Evolutionary Instance Resampling for Difficult Data Sets

by

William Dale Richardson

(Under the Direction of Khaled Rasheed)

Abstract

In the field of machine learning, data set features such as across-class imbalance and class overlap often pose difficulties for classifier algorithms. A number of methods alleviate these difficulties by adjusting the distribution of the data set before classifier construction. Resampling is typically effected by re-weighting, removing, or duplicating instances. Finding a good distribution for the data set, however, is a nontrivial problem. Evolutionary algorithms are frequently used to search for solutions in large, difficult search spaces. In this thesis, four evolutionary approaches are applied to the problem of instance resampling across a variety of data sets and classifier paradigms. In many cases, the evolutionary pre-processing methods are able to produce better classifiers. In particular, an integer-based, one-to-one representation and a cluster-based, real-valued weighting scheme are shown to be beneficial for improving classifier performance on difficult data sets.

Index words: genetic algorithms, machine learning, imbalance, undersampling, oversampling, instance selection

Evolutionary Instance Resampling for Difficult Data Sets

by

William Dale Richardson

B.S., Washington and Lee University, 2011

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2013

Evolutionary Instance Resampling for Difficult Data Sets

by

William Dale Richardson

Major Professor:	Khaled Rasheed
Committee:	Walter D. Potter
	Prashant Doshi

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2013

Evolutionary Instance Resampling for Difficult Data Sets

William Dale Richardson

November 13, 2013

Acknowledgments

My mother, Sherry Richardson, and my father, Daniel Richardson, have been a source of unwavering support throughout my life. This thesis would not have been possible without them. I would also like to express my gratitude toward my instructors at Washington and Lee University who helped prepare me for this endeavor including Dr. Kenneth Lambert, Dr. Simon Levy, Dr. Rance Necaise, Dr. Sara Sprenkle, Dr. Joshua Stough, and Dr. Nathaniel Goldberg. At the University of Georgia, I would like to thank Dr. Walter Potter, Dr. Khaled Rasheed, and Dr. Prashant Doshi. Their guidance throughout my time at the Institute for Artificial Intelligence has been invaluable. Additionally, the resources prepared by Dr. Michael Covington have been an enormous help in formatting this work. Finally, I would like to thank my friends who have helped me along this proverbial journey, especially Peter Geiger, Greg Lennon, and E.W. Malachosky.

Contents

Acknowledgements **iv**

1 Introduction **1**

2 Problem Background **5**

2.1 Boosting 16

2.2 Evolutionary Methods for Improving Classification 21

2.3 Outlier Detection 27

3 Proposed Methods **31**

3.1 Measure of Merit 32

3.2 Genetic Representations 34

3.3 Population Initialization 36

3.4 Genetic Algorithm Configuration and Operators 38

4 Experimental Setup **42**

4.1 Artificial Data Sets 42

4.2 Real-World Data Sets 44

4.3 Machine Learning Components 45

5 Results and Analysis **47**

5.1 3-NN Classifier Performance 48
5.2 J48 Classifier Performance 51
5.3 Multilayer Perceptron Classifier Performance 54
5.4 Support Vector Machines Classifier Performance 55
5.5 Effects of Imbalance and Overlap 58
5.6 UCI Data Set Performance 59

6 Conclusions and Future Directions **62**

Bibliography **66**

Appendices **73**

A Data Resampling Techniques **74**

B Data Set Performance Charts **75**

List of Tables

3.1 Genetic algorithm parameter configuration 38

3.2 Real-valued mutation operator 40

3.3 Across-class cluster weighting mutation operator 41

4.1 Artificial data set controlled parameters 43

4.2 UCI data sets 45

4.3 Modified UCI data set class composition 45

5.1 3-NN classifier performance on the artificial data sets 50

5.2 Wilcoxon signed-rank test for the 3-NN classifier's performance on the artificial data set 51

5.3 J48 classifier performance on the artificial data sets 53

5.4 Wilcoxon signed-rank test for the J48 classifier's performance on the artificial data sets 54

5.5 Wilcoxon signed-rank test for the MLP classifier's performance on the artificial data sets 55

5.6 SVM classifier performance on the artificial data sets 57

5.7 Wilcoxon signed-rank test for the SVM classifier's performance on the artificial data sets 58

5.8 3-NN classifier performance on the UCI data sets 59

5.9 J48 classifier performance on the UCI data sets 59
5.10 MLP classifier performance on the UCI data sets 60
5.11 SVM classifier performance on the UCI data sets 60
5.12 Wilcoxon signed-rank test for the 3-NN classifier's performance on the UCI data sets 61
5.13 Wilcoxon signed-rank test for the J48 classifier's performance on the UCI data sets 61

Chapter 1

Introduction

In the field of machine learning, a number of features that can make data sets more difficult to model have been identified. One such feature that has received a great deal of attention is between-class imbalance, or simply imbalance. A data set is said to be imbalanced when the number of instances belonging to a class of interest is relatively small. If data suffers from imbalance, it can be substantially more difficult to construct an effective classifier, particularly one able to recognize members of the small class. Additionally, it can cause some conventional methods of measuring classifier performance, such as accuracy, to fail to effectively measure the degree to which a classifier has "learned" a concept. Recent research, however, indicates that relative imbalance between classes is typically not a problem *per se*, but that it can combine with other factors such as overlap, small disjuncts, high concept complexity, and sampling differences between the training and testing sets to decrease classifier performance for many machine learning paradigms. A data set suffers from overlap if many instances of differing classes share the same region of the instance space. Overlap is most obviously apparent in the case of noisy or erroneous data, but points in overlapping space may merely represent the complexity of a target concept. Overlap may often be more disruptive for a machine learning algorithm than imbalance. As both imbalance and overlap

increase, they produce a synergistic effect, degrading classifier performance and increasing model complexity drastically. High concept complexity is present when a class is composed of a relatively large number of separate clusters. For classes that are under-represented, the points are spread across a larger area, forming small disjuncts or rare cases. A disjunct is an area of the instance space defined by a collection of instances corresponding to a sub-concept, and a disjunct is small if the number of instances composing it (i.e. its coverage) is also small in the context of the data set. These regions are problematic for many inductive algorithms to learn, because they are difficult to distinguish from noise or other statistically insignificant data, especially for methods that are biased toward maximum generality. Finally, should the distributions of the training data and the testing data be different. This phenomenon is known as data fracture. Nonetheless, the data sets available in a given problem may suffer from any combination of these features in varying degrees. They may arise from shortcomings or difficulties in data acquisition, or they may represent intricacies of the underlying distribution. In either case, it is important to develop techniques for improving classifier learning on difficult data sets, and, having done so, it is important to evaluate under what circumstances these techniques are most beneficial.

There are a number of techniques for improving classifier performance on difficult (particularly imbalanced) data sets. They are typically divided into two classes: those which modify the behavior of an inductive learning method and those which modify the training data so that a learning method's behavior is more effective. Comparisons of the two classes have indicated that, in general, neither has any discernible performance advantage and that both can produce improvements. One advantage of modifying the data set's distribution, however, is that it does not require one to select a particular machine learning paradigm in advance. Having altered the data set, one can then experiment with different learners to determine which one is best suited for the problem at hand. Modifying the distribution, or resampling, can be accomplished in many ways. Removing instances (undersampling)

and duplicating instances (oversampling) have been shown to improve performance in imbalanced domains. Additionally, some methods employ synthetic data points to modify the distribution. In all of these cases, the data sets are manipulated in order to increase or decrease emphasis on certain instances or regions of the sample space, more directing the inductive behavior of the classifier than changing it. The effectiveness of data pre-processing is, of course, limited by the initial distribution of the data set. It is impossible to extract information that is not present in the data, and given a high level of noise, it may be impossible to discern an underlying concept at all. Even so, existing pre-processing methods are capable of data refinement and adjustment in ways that produce substantial classifier improvement, and they do so without the use of *a priori* or task-specific information.

The difficulty of challenging data sets often lies in distinguishing between noise and exceptional, valid cases that are important for constructing an effective classifier. For a pre-processing method applied to such a data set, the inclusion, exclusion, duplication, or weighting of instances becomes more difficult accordingly. Many pre-processing algorithms use simple heuristics to select which points in the data set should be discarded, duplicated, or used to create artificial instances. Other methods make these selections entirely randomly. In either case, the search among possible instance subsets is (with few exceptions) unguided by feedback from machine learning methods. Searching this space is problematic because even in the simplest cases (including/excluding each instance or duplicating select instances a single time), the size of the search space grows exponentially with the number of instances in the data set. For variable-quantity instance duplication or real-valued instance weighting for a data set of the same size, the solution space is even larger. As the search space grows, and brute force becomes increasingly impractical, it becomes more important that the search method be efficient in navigating that search space.

Genetic algorithms (GA) have proven to be a robust, reliable method for finding good solutions in large search spaces with many local optima. This suggests that the GA is well-

suited to the problem of adjusting the sample space distribution. This adjustment can be effected in many different ways, and this is reflected in the wide variety of approaches that have been proposed. One strength of the GA is that it can accommodate many different representations within a powerful and robust heuristic search framework. This potential, however, is not without drawbacks. Genetic algorithms, while well-suited to a wide variety of problems, often offer poorer performance than domain-specific algorithms for a specific problem. They have no guarantee of finding a global optimum, and they may require substantially more computation time. While domain-specific operators can ameliorate this inefficiency, ideal applications for the genetic algorithm are relatively difficult instances in domains where no adequate domain-specific method exists and simpler methods such as hill-climbing are not sufficient. In the context of this research, that means that ideal data sets for modification by a genetic algorithm exhibit high degrees of imbalance, overlap, concept complexity, or data fracture, as existing methods (random oversampling/undersampling, cost-sensitive learning, SMOTE and its variations) already offer substantial classification improvements for moderately difficult data sets at a far lower computational cost. In this thesis, genetic algorithms are applied to the task of modifying the distribution of a data set to improve the quality of classifiers constructed from the set. Five different representations are developed, and their performances are evaluated on both real and artificial data sets. The objective is to develop a data pre-processing method for adjusting the distributions of difficult data sets so that classifiers produced from the data set are more effective than classifiers built from the raw data set. Finally, analysis is undertaken to determine under what conditions each method produces the greatest improvements.

