achieved by Maj for every level of every other main factor. The situation is much the same for the APRC performance measure, though Both does perform better for Learner={C4.5D,RIPPER,SVM}, NoiseLevel=50, NoiseDistribution=0, and Threshold=1.5S.

The mean AROC and APRC values and HSD groupings of the two levels of Treatment for each level of the other main factors is given in Table 7.14. The HSD groupings are listed horizontally, comparing Remv to Rlbl. Based on APRC, Remv is clearly the superior option, outperforming Rlbl for every level of every other factor, often by a relatively large margin. According to the area under the ROC curve, relabeling is the better way to handle the instances identified as noisy for the C4.5D,

	AROC				APRC			
Factor	NB		RF100		NB		RF100	
	Mean	HSD	Mean	HSD	Mean	HSD	Mean	HSD
Learner								
2NN	0.908	A	0.904	B	0.653	B	0.664	A
5NN	0.937	A	0.935	B	0.734	B	0.769	A
C4.5D	0.789	A	0.778	B	0.434	B	0.438	A
C4.5N	0.897	A	0.891	B	0.555	B	0.574	A
LR	0.936	A	0.934	B	0.711	B	0.721	A
MLP	0.947	A	0.945	B	0.761	A	0.752	B
NB	0.930	B	0.933	A	0.693	B	0.702	A
RBF	0.869	A	0.869	A	0.614	B	0.621	A
RF100	0.950	B	0.950	A	0.740	B	0.766	A
RIPPER	0.786	A	0.781	B	0.424	B	0.435	A
SVM	0.944	A	0.935	B	0.747	A	0.734	B
ClassDistribution								
1	0.831	A	0.814	B	0.503	A	0.495	B
2	0.869	A	0.864	B	0.578	B	0.584	A
4	0.905	A	0.905	A	0.654	B	0.668	A
6	0.922	B	0.923	A	0.689	B	0.706	A
8	0.931	B	0.932	A	0.709	B	0.725	A
10	0.937	B	0.937	A	0.721	B	0.737	A
NoiseLevel								
10	0.920	A	0.920	A	0.691	B	0.703	A
20	0.912	A	0.912	B	0.673	B	0.687	A
30	0.902	A	0.899	B	0.649	B	0.662	A
40	0.882	A	0.874	B	0.608	B	0.611	A
50	0.875	A	0.869	B	0.577	B	0.585	A
NoiseDistribution								
0	0.914	B	0.918	A	0.655	B	0.681	A
25	0.909	B	0.912	A	0.653	B	0.676	A
50	0.899	A	0.899	B	0.641	B	0.655	A
75	0.884	A	0.876	B	0.619	A	0.618	B
100	0.888	A	0.870	B	0.643	A	0.628	B
BothClasses								
Maj	0.901	A	0.897	B	0.645	B	0.653	A
Both	0.897	A	0.895	B	0.640	B	0.652	A
Treatment								
Remv	0.898	A	0.893	B	0.680	A	0.676	B
Rlbl	0.900	A	0.899	B	0.605	B	0.629	A
Threshold								
2%	0.894	A	0.894	B	0.677	A	0.677	A
5%	0.904	A	0.903	B	0.669	B	0.671	A
10%	0.908	A	0.905	B	0.644	B	0.649	A
15%	0.909	A	0.898	B	0.622	B	0.623	A
25%	0.903	A	0.869	B	0.584	A	0.564	B
1.0S	0.901	B	0.908	A	0.640	B	0.668	A
1.5S	0.900	B	0.904	A	0.654	B	0.674	A
2.0S	0.900	A	0.900	A	0.666	B	0.677	A
0.5	0.873	B	0.883	A	0.625	B	0.669	A

Table 7.12: Mean AROC and APRC values and HSD groupings of the two levels of FilterLearner for each level of the other main factors for the classification filtering experiments with noise injection

	AROC				APRC			
Factor	Maj		Both		Maj		Both	
	Mean	HSD	Mean	HSD	Mean	HSD	Mean	HSD
Learner								
2NN	0.909	A	0.904	B	0.660	A	0.657	B
5NN	0.938	A	0.934	B	0.756	A	0.748	B
C4.5D	0.784	A	0.782	B	0.435	B	0.437	A
C4.5N	0.896	A	0.892	B	0.566	A	0.563	B
LR	0.936	A	0.933	B	0.720	A	0.712	B
MLP	0.948	A	0.944	B	0.758	A	0.755	B
NB	0.934	A	0.930	B	0.698	A	0.698	A
RBF	0.871	A	0.866	B	0.619	A	0.616	B
RF100	0.953	A	0.947	B	0.757	A	0.748	B
RIPPER	0.784	A	0.782	B	0.429	B	0.430	A
SVM	0.940	A	0.939	B	0.740	B	0.741	A
ClassDistribution								
1	0.826	A	0.820	B	0.502	A	0.496	B
2	0.869	A	0.864	B	0.584	A	0.579	B
4	0.907	A	0.903	B	0.662	A	0.660	B
6	0.924	A	0.921	B	0.698	A	0.697	B
8	0.933	A	0.930	B	0.717	A	0.716	B
10	0.938	A	0.936	B	0.730	A	0.728	B
NoiseLevel								
10	0.923	A	0.917	B	0.701	A	0.694	B
20	0.914	A	0.910	B	0.682	A	0.678	B
30	0.902	A	0.899	B	0.657	A	0.654	B
40	0.880	A	0.877	B	0.610	A	0.608	B
50	0.873	A	0.871	B	0.581	B	0.582	A
NoiseDistribution								
0	0.916	A	0.915	B	0.667	B	0.669	A
25	0.911	A	0.909	B	0.665	A	0.665	A
50	0.900	A	0.898	B	0.649	A	0.647	B
75	0.882	A	0.878	B	0.620	A	0.617	B
100	0.884	A	0.874	B	0.641	A	0.629	B
FilterLearner								
NB	0.901	A	0.897	B	0.645	A	0.640	B
RF100	0.897	A	0.895	B	0.653	A	0.652	B
Treatment								
Remv	0.897	A	0.894	B	0.679	A	0.677	B
Rlbl	0.902	A	0.898	B	0.619	A	0.615	B
Threshold								
2%	0.895	A	0.894	B	0.678	A	0.677	B
5%	0.904	A	0.904	B	0.670	A	0.670	A
10%	0.907	A	0.906	B	0.646	A	0.646	A
15%	0.904	A	0.902	B	0.623	A	0.622	B
25%	0.890	A	0.882	B	0.576	A	0.572	B
1.0S	0.905	A	0.904	B	0.655	A	0.653	B
1.5S	0.903	A	0.901	B	0.664	B	0.665	A
2.0S	0.900	A	0.900	B	0.672	A	0.671	B
0.5	0.886	A	0.870	B	0.656	A	0.638	B

Table 7.13: Mean AROC and APRC values and HSD groupings of the two levels of BothClasses for each level of the other main factors for the classification filtering experiments with noise injection

LR, RBF, and RIPPER learners, while removing them works better for the other seven classifiers. At all six levels of ClassDistribution, Rlbl results in the higher AROC values, as well as for every level of NoiseLevel except for the lowest. Similarly, Remv achieved superior AROC measures for the two lowest levels of NoiseDistribution, while Rlbl was better as more noise came from the minority class. Better AROC measures were achieved by Rlbl for both levels of FilterLearner as well as both levels of BothClasses. Finally, for all levels of Threshold except for 25% and 0.5, Rlbl continued to outperform Remv according to the AROC response variable.