Chapter 2

Problem Background

Class imbalance in data sets has been identified as a hindrance for classifier performance in a number of real world domains including intrusion detection, oil-spill detection from satellite imagery, and medical data [Kubat et al., 1997, Autio et al., 2007]. Although the negative effects of imbalanced data were once widely believed to be caused by large relative differences in the number of instances in each class, recent work indicates that the situation is more complicated. In one of the most influential papers on the effect of imbalance, Japkowicz and Stephen perform a comprehensive experiment to measure the effect of imbalance. They study imbalance independently and in conjunction with other features of data sets: set size and target concept complexity (i.e. the number of sub-clusters that compose each class). They also compare three general pre-processing methods for correcting imbalance: random oversampling, random undersampling, and cost-adjusting the minority examples by the imbalance ratio. Their experiments indicate that, of C5.0 decision trees, multi-layer perceptrons, and support vector machines (SVMs), decision trees were the most negatively affected by imbalance, multi-layer perceptrons are affected proportionately to the degree of imbalance, and SVMs were virtually unaffected. Among remedial methods, oversampling and cost-modifying are shown to work very well for the classifiers that suffer from imbalance,

though they hindered the performance of SVMs. Most importantly, the authors conclude that imbalance itself does not actually effect classification negatively; rather, when classes are very imbalanced, the sub-clusters comprising each class are not large enough to be generalized upon effectively. Hence, the authors argue, imbalance together with small data set size and/or high target concept complexity produce the negative effect traditionally attributed to imbalance alone [Japkowicz and Stephen, 2002]. Other work supports these conclusions: Batista et al. conduct an experiment to determine the effect of imbalance across a variety of UCI data sets and gauge the benefit of a number of pre-processing techniques. The authors find that, in general, oversampling is more effective than undersampling. They also find that imbalance is most problematic in conjunction with other aggravating features. Finally, the authors find that in many cases, using approaches based on Chawla et al.'s Synthetic Minority oversampling Technique (SMOTE) produced better results than randomly oversampling, though the latter was more computationally efficient for some domains [Batista et al., 2004]. The success of SMOTE and its variants indicates that greater increases in performance are possible through more sophisticated pre-processing methods, i.e. that random resampling is not the upper bound on data pre-processing performance.

SMOTE is a popular method of data pre-processing that works by generating artificial examples to change the local distribution of instances. It was developed specifically to operate on imbalanced data sets as an oversampling method that would improve recall without causing overfitting in the classifier. The algorithm operates by randomly selecting minority-class instances and interpolating at a random interval between those instances and their nearest neighbors. It has a single parameter: intuitively, the number of artificial samples to generate. The authors are able to demonstrate the benefit of SMOTE both accompanying random majority undersampling and in isolation. Toward this end, they employ an analysis of the receiver-operating characteristic curve (ROC) and compare the areas under these curves (AUC) [Chawla et al., 2002]. SMOTE is relevant to this thesis because it is

a powerful pre-processing method and because it forms the basis of numerous resampling methods for difficult data sets [Chawla et al., 2003, Batista et al., 2004, Prati et al., 2004, Batista et al., 2005, Han et al., 2005]. It has been widely adapted likely because introducing synthetic data points was a noteworthy innovation in data pre-processing, SMOTE has produced excellent results in a number of studies, and because it is a simple algorithm that is easy to alter or extend.

Another idea important to the methods examined in this paper is that data pre-processing can be applied in a wrapper framework so that the resampling is guided. Chawla et al. conduct an investigation to gauge the effectiveness of a SMOTE-based wrapper method to pre-process the data. The intuition behind this is that the resamplings produced by SMOTE with different parameter settings are not necessarily equal in quality, but if an algorithm can receive feedback from the classifier, it can search for a good configuration. The wrapper extension of SMOTE operates this way, using the classification results on a validation set to tune two parameters: the number of synthetic examples generated and the percentage of the majority class to undersample randomly. The authors note that using a wrapper approach imposes some requirements that non-wrapper methods do not have. The data set will require a subset of the training data to guide the wrapper algorithm; for data sets undergoing cross-fold validation, this may be implemented by a second level of cross-validation within the training set. Second, wrapper adaptations of any method naturally require more computational time than the method itself. As a counter to this increased time, the authors suggest that parallelization can be used to reduce execution time, if not the total compuational burden, as the problem lends itself well to division (e.g. each fold could be assigned to a different processing unit). The authors also compare the effectiveness of various guiding criteria for the wrapper technique: a cost-sensitive metric (with the imbalance ratio used to control the relative cost), AUC, F1-measure, and a variant of the F-measure in which the parameter β varies with the relative cost/imbalance. They

find that the non-cost-sensitive measures produce good performance, which is valuable since these methods are available in domains where the cost matrix is not know *a priori* or the misclassification costs changes over time. Among these two measures, the AUC performs at least as well as the F-measure, and Chawla et al. recommend it for guiding wrapper-based pre-processing approaches [Chawla et al., 2008]. To gauge the effectiveness of their method, they also compare all of the SMOTE versions against cost-sensitive techniques MetaCost and CostSensitiveClassifier (CSC), varying the cost disparity between two classes. They observe that at lower disparities (2:1), the two cost-sensitive methods are able to produce classifications with lower overall cost, but at higher ratios (10:1 and 20:1), the wrapper methods are more effective [Chawla et al., 2008]. This study is particularly valuable to this research because it is an excellent example of corrective data pre-processing applied in a wrapper framework.

The relationship between imbalance and overlap has also been the subject of substantial research. Prati et al. conduct an experiment in which they vary overlap and imbalance independently on artificial data sets. These data sets consist of five-dimensional data points (not including the label) in two classes drawn from a Gaussian distribution with a standard deviation of 1.0. Across these sets, the imbalance ratio varied from ninety-nine to one, and the distance between the means varied from nine standard deviations to zero (i.e. the class distributions had the same mean) to increase overlap. The authors evaluate the AUC of C4.5 classifiers trained on these data sets. They find that overlap causes severe degradation in classifier performance and that this degradation was more acute in the presence of imbalance [Prati et al., 2004]. Imbalance alone also hindered effective tree construction, but its effect in isolation was less pronounced than that of overlap. As this thesis attempts to mitigate the difficulty of data sets, studies like this one that seek to identify the source of this difficulty are very valuable.

Garciía et al. perform an empirical comparison of various classifier paradigms' perfor-

mance on imbalanced data sets with varying degrees of overlap. Their experiment also investigates the effects of having an globally imbalanced minority class be dominant within the overlapping region. Using a 1-NN classifier, a radial basis function (RBF) network, a naïve Bayes classifier, a multi-layer perceptron, a C4.5 classifier, and a support vector machine (SVM) classifier, they examine the effect of increasing amounts of overlap, holding the degree of imbalance and training set size constant for artificial data sets with a uniform distribution. Their findings indicate that classifier recall for the minority class typically decreases as overlap increases, but the degree to which this occurs varies across paradigms. While 1-NN and SVM are very sensitive to overlap, naïve Bayes classifiers are relatively robust in the face of overlap [García et al., 2007]. Additionally, when the distribution of data in the overlapping region was skewed heavily toward the minority class, the recall is substantially higher, even demonstrating more reliable classification of the minority class than the majority class. This second conclusion suggests that examining data set features and altering distributions on a global scale may not be optimal. To produce greater improvements, it may be beneficial to develop methods which are sensitive to local imbalance and overlap and are capable of resampling at a local scale.

Focusing exclusively on the behavior of SVM classifiers, Denil and Trappenberg investigate the effects of varying data set size, class imbalance, and degree of overlap. They hypothesize that imbalance and overlap operate independently to inhibit SVM learning. From their experiments, they find that imbalance and overlap together drastically increased the complexity of the model and decreased the F1-measure of the SVM classifier [Denil and Trappenberg, 2010]. Their research suggests that while SVMs are largely unaffected by imbalance (provided that there is sufficient training data), imbalance and overlap work non-linearly to inhibit learning when both are present. Although this research is limited to one classifier paradigm, it demonstrates that the effects of complicating factors in a data set can be amplified when they are found in combination, i.e. they produce synergistic degradation.

Another factor affecting data sets and their ease of classification is the problem of rare cases. This is also known as the problem of small disjuncts or within-class imbalance. As early as 1989, the problem of small disjuncts and their inhibitive effects on machine learning have been a subject of research [Holte et al., 1989]. Most inductive concept learners operate by identifying groups of instances in the attribute space referred to as disjuncts. Since these methods are often biased towards maximum generalization, disjuncts with a high degree of coverage (those which contain a large number of instances) tend to be more easily recognized. Small disjuncts, or rare cases, occur when sub-concepts (particularly of the minority class) are represented by few instances in the data set and thus may be difficult to distinguish from noise. Holte et al.'s work focuses primarily on undersampling: each disjunct identified in a model is subject to possible removal from the concept representation. Its inclusion or exclusion is based on criteria such as error rate and statistical significance. They also investigate the effect of applying different inductive biases to the disjuncts depending on their sizes; while this study does not definitively answer the problem of how to treat small disjuncts, it is certainly a great step forward in understanding and responding to them.

Nathalie Japkowicz and her associates also examine the effect of small disjuncts. They refer to this issue as within-class imbalance, framing the problem as one of classes' component sub-concepts being too small. In one article, Japkowicz demonstrates that within-class imbalance can have detrimental effects on classifier performance, and that relatively simple methods can substantially reduce the number of misclassifications [Japkowicz, 2001]. She performs an empirical experiment using artificial data sets to analyze the effect of within-class imbalance and its interaction with between-class imbalance. She also proposes a corrective data pre-processing method using cluster-based oversampling which balances both within-class imbalance and between-class imbalance. Nickerson et al. extend this method by extending it with an unsupervised clustering technique called Principle Direction Divisive Partitioning (PDDP). The previous version required that one have *a priori* knowledge

about the underlying distribution and its clusters, but the addition of PDDP eases this constraint. The authors resample a small number of data sets with their cluster-based "guided oversampling" approach as well as "blind oversampling," and classifiers constructed using these techniques are compared with a classifier constructed from the original data. Nickerson et al. do not report results that are especially promising, but they do note some improvement over both blind oversampling and the control, suggesting that correcting within-class imbalance may still be important [Nickerson et al., 2001]. In 2004, Jo and Japkowicz examine further the degree to which the within-class imbalance problem is responsible for the performance degradation previously ascribed to between-class imbalance. They perform a number of experiments using C4.5 and backpropogation artificial neural networks to classify both artificial and UCI data sets. They compare the use of traditional imbalance pre-processing methods with the cluster-based method (without PDDP) proposed in the aforementioned paper [Japkowicz, 2001]. They conclude that the small disjuncts problem is more responsible for the degradation of classifier accuracy than between-class imbalance and that addressing both forms of imbalance is more effective than remedying only between-class imbalance[Jo and Japkowicz, 2004]. They argue that remedying the small disjuncts problem should hence be the focus of research on improving difficult data sets. These works are important to this thesis because they elucidate the relationship between imbalance and rare cases and because using clustering to identify sub-concepts during resampling is essential to two of the metods evaluated here.

In 2004, Weiss also investigates the interaction of rare cases and imbalance. He presents the empirical data regarding their interaction with methods for correcting data sets in a literature survey. Weiss's study identifies improper evaluation metrics (e.g. accuracy), absolute and relative rarity (in terms of both classes and concepts), data fragmentation (an artifact of some classifier methods), inappropriate inductive biases, and noise as factors which can contribute to the difficulty of classifying rare disjuncts. The author also goes on to examine

a variety of methods designed to improve data mining performance on difficult data sets, and provides a summary of which methods are applicable for correcting which complicating features. He concludes that rare classes and rare cases (i.e. between-class and within-class imbalance) are very similar in nature and can be described and remedied using a single abstract framework [Weiss, 2004]. This is not necessarily a contradiction of Jo and Japkowicz's conclusion, but Weiss advocates addressing two related problems in a unified approach, while Jo and Japkowicz suggest that there is only one problem (i.e. within-class imbalance) and that the traditional imbalance problem is not worthwhile by itself. In any case, this study provides an excellent summary and discussion of the work done on the small disjuncts problem.

Moreno-Torres and Herrera examine imbalance and overlap in addition to a factor called data fracture, which occurs when the data comprising the training set has a substantially different distribution than that of the testing set. While data fracture is not a feature of any data set itself, the authors argue that its effect can be especially pronounced in imbalanced domains. They propose a feature extraction method for imbalanced data sets described below [Moreno-Torres and Herrera, 2010]. Data fracture, also referred to as data-shift, receives relatively little attention compared to imbalance, overlap, and rare cases.

A number of studies have also been conducted to compare the effectiveness of various pre-processing methods in countering various complicating data features like imbalance and overlap. Batista et al. generate artificial data sets with Gaussian distributions, using a method developed previously by Prati et al.[Prati et al., 2004]. In this paper, they also apply five pre-processing methods to the data: random undersampling, random oversampling, Laurikkala's NCL, Chawla et al.'s SMOTE, and SMOTE together with Wilson's Edited Nearest Neighbor Rule (SMOTE+ENN). Measuring the AUC of a C4.5 classifier, the authors find that the undersampling techniques (NCL and random undersampling) generally do not perform as well as the oversampling techniques [Batista et al., 2005]. Of the latter, the

SMOTE and SMOTE+ENN offered the best performance; in particular, SMOTE+ENN produced great improvement in overlapping domains. The authors suggest that it excels in overlapping domain because the application of ENN allows it to "clean" noisy border areas.

As a counterpoint, Van Hulse et al. conduct a study using thirty-five real world data sets, comparing the benefits of applying random oversampling, random undersampling, Kubat and Matwin's One-Sided-Selection, SMOTE, Japkowicz et al.'s cluster-based oversampling, Wilson's editing rule, and Han et al.'s borderline-SMOTE. They evaluate the G-mean, F-measure, AUC, true positive rate, and accuracy for two configurations of C4.5, multilayer perceptrons, radial basis function netweork, RIPPER, a random forest classifier, a logistic regression learner, and a naive Bayes classifier. Each classifier is used in conjunction with each pre-processing method on every data set. The authors find that different resampling methods produce better results depending on both the classifier used and that different measures of merit rank their performances differently. In summary, though, they conclude that random undersampling performs the best overall, followed by random oversampling [Van Hulse et al., 2007]. The two SMOTE-based approaches perform moderately well, while OSS and the cluster-based method for correcting within-class imbalance perform the worst by far. Cluster-based oversampling offered the worst-ranked performance more often than using no resampling at all (i.e. the control for pre-processing). This is remarkable as the non-random methods were typically developed to address shortcomings in random oversampling and undersampling. Which pre-processing technique yields the greatest improvements for a given data set depends on the data set itself. This study covers a substantial group of data sets, but it does not give a thorough account of those sets' features: it only groups them roughly by degree of imbalance. Although Van Hulse et al., argue that random undersampling produced excellent results for imbalanced data, imbalance itself (as noted above) is considered by many researchers to be a secondary factor in the difficulty of data sets. Unfortunately, it is not possible to build strong conclusions from this study about which data

sets benefited most from which methods. If nothing else, however, this study shows that random resampling ought not be discounted as a remedial technique. Despite its simplicity, it can produce excellent results for many different data sets.

López et al. perform a comprehensive analysis comparing the effect of data pre-processing methods, cost-sensitive learners, and a combination of the two as a means for improving classification in imbalanced domains [López et al., 2012]. They examine genetic fuzzy classifiers, SVMs, k-NN classifiers, C4.5 classifiers, as they are normally implemented and as cost-sensitive variants. The authors train these classifiers on a large battery of imbalanced data sets, with and without two variants/augmentations of SMOTE. They analyze the performance of hybrids where the pre-processing techniques are applied to the data sets before they are given to the cost-sensitive methods. López et al. find that both cost-sensitive classifiers and normal classifiers trained on pre-processed data work well, but the combination of the two does not appear to offer a statistically significant advantage. Additionally, they provide an extremely thorough review of the prior research performed on the relationship between imbalance and other aspects of data sets such as overlap and data-shift (previously referred to as data fracture here). Finally, they conclude that there is still a need for classifiers which are able to deal with difficult data sets characterized by imbalance, overlap, and data-shift. This can be difficult since remedies for one deficiency of the data set may exacerbate another, e.g. as the authors observed, synthetic oversampling can also increase the degree of overlap.

In one of the most recent and comprehensive studies on the subject of difficult data sets, Jerzy Stefanowski conducts a thorough experiment varying data set size, levels of imbalance, degree of overlap, and concept complexity [Stefanowski, 2013]. While most prior studies examine only two of these factors at once, Stefanowski's work varies a set of artificial data sets across these four variables in addition to two more: boundary shape (linear and nonlinear) and, more importantly, rare cases located within the majority-class

region. Rare examples within majority regions are included in the scope of the experiment because of their observed occurrence in UCI data sets, and the author argues that such phenomena might be present in other imbalanced, non-artificial data sets. Examining the varying performance of a number of classifiers including an inductive tree builder (J48), a K-nearest neighbors classifier, and a rule-based learner (Jrip), the author is able to draw a number of conclusions. First, echoing the conclusions of Japkowicz, García, and others cited above, relative imbalance itself causes little, if any, additional difficulty by itself, but with other features, particularly concept decomposition (i.e. higher concept complexity), it can cause issues for learning. Additionally, the experiment supports the finding that overlapping (as measured by the number of instances in a borderline region) and rare examples (individuals or small groups of minority-class examples within majority-class dominated space) can cause even greater difficulty than concept complexity. Unsurprisingly, the intersection of these features provides the greatest difficulty for classifiers. With respect to class boundaries in the instance space, the authors found that non-linear boundaries typically posed greater difficulty for classifiers, which is important to note as many artificial data sets used in previous experiments had linear class/concept borders [Japkowicz and Stephen, 2002, García et al., 2007, Denil and Trappenberg, 2010].

In the study's second section, Stefanowski also evaluates the performance of various preprocessing methods for improving classifier performance and compares them to a proposed method [Stefanowski, 2013]. The experiment compared the following methods: random minority oversampling, Japkowicz's cluster-based oversampling, Laurikkala's Neighborhood Cleaning Rule (NCL/NCR), and Stefanowski et al.'s SPIDER. These techniques are evaluated by the degree to which they are able to preserve sensitivity in the presence of overlap and (separately) varying proportions of rare examples. SMOTE is notably absent from the list above, but this is deliberate. SMOTE is the subject of substantial discussion as the author addresses some of its disadvantages: it may increase overlap in borderline areas, and

it tends to increase generalization globally across the instance space, irrespective of local distributions. Since the objective of the experiment is to evaluate the relative ability of methods to improve performance in borderline regions and on rare cases (precisely the areas in which SMOTE is allegedly prone to failure), it is omitted. The experiments indicate that all of the methods examined were capable of producing improvements. While random oversampling and cluster oversampling are able to provide stronger performance on relatively simple data sets, NCR and SPIDER perform better on the more difficult data sets containing rare examples or overlap. In particular, SPIDER provided the best performance on difficult data sets; while both SPIDER and NCR provided substantial gains in sensitivity, SPIDER had a less adverse effect on specificity. This suggests that pre-processing techniques that are able to take into account local features of the instance space offer greater potential gains for difficult data sets.