Tables 7.15 and 7.16 show the HSD groupings for the AROC and APRC response variables, respectively, of the nine levels of Threshold for each level of every other main factor. The result for the level of Threshold which results in the best performance for each combination of performance metric and factor level is highlighted in bold. Based on AROC, as with the 31 unaltered datasets from Section 7.3.1, both 25% and 0.5 turn out to be some of the worst choices for Threshold, usually resulting in some of the worst performance regardless of the level of the other factor. Threshold=10% is in HSD group A, using AROC, for five of the six levels of ClassDistribution, four of the five levels of NoiseLevel, four of the five levels of NoiseDistribution, both levels of the BothClasses main factor, both levels of Treatment, and only one level of Learner. 5% does well based on Learner, falling into group A for six different learners, as well as NoiseLevel=10 and NoiseDistribution=100. 15% is in group A for three levels of Learner, NoiseLevel=50, FilterLearner=NB, and Treatment=Remv.

	AROC				APRC			
Factor	Remv		Rlbl		Remv		Rlbl	
	Mean	HSD	Mean	HSD	Mean	HSD	Mean	HSD
Learner								
2NN	0.912	A	0.900	B	0.735	A	0.582	B
5NN	0.941	A	0.930	B	0.803	A	0.701	B
C4.5D	0.759	B	0.808	A	0.453	A	0.419	B
C4.5N	0.898	A	0.890	B	0.594	A	0.535	B
LR	0.934	B	0.936	A	0.722	A	0.710	B
MLP	0.950	A	0.942	B	0.780	A	0.733	B
NB	0.943	A	0.920	B	0.755	A	0.640	B
RBF	0.856	B	0.882	A	0.622	A	0.612	B
RF100	0.956	A	0.944	B	0.803	A	0.703	B
RIPPER	0.760	B	0.807	A	0.437	A	0.422	B
SVM	0.941	A	0.938	B	0.754	A	0.728	B
ClassDistribution								
1	0.822	B	0.823	A	0.542	A	0.457	B
2	0.863	B	0.870	A	0.619	A	0.543	B
4	0.902	B	0.908	A	0.693	A	0.630	B
6	0.920	B	0.925	A	0.725	A	0.670	B
8	0.930	B	0.933	A	0.740	A	0.693	B
10	0.936	B	0.939	A	0.750	A	0.708	B
NoiseLevel								
10	0.921	A	0.919	B	0.737	A	0.658	B
20	0.912	B	0.912	A	0.715	A	0.645	B
30	0.898	B	0.903	A	0.685	A	0.626	B
40	0.873	B	0.883	A	0.632	A	0.587	B
50	0.868	B	0.876	A	0.606	A	0.557	B
NoiseDistribution								
0	0.919	A	0.913	B	0.712	A	0.624	B
25	0.911	A	0.910	B	0.703	A	0.627	B
50	0.897	B	0.901	A	0.678	A	0.618	B
75	0.874	B	0.886	A	0.640	A	0.597	B
100	0.872	B	0.885	A	0.652	A	0.619	B
FilterLearner								
NB	0.898	B	0.900	A	0.680	A	0.605	B
RF100	0.893	B	0.899	A	0.676	A	0.629	B
BothClasses								
Maj	0.897	B	0.902	A	0.679	A	0.619	B
Both	0.894	B	0.898	A	0.677	A	0.615	B
Threshold								
2%	0.890	B	0.899	A	0.682	A	0.672	B
5%	0.898	B	0.910	A	0.691	A	0.649	B
10%	0.902	B	0.911	A	0.685	A	0.607	B
15%	0.902	B	0.904	A	0.675	A	0.570	B
25%	0.895	A	0.876	B	0.645	A	0.502	B
1.0S	0.901	B	0.908	A	0.685	A	0.623	B
1.5S	0.897	B	0.907	A	0.690	A	0.640	B
2.0S	0.895	B	0.905	A	0.687	A	0.656	B
0.5	0.879	A	0.877	B	0.662	A	0.633	B

Table 7.14: Mean AROC and APRC values and HSD groupings of the two levels of Treatment for each level of the other main factors for the classification filtering experiments with noise injection

1.0S is best for one level each from Learner, ClassDistribution, NoiseDistribution, and FilterLearner, and 2% produces the highest AROC values for Learner={NB,RF100}. This is interesting, as the two classifiers that are used in both the Learner and Filter-Learner main factors work best with the 2% threshold, which typically provides the most conservative level of noise detection.

Based on APRC, the best performing level of Threshold is more consistent. 2% performs best for most of the levels of every main factor. RBF, RIPPER, and SVM are the exceptions from main factor Learner. RBF produced the best results with 5% as the Threshold, while RIPPER and SVM had both 5% and 1.5S in group A. 5% also is in group A (while 2% is not) for ClassDistribution={8,10}, NoiseLevel={40,50}, and Treatment=Remv. 1.5S falls into group A for NoiseDistribution={75,100}, and 2.0S is in gropu A for FilterLearner=RF100. 25% performs quite poorly for APRC as well, producing the worst results for every level of every learner. The other two highest percentage-based thresholds, 15% and 10%, also perform poorly using this metric, as does 0.5.

The three combinations of classification filtering parameters that produced the highest AROC values (averaged over all 11 levels of Learner) for each level of the three data quality experimental factors (ClassDistribution, NoiseLevel, and NoiseDistribution) are presented in Table 7.17. The most consistent pattern is the superlative performance of Treatment=Rlbl. For all but four cases (all three top spots for NoiseDistribution=0 and the top spot for NoiseDistribution=25), Rlbl produced the

AROC									
Factor	2%	5%	10%	15%	25%	1.0S	1.5S	2.0S	0.5
Learner									
2NN	E	A	A	D	F	B	BC	C	G
5NN	D	A	B	E	G	D	D	C	F
C4.5D	H	F	B	A	D	C	E	G	I
C4.5N	C	A	D	E	G	C	B	AB	F
LR	C	A	B	D	F	B	C	C	E
MLP	B	A	D	E	G	CD	C	B	F
NB	A	C	G	H	I	E	D	B	F
RBF	G	D	B	A	C	D	E	F	H
RF100	A	A	D	F	H	E	C	B	G
RIPPER	H	F	C	A	B	D	E	G	I
SVM	F	B	B	E	H	A	C	D	G
ClassDistribution									
1	E	B	D	F	H	A	C	D	G
2	F	B	A	C	G	B	D	E	H
4	F	C	A	B	G	C	D	E	H
6	G	C	A	B	F	C	D	E	H
8	G	C	A	B	F	C	D	E	H
10	H	E	A	B	G	C	D	F	I
NoiseLevel									
10	D	A	A	E	G	B	C	C	F
20	F	B	A	D	H	C	D	E	G
30	G	C	A	D	H	B	E	F	I
40	G	D	A	C	H	B	E	F	I
50	G	E	B	A	D	C	E	F	H
NoiseDistribution									
0	E	B	A	B	F	C	D	C	G
25	F	C	A	B	G	C	D	E	H
50	H	D	A	B	G	C	E	F	I
75	F	C	A	B	F	B	D	E	G
100	F	A	B	E	H	A	C	D	G
FilterLearner									
NB	F	B	A	A	C	D	E	E	G
RF100	G	D	B	F	I	A	C	E	H
BothClasses									
Maj	F	C	A	C	G	B	D	E	H
Both	F	B	A	C	G	B	D	E	H
Treatment									
Remv	F	C	A	A	E	B	D	E	G
Rlbl	G	B	A	F	I	C	D	E	H