2.1 Boosting

Although the term "boosting" can be used to refer to any method which strengthens a classifier, it is often used to refer specifically to the practice of assigning weights to instances in order to improve classifier performance. In this paper, "boosting," takes the latter meaning. Furthermore, most references to "boosting" refer to extensions and adaptations of one particular method of assigning weights to instances: AdaBoost. Fruend and Schapire's AdaBoost has demonstrated strong performance on a variety of domains, and it has been shown to substantially extend the capability of weak learners [Freund et al., 1996]. In the domain of character recognition, AdaBoost is used for prototype selection for a k-NN classifier. AdaBoost constructs an ensemble of hypotheses one by one; these hypotheses are typically generated by weak learners. A weighted combination of votes is used to determine the ensemble's classification of a given instance. With the addition of each new classifier,

the ensemble's performance is evaluated and instances that are classified correctly have their relative weights reduced so that consistently misclassified instances are given higher priority. Weights affect the classifier either by using a classifier sensitive to weights applied to instances or by generating the training distribution by using the weight as a probability of selection, i.e. oversampling difficult instances and undersampling easy ones probabilistically. The relevance of boosting to this thesis is easily apparent, as improving classification on difficult concepts and the instances which comprise them is the purpose of this research.

In imbalanced domains, it has been shown that cost-sensitive classifiers, i.e. those which can take into account different misclassification costs, can perform better than naive counterparts [Japkowicz and Stephen, 2002, Chawla et al., 2008, López et al., 2012]. Fan et al. propose AdaCost, an adaptation of AdaBoost designed to provide improved classification in domains where misclassifications of some data are considered more costly (i.e. less desirable) than others; their application domain is the detection of credit card fraud [Fan et al., 1999]. AdaCost proves effective, as it is consistently able to lower classification costs over the data without using more computational power. Since the objective was framed specifically in terms of lowering costs as given by a cost matrix, any improvements of traditional classifier performance measures scuh as the AUC, F-measure, or G-mean, are not noted, but it is certainly possible to devise cost weights to maximize any of those measures. The success of other approaches which have used cost-sensitivity to adjust for imbalance suggest that cost-sensitive boosting could be a fruitful avenue of research.

Adapting AdaBoost for imbalanced domains, Sun et al. propose and evaluate three possible ways to incorporate a misclassification cost into AdaBoost's weight update function: inside the exponential term, outside the exponential term (as a factor), and in both places. These methods are called simply AdaC1, AdaC2, and AdaC3, respectively. The cost factor is used to encourage learning false negatives more than false positives, resulting in better recall for the minority class, with the ultimate goal maximization of the F-measure. The authors'

methods are compared to AdaBoost, AdaCost, and another boosting technique called CSB2; these methods are tested using C4.5 trees and a high-order pattern and weight-of-evidence rule (HPWR) based classifier. F-measure is used as the primary measure of classifier performance across four data sets from medical domains. Sun et al. observed that AdaC2 generally performed better than the other methods evaluated, and that it was more sensitive to the costs applied, making it more suitable for imbalanced domains [Sun et al., 2007]. This method is important for this thesis because it is an application of boosting specifically for data sets suffering from imbalance.

Furthering the idea that boosting can improve performance on imbalanced data, Chawla et al. augment AdaBoost with SMOTE to form SMOTEBoost [Chawla et al., 2003]. SMOTE, as described above, is a stochastic technique for synthetic data generation in order to alleviate the effects of imbalance. Using the AdaBoost.M2 algorithm as a framework, SMOTE is called repeatedly to generate data points in the minority class before each new weak hypothesis was generated. Thus, the authors claim that they are able to direct boosting to focus not merely on the "difficult" instances, but specifically the difficult instances in the minority class. In Chawla et al.'s results, SMOTEBoost is able to perform better in the imbalanced intrusion detection domain than the weak classifier (RIPPER), the weak classifier augmented with AdaBoost, and the weak classifier augmented with SMOTE as measured by the classifiers' F-score and recall. This demonstrates the effectiveness of boosting combined with other methods for correcting data set imbalance.

In order to compare the efficacy of bagging and boosting in the presence of noise and imbalance, Khoshgoftaar et al. conduct an empirical study comparing four metalearner methods across seven different performance measures. Specifically, the authors evaluate SMOTEBoost, RUSBoost, Exactly Balanced Bagging (EBBag), and Roughly Balanced Bagging (RBBag). The bagging techniques are employed both with and without replacement. For each boosting method two post-correction levels of imbalance were used, so that after

application, the data set was composed of either 35% or 50% minority examples. Thus, in effect, eight different methods were used. By artificially altering the levels of noise, the imbalance level, and the class distribution of the noise in four UCI data sets, the authors compare their comparative performance across those factors. Their findings show that bagging consistently outperforms boosting, especially as the amount of noise present in the data set increases [Khoshgoftaar et al., 2011]. Confirming earlier work, they find that more noise in the minority class (i.e. false negatives in the data set) is more detrimental to performance than noise in the majority class (i.e. false positives) in all cases, though boosting suffers the more than bagging. Between EBBag and RBBag, performance was virtually identical, though the versions without replacement offered better classification results than the versions with replacement. Between the two boosting methods, RUSBoost generally yielded better results than SMOTEBoost. The authors explain the effect of noise on boosting by pointing out that noisy points will typically remain difficult to classify, and so as boosting progresses, they will be given increasing amounts of weight. To demonstrate this effect, they provide tables which show the progression of weights over time for noisy examples. Thus, they are able to effectively argue that in the presence of noise, traditional boosting methods' performance suffers substantially, especially compared to bagging.

One issue in the boosting literature relevant to the methods in this thesis refers to whether it is better to effect weights on instances through direct weighting or resampling. An empirical study of this question is presented by Seiffert et al. to compare a variety of boosting methods using both resampling and direct reweighting. The boosting algorithms examined were AdaBoost, AdaCost, AdaC1, AdaC2, AdaC3, CSB0, CSB1, CSB2, RareBoost, and SMOTEBoost. Since SMOTEBoost, as originally defined, does not have a method for applying weights directly, the authors devise an adaptation. The area under receiver-operating characteristic curve and area under precision recall curve were examined across fifteen data sets, and four classifiers were used (two configurations of C4.5, RIPPER, and a naive Bayes

classifier). From this experiment, the authors find that across the boosting methods evaluated, resampling the data generally performed as well as (and often better than) directly weighting instances [Seiffert et al., 2008]. This is notable because the latter is typically the default implementation. As an interesting side note, (and an exception to the result above) the authors' resampling adaptation of SMOTEBoost performed better than the original. The general trend, however, seems to indicate that resampling data may be more useful in directing the emphasis of machine learning methods onto problematic instances.

Finally, Galar et al. propose a combination of boosting with an evolutionary undersampling technique to simultaneously maintain classifier diversity within the ensemble and improve classification on imbalanced data sets. Adapting AdaBoost.M2, they use a steady-state genetic algorithm (CHC) to evolve inclusion/exclusion of majority class instances for each boosting iteration. The fitness criterion in the genetic algorithm is a function of the G-Mean of a 1-NN classifier trained with the included majority examples as well as the imbalance ratio, a term composed of the number of examples and the iteration number, and (optionally) a measure of how different the individual is from the rest of the population. Two different measures of diversity are evaluated: the Q-statistic and the Hamming distance. Both measure the distance between two individuals, so the diversity of an individual is measured by examining it with each data set used in previous iterations of boosting and taking the maximum distance among these. The method is also used without any diversity maintenance mechanism included at all. These three configurations of EUSBoost are compared against one another, and the authors find that the EUSBoost that incorporated the Q-statistic to establish and maintain classifier diversity performed significantly better than the other two on the thirty-three most imbalanced binary data sets from the KEEL repository. These methods are also compared against RUSBoost, SMOTEBoost, SMOTEBagging, EasyEnsemble, and Underbagging, which are state-of-the-art metalearners for imbalanced domains. All of these methods, with the exception of RUSBoost, perform significantly worse than EUSBoost

with Q-statistic; although RUSBoost's performance was worse than EUSBoost, the difference was not statistically conclusive as determined by a Holm test [Galar et al., 2013]. The experiments show that EUSBoost is a very competitive method for improving boosting in imbalanced domains. The authors also use this paper to demonstrate the usefulness of kappa-error diagrams for comparing the performance of different learning ensemble techniques, and they propose an adaptation that is better suited for imbalanced domains (since the original makes use of the accuracy). This technique allows them to determine that while EUSBoost tends to produce better classification results, it did so at the expense of classifier diversity, at least when compared to RUSBoost [Galar et al., 2013]. This paper is relevant because it is a recent, successful application of evolutionary techniques to conventional machine learning (i.e. boosting) to modify instance weights and direct classifier learning emphasis.

2.2 Evolutionary Methods for Improving Classification

A wide variety of machine learning approaches have been developed which implement evolutionary techniques to enhance the performance of conventional machine learning algorithms. Addressing all such methods is beyond the scope of this thesis; however, a number of particularly relevant techniques are described below. Many of these methods were devised to counter imbalance, noise, or other complicating factors in data. Their relevance to the thesis is self-apparent. Other methods are included because the tasks performed by their evolutionary components are similar to those performed by the methods evaluated in this research. Genetic classifier systems are not thoroughly addressed here as they comprise more an independent machine learning technique than an application of evolutionary methodology to improve other methods.

A number of algorithms use genetic methods to select a representative sample of the training set. Kuncheva uses a GA to select a reference set for a k-NN classifier from a set of training

data, removing redundant examples and reducing the computational burden of classification [Kuncheva, 1995]. Byeon et al. use an evolutionary approach, using genetic algorithms to help identify and remove noisy instances in their GAPS approach, allowing a C4.5 classifier to achieve better accuracy, especially at higher levels of noise [Byeon et al., 2008]. Later, this technique is extended into the comprehensive NDFS approach which also performs feature selection using a GA, improving classification performance substantially [Byeon, 2009]. Developing a similar method, Kim evolves the set of training instances and the connection weights of an artificial neural network simultaneously [Kim, 2006]. This method is employed specifically to reduce the effect of noisy data in the domain of financial data mining. García et al. also use a binary genetic algorithm for prototype selection in imbalanced domains. In their work, they present a method called Evolutionary Undersampling (EUS) which evolves the inclusion/exclusion of each instance as a binary allele. They compare the use of two fitness functions: one which is based upon the percentage of reduction and classifier accuracy of a 1-NN learner and another which is a factor of the relative balance of the prototype and the G-mean of a 1-NN learner. They use two different kinds of evolutionary algorithm, CHC and a technique called PBIL specifically for binary representations, and they compare the four genetic methods (each combination of EA and fitness function) with a variety of canonical undersampling methods in the imbalance literature as well as classic prototype selection algorithms DROP3 and IB3. They then compare the relative data set reduction and G-mean of classifiers constructed from the prototypes and find that the CHC method based on the G-mean and relative balance offers the best trade-off between classifier accuracy and reduction: while it is unmatched in classification performance, it is only worse than one other method at reducing the size of the data set [García et al., 2006]. The single method that provides better performance is, incidentally, the CHC technique whose fitness function selects for greater reduction. This work demonstrates that evolutionary techniques can provide very strong performance in general, and that additionally, they are flexible in

the ways in which they can alter the data (and consequently, the classifiers constructed from that data). The efforts of Byeon et al., Kim, and Garciía et al. provide the foundation for the methods presented in this thesis.

In other research, experimenters use a real-valued GA to perform attribute weighting for a k-NN classifier, effectively warping the instance space. Kelly and Davis attempted to use a straightforward real-value GA approach, and reports small improvements in classifier performance [Kelly Jr and Davis, 1991]. Later, Punch et al. implements a similar approach, although theirs incorporates a simultaneous feature extraction method by which three feature pairs were multiplied to produce new attributes [Punch III et al., 1993]. Which pairs of attributes were chosen, as well as weighting factors for those products, were evolved by the GA simultaneously. Using this method, they were able to classify a real-world data in a difficult, real-world domain (soil sample classification) with better accuracy than either naive k-NN classification or k-NN classification with binary GA feature selection. The second point in particular is of note, as the object of this experiment is also to determine whether diminishing the influence of an instance continuously might provide better performance than disregarding it entirely.

Evolutionary methods are also used for feature extraction. Moreno-Torres and Herrera use genetic programming (GP) to evolve a more descriptive feature from the set of attributes. Their method evolves expressions from the set of attributes (terminals) with a set of basic arithmetic operations to create new features. Their objective was to produce a new feature space so that the data would suffer less from overlap and data fracture, allowing for better performance from a classifier. Additionally, the new feature space was composed of two dimensions in order to provide easier visualization of the transformed instance space [Moreno-Torres and Herrera, 2010]. In contrast to the general method of Moreno-Torres and Herrera, Orlic and Loncaric use a genetic algorithm to evolve features specifically for the classification of difficult seismogram data [Orlic and Loncaric, 2010]. Feature extraction, the

authors explain, is a good approach because not only are the classes difficult to distinguish, but the instances have a waveform representation that poses challenges to conventional machine learning without some degree of processing. Although their method uses a fixed-length binary GA, in effect, they produce a similar effect to genetic programming by recombining relations, operations, and features of the data, evolving based on the performance of a very simple classifier. Additionally, each sample is assigned a weight (initially uniform weights of 1), and these are modified as the GA runs, emphasizing difficult examples.

Incorporating boosting in a genetic fuzzy rule base approach, Frank Hoffmann produces a successful classification method using evolution strategies (ES) [Hoffmann, 2001, Hoffmann, 2004]. Encoding the parameters of fuzzy rules as a vector of real numbers, Hoffmann constructs a rule base by iteratively evolving rules. Initially, all instances are weighted equally and the set of fuzzy classification rules is empty. An evolutionary approach is then used to evolve a new rule, this rule is entered into the rule set, and the performance of the rule-based classifier is tested. The training examples which are not classified correctly are given additional weight, and ES is used again to produce another rule. Due to the weighting, rules which perform better on the "difficult" examples are given evolutionary preference. In short, it is an evolutionary method that is very similar to conventional boosting in many ways. Eventually, a fuzzy rule base is assembled that was able to classify new examples. Özyer et al. use boosting in conjunction with a genetic fuzzy rule base scheme to construct a classifier for intrusion detection. Their method differs from that of Hoffmann primarily in that they use a genetic algorithm and their rule encoding is different. There are a few other minor differences, but the two techniques are fundamentally very similar. Testing their classifier against the contest-winning results on their data set, they report that it is competitive [Özyer et al., 2007]. Both of these methods are relevant to the proposed methods in this thesis because they use evolutionary methods to adjust weights on training instances. To a large degree, however, they resemble AdaBoost and its related approaches as they assemble

a rule base which behaves similarly to the ensemble of weak hypotheses. In contrast, the methods proposed in this thesis do not assemble a group of weak methods, and the weights evolved by the GA reflect relative overall emphasis rather than emphasis at a particular iteration in the algorithm.

Since cost-sensitive machine learning methods have proven effective in improving minority-class performance in imbalanced domains, ICET merits inclusion in this paper. Developed by Peter Turney, ICET is an extension of C4.5 that is sensitive both to test costs and varying misclassification costs for members of different classes [Turney, 1995]. As with AdaCost, the primary objective is not necessarily to reduce classification error but to reduce total cost; depending on the disparity of misclassification costs and the class distribution of the data, these objectives may come into conflict. In order to produce classifier that minimizes this total cost, Turney uses a genetic algorithm to evolve biases on each test so that the bias determines how likely the inductive tree-builder is to use it for classification. This is analogous to the work of Kelly and Davis or W.F. Punch et al. in that the attributes which compose instances are assigned relative weights, but ICET is much more advanced, is designed to account for the costs of finding different attribute values (i.e. conducting medical tests), and constructs a tree structure which more closely resembles an optimal plan for diagnosis than a vector of attribute weights. ICET is accordingly fairly sophisticated, and a full discussion of its operation is beyond the scope of this thesis. It did, however, perform very well at minimizing classification cost across a variety of medical data sets.

One recent approach to classifying imbalanced data sets consists of an evolutionary adaptation of the Nested Generalized Exemplar (NGE) classifier archetype. Although this is not a data pre-processing or weighting method, it deserves mention here as the underlying task is very similar to that of Kuncheva or others as it falls within the frameworklike instance selectionof data reduction [García et al., 2012]. NGE methods work by creating exemplars which model the data as a whole. Exemplars are composed of either single instances or hyper-

rectangles in the instance space (a hyper-rectangle is composed of a maximum value and a minimum value for each dimension). Unknown examples are classified either by the smallest hyper-rectangle which covers them or, if they are not covered by any hyper-rectangle, the nearest exemplar's label. The proposed method in this paper, EGIS-CHC, works by constructing a set of hyper-rectangles covering the data according to a simple heuristic. Then, the inclusion of each hyper-rectangle is determined by a binary-encoding evolutionary algorithm using the HUX recombination operator, which helps prevent premature convergence. No mutation operator is used, but upon convergence, the evolutionary algorithm restarts, using the best-seen solution to seed the new initial population (35% of the loci comprising the best chromosome are changed), so that diversity is restored without losing progress. Fitness of individuals is calculated as a function of the area under the receiver-operating characteristic curve (AUC) and the number of generalized examples included such that models which produce a high AUC and use few exemplars are considered most desirable. EGIS-CHC is compared against three other (non-evolutionary) NGE methods: BNGE, RISE, and INNER as well as two rule-induction methods: RIPPER and PART. Each technique is employed on the original data and separately with SMOTE pre-processing. The authors' evaluation takes into account thirty-six data sets in the KEEL repository and results are tested for statistical significance using the Wilcoxon signed-rank test. From this experiment, the authors find that EGIS-CHC is able to achieve a better AUC measure than the other methods evaluated, regardless of whether SMOTE was used. Furthermore, EGIS-CHC was also generally able to create models that were less complex (in terms of the number of rules or generalized examples required) than its competitors'. Finally, it is of note that, unlike the other techniques examined in the article, the performance of EGIS-CHC was hampered by the application of SMOTE. The proposed method in this paper provides strong evidence that evolutionary algorithm-based approaches can effectively partition instance space and locate concepts present in a data set.

2.3 Outlier Detection

The field of outlier detection is also relevant to the task of instance selection and weighting. Sensibly, outliers represent either noise (which would ideally be removed or diminished) or rare cases (which would be duplicated or emphasized). Hence identifying outliers effectively is important to the task of altering the distribution of data in a beneficial way, but doing so is a nontrivial problem, especially in domains with high dimensionality [Aggarwal and Yu, 2005]. Accordingly, the task has received substantial attention, and it is worthwhile to examine some of the methods developed in the course of this thesis; of especial interest are techniques that incorporate evolutionary components to guide the search for outliers.