Table 7.15: AROC HSD groupings of the nine levels of Threshold for each level of the other main factors for the classification filtering experiments with noise injection

APRC									
Factor	2%	5%	10%	15%	25%	1.0S	1.5S	2.0S	0.5
Learner									
2NN	A	C	F	G	H	E	D	B	F
5NN	A	B	F	H	I	E	D	C	G
C4.5D	A	B	E	G	H	D	C	AB	F
C4.5N	A	C	F	G	H	E	D	B	D
LR	A	B	E	G	G	E	D	C	F
MLP	A	C	G	H	I	E	D	B	F
NB	A	C	F	G	H	E	D	B	B
RBF	B	A	C	F	H	E	D	C	G
RF100	A	C	F	G	H	E	D	B	F
RIPPER	C	A	E	G	H	D	A	B	F
SVM	C	A	C	D	F	B	A	B	E
ClassDistribution									
1	A	F	G	H	I	E	D	B	C
2	A	C	F	G	H	D	C	B	E
4	A	B	E	G	H	D	C	B	F
6	A	B	F	H	I	E	D	C	G
8	B	A	E	H	I	F	D	C	G
10	C	A	E	H	I	F	D	B	G
NoiseLevel									
10	A	D	G	H	I	F	E	B	C
20	A	C	G	H	I	F	D	B	E
30	A	C	F	H	I	E	D	B	G
40	C	A	F	G	H	E	D	B	G
50	C	A	D	F	H	E	C	B	G
NoiseDistribution									
0	A	C	F	H	I	G	E	B	D
25	A	C	F	G	H	F	D	B	E
50	A	B	E	G	H	D	C	B	F
75	D	A	E	G	H	C	A	B	F
100	D	C	E	G	H	C	A	B	F
FilterLearner									
NB	A	B	E	H	I	F	D	C	G
RF100	A	C	F	G	H	E	B	A	D
BothClasses									
Maj	A	C	G	H	I	F	D	B	E
Both	A	C	F	H	I	E	D	B	G
Treatment									
Remv	F	A	D	G	I	E	B	C	H
Rlbl	A	C	G	H	I	F	D	B	E

Table 7.16: APRC HSD groupings of the nine levels of Threshold for each level of the other main factors for the classification filtering experiments with noise injection

highest AROC results. The best FilterLearner varied, though the values in all three top spots were always the same except for ClassDistribution=2 and NoiseDistribution={25,75,100}. There was no consistency with the BothClasses factor, as every level of the three data quality factors had both Maj and Both within the top three. For the Threshold factor, 10%, 1.0S, and 15% were consistently in the top three for each data quality factor.

Table 7.18 displays the best three sets of classification filter options for each of the three data quality factors based on the APRC performance metric. In all but three cases (the second and third spots for ClassDistribution=1 and the second spot for NoiseDistribution=0), FilterLearner=NB produced the highest APRC scores. There was still little consistency in the best level of BothClasses, though there were four cases where all three top spots used the same level (NoiseLevel={40,50} and NoiseDistribution={25,75} all had Both for all three top spots). Remv was the best treatment for every level of all three data quality factors. As usual, there wasn't a lot of agreement on the level of Threshold that produces the highest performance values, but 2%, 5%, and 1.5S appeared most frequently.

7.4 Conclusions

Class noise afflicts many real world classification domains, as does class imbalance. Both lead to decreased classification performance, and the effects are even worse when the two data quality factors are combined. Both topics have been explored in

Factor	Rank	FilterLearner	BothClasses	Treatment	Threshold	AROC	APRC
ClassDistribution							
1	1	NB	Maj	Rlbl	15%	0.849	0.413
	2	NB	Both	Rlbl	15%	0.848	0.412
	3	NB	Maj	Rlbl	10%	0.848	0.436
2	1	RF100	Maj	Rlbl	1.0S	0.891	0.578
	2	RF100	Both	Rlbl	1.0S	0.888	0.575
	3	NB	Both	Rlbl	10%	0.887	0.524
4	1	RF100	Maj	Rlbl	1.0S	0.925	0.660
	2	RF100	Maj	Rlbl	10%	0.924	0.633
	3	RF100	Both	Rlbl	1.0S	0.923	0.662
6	1	RF100	Maj	Rlbl	10%	0.940	0.678
	2	RF100	Maj	Rlbl	1.0S	0.939	0.697
	3	RF100	Both	Rlbl	10%	0.939	0.681
8	1	RF100	Maj	Rlbl	10%	0.946	0.704
	2	RF100	Both	Rlbl	10%	0.946	0.707
	3	RF100	Maj	Rlbl	1.0S	0.945	0.716
10	1	RF100	Maj	Rlbl	10%	0.951	0.721
	2	RF100	Both	Rlbl	10%	0.951	0.726
	3	RF100	Maj	Rlbl	1.0S	0.950	0.729
NoiseLevel							
10	1	RF100	Maj	Rlbl	1.0S	0.939	0.691
	2	RF100	Maj	Rlbl	1.5S	0.936	0.708
	3	RF100	Both	Rlbl	1.0S	0.935	0.686
20	1	RF100	Maj	Rlbl	1.0S	0.930	0.678
	2	RF100	Both	Rlbl	1.0S	0.928	0.678
	3	RF100	Maj	Rlbl	1.5S	0.926	0.694
30	1	RF100	Maj	Rlbl	1.0S	0.920	0.657
	2	RF100	Both	Rlbl	1.0S	0.919	0.662
	3	RF100	Maj	Rlbl	10%	0.916	0.627
40	1	NB	Maj	Rlbl	10%	0.902	0.579
	2	NB	Both	Rlbl	10%	0.902	0.580
	3	NB	Maj	Rlbl	15%	0.900	0.546
50	1	NB	Both	Rlbl	10%	0.893	0.552
	2	NB	Maj	Rlbl	10%	0.893	0.550
	3	NB	Both	Rlbl	15%	0.892	0.522
NoiseDistribution							
0	1	NB	Both	Remv	25%	0.926	0.672
	2	NB	Both	Remv	15%	0.925	0.697
	3	NB	Maj	Remv	25%	0.924	0.670
25	1	NB	Both	Remv	25%	0.922	0.670
	2	RF100	Maj	Rlbl	1.0S	0.921	0.643
	3	RF100	Both	Rlbl	1.0S	0.920	0.650
50	1	RF100	Maj	Rlbl	1.0S	0.916	0.647
	2	RF100	Maj	Rlbl	10%	0.915	0.620
	3	RF100	Both	Rlbl	1.0S	0.914	0.648
75	1	RF100	Maj	Rlbl	1.0S	0.905	0.636
	2	NB	Maj	Rlbl	10%	0.904	0.589
	3	NB	Both	Rlbl	10%	0.903	0.588
100	1	RF100	Maj	Rlbl	1.0S	0.917	0.681
	2	RF100	Both	Rlbl	1.0S	0.913	0.671
	3	NB	Maj	Rlbl	10%	0.913	0.609

Table 7.17: Top three classification filtering schemes based on AROC for each level of ClassDistribution, NoiseLevel, and NoiseDistribution from the noise injection experiments

Factor	Rank	FilterLearner	BothClasses	Treatment	Threshold	AROC	APRC
ClassDistribution							
1	1	NB	Both	Remv	2%	0.828	0.562
	2	RF100	Maj	Remv	1.0S	0.833	0.562
	3	RF100	Maj	Remv	2%	0.829	0.562
2	1	NB	Both	Remv	1.5S	0.864	0.646
	2	NB	Both	Remv	5%	0.868	0.635
	3	NB	Maj	Remv	5%	0.868	0.634
4	1	NB	Both	Remv	1.5S	0.897	0.734
	2	NB	Both	Remv	5%	0.905	0.708
	3	NB	Maj	Remv	5%	0.905	0.707
6	1	NB	Both	Remv	1.5S	0.916	0.762
	2	NB	Both	Remv	5%	0.923	0.741
	3	NB	Maj	Remv	5%	0.923	0.740
8	1	NB	Both	Remv	1.5S	0.926	0.776
	2	NB	Both	Remv	5%	0.932	0.757
	3	NB	Maj	Remv	5%	0.931	0.755
10	1	NB	Both	Remv	5%	0.936	0.765
	2	NB	Maj	Remv	5%	0.936	0.764
	3	NB	Maj	Remv	2.0S	0.936	0.761
NoiseLevel							
10	1	NB	Both	Remv	1.5S	0.917	0.759
	2	NB	Both	Remv	2%	0.921	0.757
	3	NB	Maj	Remv	2%	0.921	0.756
20	1	NB	Both	Remv	1.5S	0.911	0.737
	2	NB	Both	Remv	5%	0.915	0.731
	3	NB	Maj	Remv	5%	0.915	0.731
30	1	NB	Both	Remv	1.5S	0.899	0.707
	2	NB	Both	Remv	5%	0.901	0.701
	3	NB	Maj	Remv	5%	0.901	0.701
40	1	NB	Both	Remv	1.5S	0.874	0.665
	2	NB	Both	Remv	5%	0.878	0.650
	3	NB	Both	Remv	10%	0.886	0.650
50	1	NB	Both	Remv	1.5S	0.870	0.644
	2	NB	Both	Remv	10%	0.881	0.623
	3	NB	Both	Remv	5%	0.872	0.622
NoiseDistribution							
0	1	NB	Both	Remv	2%	0.917	0.731
	2	RF100	Both	Remv	2%	0.916	0.730
	3	NB	Maj	Remv	2%	0.916	0.730
25	1	NB	Both	Remv	1.5S	0.909	0.723
	2	NB	Both	Remv	5%	0.913	0.716
	3	NB	Both	Remv	2%	0.907	0.715
50	1	NB	Both	Remv	1.5S	0.897	0.704
	2	NB	Both	Remv	5%	0.900	0.695
	3	NB	Maj	Remv	5%	0.899	0.691
75	1	NB	Both	Remv	1.5S	0.878	0.678
	2	NB	Both	Remv	10%	0.890	0.665
	3	NB	Both	Remv	15%	0.894	0.661
100	1	NB	Both	Remv	1.5S	0.870	0.692
	2	NB	Maj	Remv	15%	0.900	0.687
	3	NB	Maj	Remv	10%	0.894	0.686

Table 7.18: Top three classification filtering schemes based on APRC for each level of ClassDistribution, NoiseLevel, and NoiseDistribution from the noise injection experiments

research, but relatively little work has examined both of them together. Classification filtering is a data preprocessing technique where a classifier is built and evaluated on the training data (typically with cross-validation). Any training instances misclassified during this process are considered noisy and dealt with either by removing them or relabeling them. Different studies have examined various ways of performing this technique, though none, to our knowledge, have performed an integrated study combining all the different options and their combinations.

When utilizing a classification filter, the practitioner can choose different learners (or ensembles thereof) to use to perform the filtering, whether to remove the examples identified as potentially noisy or relabel them, whether to remove noisy instances from all classes or to spare the minority class instances, and how to determine a threshold on declaring an instance as potentially noisy. One might even think about how the choice of classifier to perform prediction (after the classification filtering is performed) affects the selection of these choices. Previous studies have usually addressed only one of these issues at a time. In this work we performed an extremely thorough, comprehensive, demanding set of empirical experiments to examine all of these factors affecting classification filter performance, as well as how the performance varies as these factors interact with data factors such as class distribution, noise level, and distribution of noise among the classes. Our general conclusions based on this exhaustive experimentation include the following.

1. The best learner to use to perform the classification filtering depends on both

performance measure and data quality. Based on the APRC performance measure, RF100 produced the best results for all other levels of every other factor for the 31 unaltered datasets and for the majority of other factor levels for the noise injection experiments. Based on AROC, RF100 again was generally superior for the 31 regular datasets, but for the scenarios in the noise injection experiments with more extreme data quality issues (more imbalanced or noisier data), NB was the better choice for FilterLearner. NB also achieved higher AROC values than RF100 for both levels of Treatment, six out of nine levels of Threshold, and eight out of 11 levels of Learner on the noise injection experiments.

2. Perhaps unsurprisingly, treating only the potentially noisy instances from the majority class leads to better classification performance than if instances from both classes are removed or relabeled. This was the case based on both AROC and APRC, unanimously for the 31 regular datasets and with limited exception for the noise injection experiments.

3. Using the APRC performance metric, removing the noisy instances is clearly better than relabeling them. For each level of every other main factor in both sets of experiments, Remv resulted in superior or similar performance to Rlbl. For the noise injection experiments, the three best performing combinations of parameters (in terms of APRC) for each level of ClassDistribution, NoiseLevel,

and NoiseDistribution all used Remv. According to the AROC measure, however, the results are less clear. Relabeling results in better AROC values for many of the levels of the other factors for the 31 regular datasets as well as the noise injection experiments. Also, the three best parameter combinations (based on AROC) for each level of Learner for the 31 unaltered datasets and each level of the data quality factors for the noise injection experiments included Rlbl in the vast majority of cases.

4. The ideal strategy for determining how to determine the cutoff for declaring instances as potentially noisy was not clear, however, it was consistently observed that it doesn't pay to be liberal and flag too many instances. For both performance measures, considering 25% of the instances in a class as noisy almost always resulted in the worst performance of all nine thresholding strategies tested. In addition, the default decision threshold (0.5 in the case of binary class data), which appears to be most often used when performing classification filtering, also resulted in some of the worst performance in almost all cases. Based on AROC, marking 10% and 15% of the instances of a class as noisy performed best most consistently, while for APRC, flagging 2% produced the highest level of success.