One approach for detecting outliers or mislabeled data is proposed by Brodley and Friedl. They frame their research specifically in the context of machine learning, using machine learning techniques to filter the data and then using the performance of classifiers (measured by classification accuracy) built on the filtered data to gauge the effectiveness of their method. The authors compare three machine learning techniques: 1-NN, C4.5 decision trees, and linear machines. These algorithms are used to filter the data in three ways. First, a single technique can be used to filter the data and then develop a model from the filtered data. The other methods are ensemble methods in which all of the classifiers are construct a model and then remove instances according to either a majority vote or a unanimous misclassification. As might be expected, the majority vote scheme is more likely to remove misclassified instances (as well as non-noisy ones), and the consensus scheme is less likely to remove any instance from consideration. As a control, the classifier methods were used to learn from the data with no filtering. Additionally, an ensemble classifier consisting of the three methods was used as the classification method, with filtering performed by consensus, majority vote, and not at all to determine whether an ensemble for classification could replace a filter method. Using a number of real-world data sets with varying amounts

of artificially-introduced noise, the authors reach a number of valuable conclusions. First, filtering can be very beneficial for improving classifier performance in the presence of noise. While filtering did not produce improvements from data sets without artificial noise, it did produce improvements for noisy data sets, particularly as the amount of noise increased [Brodley and Friedl, 1999]. Furthermore, filtered data sets produced decision trees with substantially fewer nodes; for the data sets employed, the number was often reduced by at least a factor of two. With regards to which filter method was the most effective, majority-vote filters generally offered the best performance from the classifiers built on their data; the authors suggest that this is because it is, in general, more important to exclude bad examples than it is to include valid ones. The authors also find that the use of an ensemble classifier was not sufficient to replace filtering: not only did filtered ensemble learners perform better than unfiltered learners, but filtered single method learners were able to outperform unfiltered ensemble learners in many cases. As a side note, the authors did note some effects of class imbalance in the behavior of their techniques though they did not take any measures to correct it; this is especially worth noting since the authors use accuracy as their classifier performance measure. This experiment is relevant as it provides an application of outlier detection to (and by) machine learning and demonstrates that there are great potential gains, particularly for noisy data sets.

Crawford and Wainwright investigate using a genetic algorithm to search for a set of outliers. Citing the combinatorial explosion of possible noisy subsets, the authors argue that a GA could offer efficient and effective search. They experiment using different fitness functions to guide the genetic algorithm: three different pre-existing measures of the degree to which a set of points are outliers are used to search for (known) outliers in five data sets. They find that Cook's Squared Distance formula for multiple-case diagnostics generally produced the better performance than Least-squares or Andrews and Pregibon's diagnostic, but it had the notable drawback of being unable to adequately address cases in which there

were no outliers [Crawford and Wainwright, 1995]. Based on a small number of relatively small data sets, the authors make the claim that genetic algorithms are a promising method for outlier detection.

In similar research, Tolvi proposes using a genetic algorithm guided by an informational-based heuristic called the BIC. Unlike Crawford and Wainwright, this representation is a binary encoding in which each locus corresponds to a point in the data set, and its value indicates whether it is an outlier. This experiment consists of a general test for performance on two data sets as well as an investigation into the scalability of the algorithm in terms of the number of variables in the data as well as the size of the data set. The former tests indicate that the GA was able to locate the global optimum every time, though only two data sets were used [Tolvi, 2004]. The latter experiment indicates that the algorithm's time requirements grow more with increasing data set size than increasing numbers of variables. This is unsurprising, as the data set size dictates the dimensionality of the genetic representation. In spite of this drawback, Tolvi's experiment lends more support to the idea that genetic algorithms offer great potential gains for the task of detecting outliers.

Still another evolutionary approach to outlier detection is presented by Aggarwal and Yu, the focus of which is solving the curse of dimensionality. As the dimensionality of a space increases, so does the sparseness. Since most outlier detection schemes are based upon concepts of locality and distance, this is extremely problematic. Unfortunately, many of the most interesting or valuable domains have a high dimensionality relative to the number of data points in the space. Examining prior literature, the authors note that it is often the case that one can use lower-dimensionality projections to find outliers, circumventing the distance problem but introducing the problem of finding which projections to use. The goal is to find lower-dimensionality projections that are exceptionally sparse compared to the others, since they are more likely to contain outliers. The projection space is formed by constructing a discrete grid over the instance space such that the each dimension's intervals

contain an equal number of instances. Sparseness is measured by a value called the sparsity coefficient, which is based on the assumption of a uniform distribution. The authors note that while the assumption of a uniform distribution is unfounded, that this measure still provides an effective way of searching for outliers in practice. Unfortunately, as the dimensionality increases, the number of possible projections undergoes a combinatorial explosion, making a brute-force approach impractical. To make matters worse, such projections are relatively rare, making the problem, "Akin to finding a needle in a haystack" [Aggarwal and Yu, 2005]. Fortunately, genetic algorithms have proved effective in similarly difficult domains, and so the authors select the GA as their search method for finding projections which are likely to contain outliers. Each individual represents one possible region of the grid, each allele of which specifies either a grid restriction in that dimension or a "don't care" value. A list of the best (i.e. most sparse) areas found during the search is updated at the end of each generation, and the best areas overall are returned as the GA terminates. Noting that black-box GA applications often yield relatively poor results, the authors devise domain-specific recombination and mutation operators. The results from this experiment indicate that not only was the algorithm effective in finding regions which actually contained outliers, but its required time also scales approximately linearly in the number of dimensions, which is very good considering how the number of combinations grows [Aggarwal and Yu, 2005]. Aggarwal and Yu's method is the most effective application of evolutionary techniques to outlier detection of the literature surveyed here. This is most likely due to the framing of the problem, and it suggests that it may be more useful to address regions of instance space rather than individual instances. The method also likely benefits from the use of customized operators to facilitate navigating the space. Both of these points deserve consideration in designing an evolutionary method for similar problems.

Chapter 3

Proposed Methods

In this thesis, I evaluate the effectiveness of five different evolutionary methods for resampling data sets so that each instance in the training set is assigned an influence proportionated to its importance to the classifier. In three of these methods, the weights are evolved on the basis of individual examples. In the remaining two, the weights are evolved and applied on the basis of cluster membership. While the former allows finer tuning within the search space, it results in a much more difficult problem space for the genetic algorithm. The latter, on the other hand, provides much simpler search space, but can only operate on the instance space in relatively broad strokes. Additionally, it introduces new parameters related to clustering, and the additional computational expense of clustering itself. This is significant as the performance of the clustering determines the ability of the genetic algorithm to produce a beneficial weighting.

The intuition behind instance weighting is that some examples should influence the classifier's model of the target concept more than others. When minority class examples are assigned more weight in an imbalanced domain, classifiers trained on that data typically have higher recall. If the imbalance ratio is very high, classifying each of these minority instances correctly is vital to constructing an effective model of the data. In the same hy-

pothetical data set, many majority points may be discarded without negatively affecting classifier performance. Points in the majority class far from class borders do not need to have much influence in training as they do not contribute very much to the learning. Further, majority undersampling also reduces the storage and classification time required for k-NN classifiers and reduces the training/construction time for other paradigms. In a domain that also exhibits a high degree of overlap, however, which points should be emphasized is less clear: an instance surrounded by members of the opposite class could either be an important rare case or it could be noise. Through feedback from a classifier's performance on a validation set, a robust method such as the GA could discern which of these is the case, provided that there were sufficient information in the validation set to do so. By amplifying or diminishing the weight accordingly, the classifier's model reflects the importance or undesirability of that instance, respectively. In some ways, the methods proposed are an extension of previous work in which genetic algorithms have been used for instance selection. Allowing the pre-processor to assign varying degrees of importance will allow more nuanced alterations to the distribution of the data. Allowing the GA to duplicate data points multiple times permits larger changes to the distribution, which may be necessary for difficult data sets. Intuitively: it may be more beneficial to mitigate an instance's influence than to ignore it entirely, and experiments using random oversampling have shown that duplicating an instance can be more helpful than merely including it.

3.1 Measure of Merit

A single measure of merit is used in this experiment to both guide the genetic algorithm and evaluate the final performance of classifiers trained on the resampled data. The F-measure, or F-score, is a function of the precision and recall containing a parameter, β, whose value can emphasize either the precision or the recall. If the two factors are given equal importance,

β is assigned a value of 1.0; this has been taken as a default value by many authors, and the term F-measure often denotes the general F-measure with β set to 1.0. Here, this parameter is given a value of 2.0, meaning that false negatives will reduce the F-measure more than false positives, promoting higher recall at the expense of precision. In imbalanced domains, recall is often more important for constructing useful classifiers, and the F-measure is commonly used as a measure of classifier performance in domains with skewed underlying class distributions [Chawla et al., 2003, Han et al., 2005, Sun et al., 2007, He and Garcia, 2009, Denil and Trappenberg, 2010, Khoshgoftaar et al., 2011]. In the context of this experiment, the term F-measure will refer specifically to the F2-measure.

When classifier performance is measured across cross-validation folds, there are multiple ways to calculate the F-measure describing the classifier's performance on the data set as a whole. In this experiment, the F-measure is calculated by summing the true positives, false positives, true negatives, and false negatives over all folds and using these summed values to calculate the overall F-measure. This approach has been shown to be less biased than either taking the average of each fold's F-measure or using the average precision and recall over the folds to calculate the F-measure [Forman and Scholz, 2010]. Each of these calculations can produce different F-measure values, so it is important to select the one which offers the least biased measure of performance.