As with many problems in machine learning, the answer to the question "What technique will work best?" in regards to classification filtering, specifically with imbalanced data, is "It depends." There were several consistent patterns, though, that allow recommendations to be made. When performing classification filtering, noise detection should be conservative. The most agressive strategies resulted consistently in the worst classification performance. It was also clear that it is unwise to remove or relabel instances from the minority class for imbalanced data, which almost always reduced performance compared to only treating instances of the majority class. If area under the PRC curve is the performance metric of interest, removing instances flagged as noisy (as opposed to relabeling them) and using the random forest learner to identify the noise (as opposed to Naive Bayes) can be recommended due to their consistently superlative performance, though based on area under the ROC curve, there wasn't a clear choice of FilterLearner or Treatment that produced reliably superior results. Interestingly, the two learners used for performing classification filtering produced the best results for classification when the filtering threshold was most conservative. Perhaps this indicates that the learner being used to build the final classifier does not work as well simultaneously for noise filtering to identify the instances that would impair classification performance.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

Results from the use of machine learning algorithms can only be as good as the data used to train them. In many real world data mining applications, the presence of data quality issues erects obstacles to the optimal performance of the predictive models produced through machine learning. Supervised learning techniques are often led astray by erroneous values in the independent attributes (attribute noise), or, more critically, incorrect values in the dependent attribute (class noise). Class rarity is another dataset characteristic that can be extremely detrimental to classification performance and commonly occurs in many important real world applications. Due to a disproportionately small representation in the data, classification algorithms tend to put little importance on accurately identifying instances of such a minority class, though this is often the class that it is most important to detect.

Class noise and class imbalance combine to make accurate classifier induction even more difficult. Noisy instances either pollute the fragile minority class with instances that will further confuse the classifier, or reduce the numbers of the class which is already rare in the data and thus lacking in attention. While there has been considerable empirical research on both the topics of class noise and class imbalance separately, much of it has not been very comprehensive or statistically well substantiated. Such robust examination of classification in the presence of both data quality issues is even more rare. The objectives and contributions of this work are centered on thorough empirical examination of techniques for improving classification performance on data that is both imbalanced and noisy.

Chapter 4 presented the first through empirical examination of the efficiency of different learners for performing classification filtering. Classification filtering involves training and evaluating a classifier on the training dataset (typically using cross-validation) and marking any misclassified instances as noisy. In almost all reported applications of this technique, the default decision threshold is used for determining the misclassified instances, however most learners can report a numeric measure indicating their confidence in their prediction. Adjusting the threshold on this confidence measure can allow finer control over how many instances to detect as noisy, however the prevailing strategy has been to simply use the default classification. In the experiments in Chapter 4, eight different classifiers were used to perform classification filtering on imbalanced data with injected noise, and the numbers of falsely and correctly detected noisy instances were recorded at varying decision thresholds so that ROC curves could be created showing the tradeoff between hit rate and false alarm rate when detecting noise.

The experiments showed that of the eight classifiers, the random forest and Naive Bayes algorithms clearly were the most efficient at accurately identifying noisy instances. Overall, those two produced results that were fairly similar, though they each stood out somewhat under certain circumstances. RF100 was superior at identifying majority instances mislabeled as belonging to the minority class, while NB performed better at the opposite task of identifying the mislabeled minority instances. At more extreme levels of data quality degradation, NB tended to identify noisy instances more accurately than RF100. When the data was highly imbalanced, highly noisy, or had most of its noise injected into the minority class (which is more sensitive to noise), NB produced better results than RF100. When the data quality issues were less severe, however, RF100 outperformed NB. On the other hand, the RBF neural network learner produced noise detection results that were by far the worst of any of the classifiers tested.

Performing classification filtering to remove instances only of the majority class has been performed to improve performance on imbalanced data [6], however the amount of data removed tends to be too little to greatly improve performance on the minority class. In Chapter 5, a new way of intelligently undersampling majority class instances is proposed to maintain the noise removal capacity of the prevailing techniques while adding the ability to control how much to balance the class distribution. Called Filtered Undersampling (FUS), the technique performs classification filtering only on the majority class, however, instead of using the default decision threshold for removing instances, the threshold is set such that the noisest seeming majority instances are removed up until the desired class distribution is achieved. This new data sampling technique is tested using a number of classification algorithms for both filtering and prediction and compared with other common and successful sampling techniques for imbalanced data.

The Filtered Undersampling technique proposed was not overwhelmingly successful at improving classifier performance for noisy, imbalanced data, however it did perform well under some circumstances. It was tested at two sampling levels, 35% and 50% (where the percentage represents the post-sampling minority class percentage), and 35% was unanimously found to produce superior results. The superior performace of less extreme classification filtering ties into results from the experiments presented in Chapter 7. Despite ensemble classification filters often being touted in previous work [11, 49], the ensemble classifier performed quite poorly with FUS. The only learner that did produce impressive results with FUS was RF100, which was significantly better than all the other classifiers used with FUS. Based on the area under the ROC curve performance measure, FUS with RF100 was tied (statistically) for the best performance overall on 31 regular imbalanced datasets (without artificial modification), and was the second best performer for the experiments involving controlled levels of class imbalance, class noise, and class noise distribution. When examining its performance relative to the other sampling techniques tested, FUS with RF100 produced better results as the data quality deteriorated. It performed best at the highest levels of class imbalance, class noise, and amount of noise injected into the minority class. According to the area under the PRC curve, however, all versions of FUS performed quite poorly. Even using RF100, which greatly outperformed the other classifiers used for FUS, the APRC results were disappointing. Ultimately, the results showed that when compared to the other sampling techniques, given the performance of FUS (even using RF100) and the complexity involved in its application, it might be more profitable to use simpler sampling techniques to improve classification performance. The exception, though, might be with extremely low quality data, which FUS with RF100 seemed to excel at improving according to the AROC performance measure.

Ensemble classification methods have been very successful for improving the performance of the base learners, especially boosting techniques [26]. Boosting algorithms have been proposed aimed specifically at the class noise problem, as well as at the problem of imbalanced data. None seem to target both, however. Chapter 6 presents the results of extensive experimentation to investigate the performance of two boosting algorithms targeting class noise, two boosting algorithms targeting class imbalance, and six newly proposed boosting algorithms which combine mechanisms from both areas to address data that is both noisy and imbalanced. These 10 boosting methods are employed with four different base learners on both unaltered real world data as well as data that has been artificially injected with class noise to determine which techniques employ the best strategies for improving performance in the wake of such data quality issues.

Experiments with the 10 boosting algorithms showed, most distinctly, that being robust to noise and not accounting for class imbalance can be extremely detrimental to performance on imbalanced data. The two boosting algorithms tested that targeted class noise only and didn't account for class imbalance, AveBoost2 and ORBoost, produced the worst results, even for the highest levels of class noise. When the noise handling mechanisms were combined with boosting methods meant to combat class imbalance, good results were produced. The combination of AveBoost2 and RUSBoost and SMOTEBoost was very successful. AveSMOTEBoost often performed well, especially using the APRC performance measure, and AveRUSBoost often produced the best results of any of the boosting algorithms. ORBoost and RUSBoost also combined well, as ORRUSBoost also produced some of the best results. ORSMOTEBoost, though, often ranked quite poorly compared to all techniques except plain AveBoost2 and ORBoost. The last two hybrid boosting algorithms tested added a classification filtering step using Naive Bayes to the RUSBoost and SMOTEBoost algorithms. While the results produced by these versions of boosting weren't bad, they generally performed quite similarly to RUSBoost and SMOTEBoost alone, so there didn't seem to be any advantage to adding the noise cleansing step.

Finally, Chapter 7 presents a comprehensive empirical examination of classification filtering and its several parameters. While previous studies of the technique have often used only a single parameterization of classification filtering, in the study in this Chapter we examine four important options for classification filtering: which classifier to use for noise detection, which class(es) to detect noise from, what to do with instances detected as noisy, and what strategy to use to set the cutoff between noisy and clean instances. In addition to these four experimental factors, we employ four other factors to determine classification filtering performance under different conditions. Each combination of the four filtering options is tested with different learners (for building the predictive classifier after the classification filter preprocessing step), different class distributions, different levels of data noise, and different distributions of noise among the classes in the data.

These exhaustive experiments with classification filtering showed that there is no unanimous configuration to guarantee optimal performance. The ideal values for several parameters to use will vary based on predictive learner used, qualities of the data (class imbalance, noise level, etc.), and what performance metric one is trying to optimize. For several of the options, though, the results support some recommendations. Most prominently, superior results were almost always achieved, based on both AROC and APRC, when only majority instances were detected than if instances of both classes were dealt with. Other results based on APRC tended to be more consistent than with AROC. Using RF100 as the classifier for performing the filtering almost always worked better than using NB for this performance measure. Removing noisy instances unanimously produced higher APRC values than relabeling them. Based on AROC, FilterLearner=NB produced better results than RF100 the majority of the time, especially at the more severe levels of class imbalance and class noise. Relabeling noisy instances followed a similar pattern, producing higher AROC values than the noisy instance removal option in a majority of cases, including in the presence of reduced data quality. The ideal threshold for noise detection was fairly variable, though there were clear trends based on performance metric. For AROC, Threshold={10%,15%,2.0S} performed the best, while the more conservative values, Threshold={2%,2.0S,5%}, produced better APRC results. For both performance metrics, the 25% Threshold produced the worst performance, overall, while more conservative amounts of noise detection performed better. This suggests that performing classification filtering too agressively harms performance, which agrees with the unspectacular FUS results from Chapter 5, as Filtered Undersampling is basically an agressive application of classification filtering, usually even more so than Threshold=25%, depending on class distribution.

8.2 Future Work

A number of new research questions are evoked from the findings produced in this work. The following topics require more investigation to more thoroughly understand the interaction between class noise and class imbalance, and discern what techniques are most able to improve classification performance in the wake of these data quality issues.

1. Classification filtering was utilized frequently in the experiments performed for

this work. Multiple different learners were utilized to perform this noise filtering, including an ensemble of classifiers. RF100 and NB turned out to be the most effective learners of those tested, but other than the one classifier ensemble, only standard learning algorithms were used. Using learning schemes more conducive to successful classification of imbalanced data, such as RUSBoost or cost sensitive classifiers, as FilterLearners might produce significantly better noise detection results.

2. Throughout each of the chapters, sampling and noise injection were utilized to examine the effects of class imbalance, noise level, and noise distribution on the classification techniques being utilized for this noisy, imbalance data. In many cases, one technique or option would perform better than others at more severe levels of imbalance and/or noise. In Chapter 4, RF100 often more accurately identified mislabeled isntances at moderate levels of noise and imbalance, however, at more extreme levels, NB worked better. Also in Chapter 7, NB outperformed RF100 in how much it improved classifier performance when used as part of a classification filter preprocessing technique. The FUS technique using RF100 was also most successful compared to its competitors at the extremes of these data quality factors. One is left to wonder if the performance advantages grow linearly as the corresponding data quality factors change. That is, if class distribution were 0.1% or there was a larger quantity of class noise, would NB's advantage over RF100 in classification filtering increase? Would RF100 have an

even bigger advantage if noise level and class imbalance was even more conservative? If data were extremely noisy and imbalanced, would FUS using RF100 clearly outperform the more conventional data sampling techniques?

3. The classification filtering investigation in Chapter 7 showed that when the same learner was used for both filtering and classification (NB and RF100), the best performing option for the Threshold factor was 2%, a very conservative value. This seems to suggest that, while noisy instances might be removed, other instances that are more crucial to learning the class but are difficult for the learner to accurately classify are being removed (or relabeled) also, leading actually to reduced classifier performance as the learner only tries to learn the 'easy' parts of the feature space. More work would be necessary to try to determine if this is indeed the case. Perhaps one could also develop a method for removing harmful (i.e. noisy instances) without removing instances that are important to defining the decision boundaries.

4. In the boosting experiments in Chapter 6, the two boosting techniques which target only class noise, AveBoost2 and ORBoost, perform significantly worse than all eight of the other boosting variants tested. The performance difference could be due simply to a lack of a method for accounting for class imbalance. Alternatively, it could be that the noise-robustness functions in these two boosting techniques actually harm performance on the minority class since rare class

instances can appear similar to mislabeled majority instances. Examination of the accuracy and weight changes for the minority class instances throughout the boosting iterations for AveBoost2 and ORBoost could elucidate whether this is the case or not.

BIBLIOGRAPHY

[1] The ELENA Project.
http://www.dice.ucl.ac.be/neural-nets/Research/Projects/ELENA/databases/REAL/phoneme/.

[2] J. Aczel and J. Daroczy. *On measures of information and their characterizations.* Academic Press, New York, NY, 1975.

[3] D. W. Aha. *Lazy learning.* Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[4] W. Altidor, T. M. Khoshgoftaar, and A. Napolitano. Wrapper-based feature ranking for software engineering metrics. In *Proceedings of the IEEE International Conference on Machine Learning and Applications*, Miami, FL, USA, December 2009. In Press.

[5] D. Anyfantis, M. Karagiannopoulos, S. Kotsiantis, and P. Pintelas. *Robustness of learning techniques in handling class noise in imbalanced datasets*, volume 247 of *IFIP International Federation for Information Processing.* Springer, Boston, MA, artificial intelligence and innovations 2007: from theory to applications edition, 2007.

[6] R. Barandela, R. M. Valdovinos, J. S. Sanchez, and F. J. Ferri. The imbalanced training sample problem: Under or over sampling? *In Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition (SSPR/SPR'04), Lecture Notes in Computer Science 3138*, pages 806–814, 2004.

[7] M. L. Berenson, D. M. Levine, and M. Goldstein. *Intermediate Statistical Methods and Applications: A Computer Package Approach.* Prentice-Hall, Inc., 1983.

[8] C. Blake and C. Merz. UCI repository of machine learning databases. *http://www.ics.uci.edu/ mlearn/ MLRepository.html*, 1998. Department of Information and Computer Sciences, University of California, Irvine.