The F-measure is an indicator of a classifier's ability to correctly classify data, and hence to use the F-measure to evaluate a given resampling for a data set, a classifier must be used constructed from that resampling. For this purpose, a 3-NN classifier using the Mahalanobis distance is used to assign a fitness value to each individual (i.e. weighting) in the population. The classifier is constructed from a two-fold stratified sampling of the training data is constructed. Each fold is used as the training set and the testing set in turn, and the F-measure is calculated over both folds as described above. Each instance's weight is only considered as a voting weight for the classifier; it is not considered as a misclassification

cost. The nearest neighbor paradigm was chosen for the classifier as it is simple, does not require any training time, has reasonable classification time, and is used in similar work [Cano et al., 2003, García et al., 2008]. Another good approach would have been a decision tree learner, particularly for high-dimensional data sets, but due to the implementation here, it would have entailed substantial overhead (i.e. through calls to Weka).

3.2 Genetic Representations

In this thesis, three genetic representations are evaluated in which each allele corresponds to the weight to be applied to one instance (one-to-one representations). First, binary representations have been used successfully to perform beneficial instance selection [Kuncheva, 1995, Cano et al., 2003, Kim, 2006, Byeon et al., 2008, García et al., 2008, Byeon, 2009]. In order to justify the increased computational complexity of using real or integer-valued representations, it must be shown that these methods can provide greater improvements. The binary representation is thus included as a baseline. For this formulation,a genetic algorithm is used to evolve basic inclusion/exclusion of instances from the training set: a "0" in the gene represents that the corresponding instance should be excluded from training while a "1" indicates that it should be used to prepare the final classifier. This represents a genetic implementation of undersampling, a technique which in general has yielded strong performance in imbalanced and difficult domains [Drummond et al., 2003, Van Hulse et al., 2007]. While undersampling is typically only applied to majority-class instances, this genetic representation permits the removal minority-class examples. Even in imbalanced domains, removing minority class examples may produce a better model if those examples are noisy, so the ability to do so is included for flexibility. Since the fitness function is influenced more by recall than precision, eliminating non-noisy minority instances is sufficiently discouraged by the fitness function.

The two experimental one-to-one representations employed are real-valued and integer representations for emphasizing members of the training set. These methods represent different resampling approaches. The integer representation resembles an extension of the binary method in that while it is still capable of discarding instances, it can also duplicate them up to sixteen times. There remains contention in the literature as to whether oversampling or undersampling is more beneficial, as the former may lead to overfitting and the latter may discard important information. The integer representation is capable of determining which is more effective on a case-by-case basis and foregoing either undersampling or oversampling altogether. In practice, however, it is almost certain to oversample at least some members of the data set due to the much larger portion of the allele range devoted to oversampling. It is also of note that the integer weights are applied in two ways. For the MLP, SVM, and J48 classifiers, the allele value indicates the number of times the corresponding instance is to be included in the final training set. Thus the processed training set will almost certainly be much larger than the original data sets, though it will contain no new members. As it seemed less sensible to duplicate points for a k-NN classifier, the integer values are instead used as voting weights for the instances.

The real-valued representation is used to apply weights directly to the individuals, as many classifier paradigms can either accept weights or have variants which can. Being able to fine-tune the influence of each instance in the data set allows for a more nuanced way in which all the examples can contribute to the training of the classifier. Furthermore, weights for the instances range over the interval [0.0125, 4.0], allowing for substantial differences in influence, though no instance is ever completely ignored. This representation is more expressive than the integer representation, but with this expressiveness comes a substantially larger search space. Worse, this large portions of this search space are likely to be have little variation in the fitness function as the weights of safe instances (those which are far from class borders) are largely insignificant, and, for many cases, small differences in weights will

not produce any difference in the F-measure. To compensate for these additional difficulties, problem-specific genetic operators are employed; these operators are described below.

Additionally, two cluster-based representations are examined. For both of these methods, the data is partitioned so that each instance is assigned to a cluster. Twenty clusterings are produced by performing K-means clustering for twenty-four iterations, and the best clustering is used in the GA. The K-means algorithm is initialized each time using the Forgy Method, and the clustering quality is measured by the summed distance from each point to its center as measured by the Mahalanobis distance. The genetic algorithm evolves a real-valued weight for each cluster, and these weights are applied uniformly to the cluster members. The cluster methods are different in that the clusters are formed either within class boundaries or across the entire data set (i.e. clusters can be composed of members of both classes). These are referred to as within-class cluster weighting (WCC) and across-class cluster weighting (ACC), respectively. The former presents the obvious advantage of being able to adjust the weights of instances based on their class, which is a common and effective approach here as well as in the prior literature. In theory, it is more able to select sub-concepts within classes, even in contentious areas of the instance space. It does, however, introduce an additional cluster quantity parameter, as the number of clusters must now be set for both classes. Across-class clustering is implemented primarily for comparison.

3.3 Population Initialization

Population seeding is one method used to compensate for the chromosome length of the one-to-one genetic representations. The use of seeding can have a detrimental effect by reducing genetic diversity early in the search. It is implemented here because preliminary experiments indicated that the population required an exceedingly long time to converge on good solutions, likely due to the high dimensionality of the chromosomes. Where possible,

the seeding was designed to produce a population with as much diversity as possible while at the same time steering the search from suboptimal (albeit valid) solutions. Oversampling minority-class examples and undersampling majority-class examples have been shown to be effective in the literature, and the population initialization methods reflect this.

Since the central objective of these methods is to improve classification performance on minority-class examples, weights on the minority class instances are initially set higher than the majority class instances. For the binary representation, this is implemented simply by setting all minority weights to one (i.e. inclusion), while the inclusion of majority-class examples was decided randomly with even probability. For both integer-valued and real-valued one-to-one genetic algorithms, the populations were generated by assigning weights on a class-wise basis over two disjoint weight intervals. The minority examples are assigned weights in the higher half of the allele range (giving them more consideration), and the majority examples are assigned weights in the lower half (giving them less consideration in comparison). This ensures that the mean minority instance weight will be higher than the mean majority instance weight, but for any locus, the initial population should contain a variety of weightings. Thus the population is biased towards minority inclusion/oversampling/weighting without being totally reduced to a monoculture. If it is beneficial for a majority instance to be weighted heavily, the individuals with a relatively high gene value will proliferate, and mutation will allow the gene's value to migrate into the upper half of the range. Similarly, undesirable minority instances can attain relatively low weights.

The initialization of the cluster-based populations does not include any seeding because the relative importance of each cluster is not known *a priori*. Values are drawn for each allele independently from a uniform distribution over the range of possible cluster weights. This interval is the same as for the real-valued representation: [0.0125, 4.0]. Since the cluster-based representations are drastically shorter than the one-to-one representations, it is sensible to forego seeding.

Table 3.1: Genetic algorithm parameter configuration

Crossover Rate	1.0
Mutation Rate	0.0125
Population Size	400
Children per Generation	80
Maximum Evaluations (one-to-one)	64,000
Maximum Evaluations (cluster-based)	16,000
Fitness Termination Threshold	1.0

3.4 Genetic Algorithm Configuration and Operators

The mating selection operator, survivor selection operator, and termination conditions are the same for all five genetic representations. To ensure convergence and protect good solutions, the methods employ a steady-state genetic algorithm in which eighty new offspring are produced each generation. For mating selection, a binary tournament selection scheme is used to fill a mating pool of eighty parents. Offspring are produced by selecting two parents at random from the mating pool, recombining their chromosomes via crossover, and mutating the resulting two children. The eighty offspring then replace the eighty worst individuals in the population, and the next generational cycle begins. The genetic algorithm continues until either an individual produces an F-measure value of 1.0 (i.e. a perfect classification over both folds of the training set) or until an evaluation limit has been reached. For the one-to-one representations, this limit is 64,000. For the substantially shorter cluster-based representations, the evaluation limit is 16,000.

The crossover and mutation operators for the binary and integer representations are relatively simple. The binary GA uses canonical two-point crossover and bitflip mutation. The integer-valued GA also uses two-point crossover, but it employs creep mutation, changing the value of the allele by one. Creep mutation was chosen so that applications of the operator would produce relatively small changes in the weights (and thus in the classifiers trained on

the data sets pre-processed with those weights).

The real-valued GA incorporates the most sophisticated operators of the one-to-one representations. The real-valued GA uses one of two crossover methods, one of which is selected at random each time crossover is employed. Whole arithmetic crossover combines the values of two alleles; its α parameter, which controls the interpolation between chromosomes, is chosen randomly over the interval [-1.0, 2.0] for each application. Alternately, two-point crossover recombines the genes by position, preserving the allele values. The mutation operator for the integer representation is an ensemble method composed of three different kinds of mutation operator, one of which is chosen at random for each individual. The three mutation types are creep mutation, uniform mutation, and a novel operator called Tomek Mutation (described below). Creep operators were the most probable operation by a wide margin as they are neither disruptive nor they computationally expensive. The amount of creep was also subject to varying probability, as shown in the Table 3.2 below, with small mutations typically being more probable than large ones. Uniform mutation was given a very low probability which, combined with the low mutation rate, prevented this disruptive operator from being applied excessively. Tomek mutation was also applied sparingly because it is computationally intensive and because its problem-specific nature might cause the GA to founder on a local optimum. Tomek mutation is a representation-specific and problem-specific operator implemented to help navigate the wide space of real-valued weights. This method is based on the use of Tomek links as implemented in Kubat and Matwin's One-Sided-Selection and various other pre-processing methods [Tomek, 1976, Kubat et al., 1997, Batista et al., 2004]. It operates as follows: for each instance in the training set, with probability equal to the mutation rate, the method finds that instance's nearest neighbor. If the nearest neighbor is of the opposite class, then the pair of examples forms what is called a Tomek link [Tomek, 1976]. For each Tomek link, the positive (minority) example's allele value is increased by a factor of 1.40 and the negative example's allele value is decreased by a factor of 0.70. If the nearest

Table 3.2: Real-valued mutation operator

Mutation Operator	Probability of Use
Creep Mutation (0.1)	48%
Creep Mutation (0.05)	24%
Creep Mutation (0.5)	18%
Creep Mutation (1.0)	6%
Uniform Mutation	2%
Tomek Mutation	2%

neighbor is of the same class, the mutation operator does nothing. Thus, in principle, the boundary between the positive and negative classes is expanded to enlarge the positive class without negating the influence of the negative example too severely.