[9] L. Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.

[10] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[11] C. E. Brodley and M. A. Friedl. Identifying and eliminating mislabeled training instances. In *Proceedings of 13th National Conference on Artificial Intelligence*, pages 799–805. AAAI Press, 1996.

[12] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[13] N. V. Chawla, D. A. Cieslak, L. O. Hall, and A. Joshi. Automatically countering imbalance and its empirical relationship to cost. *Data Min. Knowl. Discov.*, 17(2):225–252, 2008.

[14] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. SMOTE: Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, (16):321–357, 2002.

[15] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. Bowyer. SMOTEBoost: Improving prediction of the minority class in boosting. In *In Proceedings of Principles of Knowledge Discovery in Databases*, 2003.

[16] W. W. Cohen. Fast effective rule induction. In *In Proc. 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[17] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *ICML '06: Proceedings of the 23rd international conference on machine learning*, pages 233–240, Pittsburgh, Pennsylvania, 2006. ACM.

[18] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Knowledge Discovery and Data Mining*, pages 155–164, 1999.

[19] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[20] C. Drummond and R. C. Holte. C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on Learning from Imbalanced Data Sets II, International Conference on Machine Learning*, 2003.

[21] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 239–246, 2001.

[22] W. Fan, S. J. Stolfo, J. Zhang, and P. K. Chan. AdaCost: misclassification cost-sensitive boosting. In *Proc. 16th International Conf. on Machine Learning*, pages 97–105. Morgan Kaufmann, San Francisco, CA, 1999.

[23] T. Fawcett and F. Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, (1):291–316, 1997.

[24] A. Folleco, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano. Identifying learners robust to low quality data. *International Journal of Informatica*, 2010. In Press.

[25] A. Folleco, T. M. Khoshgoftaar, and A. Napolitano. Comparison of four performance metrics for evaluating sampling techniques for low quality class-imbalanced data. In *Proceedings of the IEEE International Conference on Machine Learning and Applications*, page 153158, San Diego, CA, USA, December 2008.

[26] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, 1996.

[27] J. Furnkranz and G. Widmer. Incremental reduced error pruning. In *International Conference on Machine Learning*, pages 70–77, 1994.

[28] D. Gamberger, N. Lavrač, and C. Grošelj. Experiments with noise filtering in a medical domain. In *Proceedings of the 16th International Conference on Machine Learning*, pages 143–151, San Francisco, CA, 1999. Morgan Kaufmann.

[29] K. Gao, T. M. Khoshgoftaar, and A. Napolitano. Exploring software quality classification with a wrapper-based feature ranking technique. In *Proceedings*

of the 21st IEEE International Conference on Tools with Artificial Intelligence, Newark, NJ, USA, November 2009. In Press.

[30] H. Han, W. Y. Wang, and B. H. Mao. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In *In International Conference on Intelligent Computing (ICIC'05). Lecture Notes in Computer Science 3644*, pages 878–887. Springer-Verlag, 2005.

[31] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer, 2001.

[32] S. Hido and H. Kashima. Roughly balanced bagging for imbalanced data. In *8th SIAM International Conference on Data Mining*, pages 143–152. SIAM, 2008.

[33] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[34] Y.-M. Huang, C.-M. Hung, and H. C. Jiau. Evaluation of neural networks and data mining methods on a credit assessment task for class imbalance problem. *Nonlinear Analysis: Real World Applications*, 7(4):720–747, 2006.

[35] J. V. Hulse, T. M. Khoshgoftaar, and A. Napolitano. Software reliability modeling using skewed measurement data. In *Proceedings of the 13th ISSAT International Conference on Reliability and Quality in Design*, page 181185, Seattle, WA, USA, August 2007.

[36] J. V. Hulse, T. M. Khoshgoftaar, and A. Napolitano. An empirical comparison of repetitive undersampling techniques. In *Proceedings of the IEEE International Conference on Information Reuse and Integration IRI09*, pages 29–34, Las Vegas, NV, USA, August 2009.

[37] J. V. Hulse, T. M. Khoshgoftaar, A. Napolitano, and R. Wald. Feature selection with high-dimensional imbalanced data. In *Proceedings of the IEEE International Conference on Data Mining - Workshops (ICDMW09)*, Miami, FL, USA, December 2009. In Press.

[38] B. Irie and S. Miyake. Capabilities of three-layered perceptrons. In *IEEE International Conference on Neural Networks*, pages 641–648, 1988.

[39] N. Japkowicz. Learning from imbalanced data sets: a comparison of various strategies. In *AAAI Workshop on Learning from Imbalanced Data Sets (AAAI'00)*, pages 10–15, 2000.

[40] N. Japkowicz and S. Stephan. The class imbalance problem: a systematic study. *Intelligent Data Analysis*, 6(5):429–450, 2002.

[41] W. Jiang. Some theoretical aspects of boosting in the presence of noisy data. In *Proceedings of The Eighteenth International Conference on Machine Learning (ICML-2001)*, pages 234–241. Morgan Kaufmann, 2001.

[42] T. Jo and N. Japkowicz. Class imbalances versus small disjuncts. *SIGKDD Explorations*, 6(1):40–49, 2004.

[43] M. V. Joshi, V. Kumar, and R. C. Agarwal. Evaluating boosting algorithms to classify rare classes: Comparison and improvements. In *In Proceedings of IEEE International Conference on Data Mining*, pages 257–264, November 2001.

[44] P. Kang and S. Cho. Eus svms: Ensemble of under-sampled svms for data imbalance problems. In *13th International Conference on Neural Information Processing, Part I*, volume 4232 of *Lecture Notes in Computer Science*, Hong Kong, China, 2006. Springer.

[45] A. Karmaker and S. Kwek. A boosting approach to remove class label noise. In *HIS '05: Proceedings of the Fifth International Conference on Hybrid Intelligent Systems*, pages 206–211, Washington, DC, USA, 2005. IEEE Computer Society.

[46] T. M. Khoshgoftaar, E. B. Allen, and J. Deng. Using regression trees to classify fault-prone software modules. *IEEE Trans. Reliability*, 51(4):455–462, 2002.

[47] T. M. Khoshgoftaar, L. A. Bullard, and K. Gao. Detecting outliers using rule-based modeling for improving CBR-based software quality classification models. In K. D. Ashley and D. G. Bridge, editors, *Proceedings of the 16th*

International Conference on Case-Based Reasoning, volume 1689, pages 216–230. Springer-Verlag LNAI, 2003.

[48] T. M. Khoshgoftaar, C. Seiffert, J. Van Hulse, A. Napolitano, and A. Folleco. Learning with limited minority class data. In *ICMLA '07: Proceedings of the Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 348–353, Cincinati, OH, USA, 2007. IEEE Computer Society.

[49] T. M. Khoshgoftaar, S. Zhong, and V. Joshi. Enhancing software quality estimation using ensemble-classifier based noise filtering. *Intelligent Data Analysis: An International Journal*, 6(1):3–27, 2005.

[50] M. Kubat and S. Matwin. Addressing the curse of imbalanced training sets: One sided selection. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, 1997.

[51] T. Landgrebe, P. Paclik, and R. Duin. Precision-recall operating characteristic (p-roc) curves in imprecise environments. *18th International Conference on Pattern Recognition*, 4:123–127, 2006.

[52] X.-Y. Liu, J. Wu, and Z.-H. Zhou. Exploratory under-sampling for class-imbalance learning. In *Proceedings of the Sixth International Conference on Data Mining*, pages 965–969, Hong Kong, China, 2006. IEEE Computer Society.

[53] M. Maloof. Learning when data sets are imbalanced and when costs are unequal and unknown. In *Proceedings of the ICML'03 Workshop on Learning from Imbalanced Data Sets*, 2003.

[54] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, NY, 1997.

[55] M. Molinara, M. T. Ricamato, and F. Tortorella. Facing imbalanced classes through aggregation of classifiers. In *Proceedings of the 14th International Conference on Image Analysis and Processing*, pages 43–48, Modena, Italy, 2007. IEEE Computer Society.

[56] J. Moody and C. J. Darken. Fast learning in networks of locally tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[57] A. Napolitano, T. M. Khoshgoftaar, and J. Van Hulse. A comparative study of data sampling methods for alleviating class imbalance. *Intelligent Data Analysis: An International Journal*, 2010. In Press.

[58] N. C. Oza. Aveboost2: Boosting for noisy data. In J. K. Fabio Roli and T. Windeatt, editors, *Fifth International Workshop on Multiple Classifier Systems*, pages 31–40, Cagliari, Italy, June 2004. Springer-Verlag.

[59] F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42:203–231, 2001.

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

[61] J. R. Quinlan. *C4.5: Programs For Machine Learning.* Morgan Kaufmann, San Mateo, California, 1993.

[62] B. D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University Press, Cambridge, UK, 1996.

[63] SAS Institute. *SAS/STAT User's Guide.* SAS Institute Inc., 2004.

[64] J. L. Schafer. *Analysis of Incomplete Multivariate Data.* Chapman and Hall/CRC, 2000.

[65] B. Schlkopf, C. J. C. Burges, and A. J. Smola, editors. *Advances in Kernel Methods: Support Vector Learning.* MIT Press, Cambridge, Massachusetts, 1999.

[66] C. Seiffert, T. Khoshgoftaar, J. Van Hulse, and A. Napolitano. A comparative study of data sampling and cost sensitive learning. In *Data Mining Workshops, 2008. ICDMW '08. IEEE International Conference on*, pages 46–52, Dec. 2008.

[67] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano. Mining data with rare events. In *Proceedings of the Nineteenth IEEE International Conference on Tools with Artificial Intelligence*, page 132139, Patras, Greece, October 2007.

[68] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano. Improving learner performance with data sampling and boosting. In *Proceedings of the* 20^{th} *IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2008)*, pages 452–459, Dayton, OH, November 2008.

[69] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano. Resampling or reweighting: A comparison of boosting implementations. In *ICTAI '08: Proceedings of the 2008 20th IEEE International Conference on Tools with Artificial Intelligence*, pages 445–451, Washington, DC, USA, 2008. IEEE Computer Society.

[70] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse, and A. Napolitano. RUSBoost: A hybrid approach to alleviating class imbalance. *IEEE Transactions on Systems, Man & Cybernetics: Part A: Systems and Humans*, 2010. In Press.

[71] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. Building useful models from imbalanced data with sampling and boosting. In *Proceedings of the* 21^{st} *International Florida Artificial Intelligence Research Society Conference (FLAIRS-2008)*, pages 306–311, Coconut Grove, Florida, USA, May 2008.

[72] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano. RUSBoost: Improving classification performance when training data is skewed. In *The 19th IEEE International Conference on Pattern Recognition*, Tampa, FL, USA, December 2008.

[73] Y. Sun, M. S. Kamel, A. K. C. Wong, and Y. Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognition*, 40(12):3358–3378, 2007.

[74] C. M. Teng. Correcting noisy data. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 239–248, 1999.

[75] K. M. Ting. A comparative study of cost-sensitive boosting algorithms. In *Proc. 17th International Conf. on Machine Learning*, pages 983–990. Morgan Kaufmann, San Francisco, CA, 2000.

[76] I. Tomek. Two modifications of CNN. *IEEE Transactions on Systems Man and Communications SMC-6*, pages 769–772, 1976.

[77] J. Van Hulse and T. M. Khoshgoftaar. Class noise detection using frequent itemsets. *Intelligent Data Analysis: An International Journal*, 10(6):487–507, December 2006.

[78] J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano. Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th International Conference on Machine Learning (ICML 2007)*, pages 935–942, Corvallis, OR, USA, June 2007.

[79] J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano. Skewed class distributions and mislabeled examples. In *Proceedings of the IEEE International Conference on Data Mining - Workshops (ICDMW'07)*, pages 477–482, Omaha, NE, USA, October 2007. IEEE Computer Society.

[80] J. Van Hulse, T. M. Khoshgoftaar, C. Seiffert, and L. Zhao. The multiple imputation quantitative noise corrector. *Intelligent Data Analysis: An International Journal*, 11(3):254–263, 2007.

[81] J. D. Van Hulse, T. M. Khoshgoftaar, and H. Huang. The pairwise attribute noise detection algorithm. *Knowledge and Information Systems Journal, Special Issue on Mining Low Quality Data*, 11(2):171–190, 2007.

[82] S. Viaene and G. Dedene. Cost-sensitive learning and decision making revisited. *European Journal of Operational Research*, 127(1):212–220, October 2005.

[83] G. Weiss, K. McCarthy, and B. Zabar. Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? *Machine Learning Journal*, 65(1):95–130, 2006.

[84] G. M. Weiss. Mining with rarity: A unifying framework. *SIGKDD Explorations*, 6(1):7–19, 2004.

[85] G. M. Weiss and F. Provost. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, (19):315–354, 2003.

[86] D. Wilson. Asymptotic properties of nearest neighbor rules using edited data sets. *IEEE Trans. on Systems, Man and Cybernetics*, (2):408–421, 1972.

[87] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, California, 2nd edition, 2005.

[88] K. Woods, C. Doss, K. Bowyer, J. Solka, C. Priebe, and W. Kegelmeyer. Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography. *International Journal of Pattern Recognition and Artificial Intelligence*, (7):1417–1436, 1993.

[89] Q. Zhao and T. Nishida. Using qualitative hypotheses to identify inaccurate data. *Journal of Artificial Intelligence Research*, 3:119–145, 1995.

[90] Z. Zheng, X. Wu, and R. Srihari. Feature selection for text categorization on imbalanced data. *SIGKDD Explorations*, 6(1):90–89, 2004.

[91] X. Zhu and X. Wu. Class noise vs attribute noise: A quantitative study of their impacts. *Artificial Intelligence Review*, 22(3-4):177–210, November 2004.

[92] X. Zhu and X. Wu. Cost-guided class noise handling for effective cost-sensitive learning. In *Proc. of the ICDM Conf*, pages 297–304, 2004.

[93] X. Zhu, X. Wu, and Q. Chen. Eliminating class noise in large datasets. In *Proceedings of the Twentieth International Conference on Machine Learning*, Washington, DC, 2003.