The operators for the cluster-based operators also vary in sophistication. For both clustering genetic algorithms, crossover is performed by the same crossover method as the real-valued, one-to-one GA: a simple ensemble of two which allows for both positional and value-based recombination. As above, the more difficult representation is given a more sophisticated mutation operator. For the GA built from within-class clusters, a single-valued, floating-point creep mutation is used. For the GA which utilizes clusters constructed across classes, however, an ensemble similar to that used for the real-valued, one-to-one GA encoding is employed (Table 3.3). Here, as there, it includes numerous creep increments and a specialized, representation-specific mutation operator. Instead of Tomek mutation, however, clusters' weights are altered depending on whether they consist primarily of majority-class instances or minority-class instances. If the cluster is composed mostly of instances in the majority class, its weight is decreased by a factor of 0.90; if it is a minority cluster, its weight is increased by 1.20.

Table 3.3: Across-class cluster weighting mutation operator

Mutation Operator	Probability of Use
Creep Mutation (0.1)	48%
Creep Mutation (0.05)	24%
Creep Mutation (0.5)	16%
Creep Mutation (1.0)	8%
Minority Cluster Mutation	2%

Chapter 4

Experimental Setup

4.1 Artificial Data Sets

The artificial data sets in this experiment based on a simple pattern: four coplanar Gaussian multivariate clusters forming an equilateral diamond with one corner at the origin. Gaussian distributions are used, as they appear frequently in artificial data sets throughout the literature [Japkowicz, 2001, Prati et al., 2004, Batista et al., 2005]. The cluster is comprised of a single class; the cluster at the origin and the cluster on the Y axis are both members of the positive (minority) class, while points generated from the two clusters on the sides are assigned to the negative (majority) class. In two-dimensions, the means of the clusters are (0,0), (0,16), (8, 8), and (-8, 8). For higher dimensionalities, the remaining attributes are generated about 0. Essentially, the clusters form a classic XOR pattern in the lowest two dimensions, rotated so that C4.5 classifiers do not encounter unnecessary difficulty. This particular shape was chosen as it made it easy to control the relevant factors for the experiment (overlap, imbalance) while resembling a real-world distribution arguably more closely than the uniformly-distributed backbone pattern or noisy linear border. The data sets were generated using the R statistical suite and the MASS package for R

Table 4.1: Artificial data set controlled parameters

Imbalance Ratio (Instance Quantities)	Standard Deviation	Dimensionality
3.0 (128+, 384-), 7.0 (64+, 448-), 15.0 (32+, 480-)	2.0, 3.0, 4.0, 5.0	2, 8, 16

[Venables and Ripley, 2002, R Core Team, 2013].

In total, thirty-six different artificial data sets were generated, varying degree of overlap, degree of imbalance, and dimensionality individually. Overlap was varied by setting the standard deviation of all the clusters uniformly; in the course of the experiment, values of 2, 3, 4, and 5 were used. Imbalance was measured by the Imbalance Ratio (IR), the ratio of the number of majority-class examples to the number of minority-class examples; values used were 3, 7, and 15. Evenly balanced class distributions were not used at all, since the purpose of the experiment was to primarily classify the experimental methods' performance over imbalanced data sets. Finally, data sets were generated in two, eight, and sixteen dimensions. In the case of the higher-dimensionality data sets, only the first two dimensions are relevant to the classification task. Each set consists of 512 points.

For each of the artificial data sets, eight-fold cross-validation was used to insure that performance on the artificial data sets was evaluated fairly. As mentioned above,two-fold cross-validation was used during fitness evaluation to split the training data further into training and validation sets, since there was no given validation set and it would have been unfair to allow the testing set to guide the genetic algorithm. In practice, it would likely benefit the method if a set of ideal training examples could be selected by experts to be included in the validation set. In particular, this would give one an opportunity to ensure that rare cases are represented in the guiding rubric. To test the method's performance in total generality, however, cross-validation is used to guide the genetic algorithm. Again, although the genetic algorithm is evolving weights for the entirety of the training set, when the fitness is

evaluated, the weights (or, in the case of the binary representation, exclusion) is not applied. Thus it is not possible for the genetic algorithm to improve the performance measure of its individuals by down-weighting/excluding difficult examples from the evaluation of that performance. It should be noted that stratified sampling is used at both levels of cross-validation so that the class ratios remain the same throughout and no training sets are produced that do not contain minority examples. Using a larger number of sub-folds for fitness evaluation would have reduced variability, but it also would have introduced higher computational cost and increased the absolute rarity of minority examples.

4.2 Real-World Data Sets

In order to evaluate the effectiveness of these methods, it is also important to examine their performance on real-world data sets. The University of California at Irvine maintains a repository of data sets collected from a wide array of research domains. Data sets from this repository are commonly used to gauge the effectiveness of machine learning techniques. The advantages of using this data are twofold. First, it provides an opportunity to demonstrate whether a method can be beneficially applied to problems which do not closely resemble the artificial data sets described above. Second, since many authors use these same data sets, they have become something of a benchmark for proposed methods. The methods examined in this thesis are applied to resampling the five data sets in Table 4.2. As this table suggests, the data sets were altered slightly. As in a number of other studies, classes were merged to create binary classification problems (Table 4.3). After merging, examples were discarded at random from each class so that the number of examples in each class would be a factor of sixteen. The data was culled so that each subfold would contain an identical, non-zero number of minority-class and majority-class instances).

Table 4.2: UCI data sets

Data Set Name	Size (after cull)	Minority Class (after cull)	Majority Class (after cull)	Imbalance Ratio
ecoli	336 (336)	112 (112)	224 (224)	2.0
iris	150 (144)	50 (48)	100 (96)	2.0
SPECTF	267 (256)	55 (48)	212 (208)	4.333
wdbc	569 (560)	212 (208)	357 (352)	1.692
yeast	1484 (1472)	244 (240)	1240 (1232)	5.133

Table 4.3: Modified UCI data set class composition

Data Set	Minority Class	Majority Class
ecoli	im, imU	cp, pp, om, omL, imL, imS
iris	iris-virginica	iris-setosa, iris-versicolor
SPECTF	0	1
wdbc	M	B
yeast	MIT	CYT, NUC, ME1, ME2, ME3, EXC, VAC, POX

4.3 Machine Learning Components

This experiment uses various sources for the implementation of the machine learning components. The Weka machine learning package is used widely throughout the machine learning community as it provides efficient implementations of many different methods in an extensive machine learning framework [Hall et al., 2009]. The Weka classifiers used to evaluate the effectiveness of the pre-processing methods are J48 (an implementation of the C4.5 induction tree learner) and the multilayer perceptron (MLP). J48 is used with Weka's default settings. During preliminary experiments, it was determined that the default configuration was artificially inhibiting the multilayer perceptron's classification. Consequently, the learning rate is set to 0.4 and the architecture consists of four nodes in a single hidden layer. The libSVM package is used to provide the support vector machine implementation used here [Chang and Lin, 2011]. Once again, preliminary experiments showed that it was necessary to set some parameters manually. Specifically, for the higher dimensionality data sets, it

was found that using a polynomial kernel produced much better results than using a radial basis function kernel, though the opposite was true when there were only two attributes. The kernel used is set accordingly for each data set. Finally, a k nearest neighbor (k-NN) classifier is used in these experiments; this is implemented by the author. It is used both in the wrapper evaluation and classifier evaluation sections with a k value of 3. The Mahalanobis distance is employed here to ensure that no attribute has disproportionate influence in calculating the distance between points.

Chapter 5

Results and Analysis

Given the number of data sets, the number of pre-processor/classifier combinations, the number of genetic runs, and the number of cross-validation folds used for each run, it is difficult to analyze the results of this experiment without condensing them. As described above, for each run, an aggregate F2-measure is calculated by using the summed true positives, false positives, true negatives, and false negatives over all folds. These form the basic measure of performance for each trial, and at no point in the analysis below is any attention given to the F2-measures of individual stratified sampling folds. Condensing the results further, the arithmetic mean of the aggregate F2-measures is used to assign a single value to each trial is taken to give a representative value for each method on each data set. This is useful for making general comparisons between resampling methods and between classifier paradigms. Since the choice of classifier makes a large difference in the benefit afforded by each pre-processing method, the discussion below addresses each machine learning archetype in turn and compares the relative performance of the evolutionary pre-processing techniques within the context of that archetype.

The Wilcoxon signed-rank test is employed to determine the comparative performance between methods over the entirety of the data set classes: artificial and UCI. This is use-

ful in that it determines whether, in general, the techniques proposed were able to make a difference in classifier performance over the original sampling. It also allows one to make pairwise comparisons between methods to determine whether the differences in improvement over the data sets were significant. The Wilcoxon signed-rank test is a pairwise significance test similar to the t-test with the additional benefit that it does not assume a Gaussian distribution. This measure is used widely throughout the literature to evaluate the performance of pre-processing techniques and learning algorithms. Examining the relative performances over the set of artificial data (i.e. where there are 36 samples), a critical value of 1.645 is used to ascertain whether the null hypothesis can be rejected with a confidence of at least 0.95. For the set of five UCI data sets, significance is calculated slightly differently. Five is the absolute minimum number of samples required to apply the Wilcoxon signed-rank test, and to reject the null hypothesis for five samples, one source's measurements must all be greater than the other's measurements.

5.1 3-NN Classifier Performance

Despite its simplicity, the 3-NN classifier is competitive with the more sophisticated classifier methods for the two-dimensional artificial data sets. The 3-NN classifier's performance is substantially diminished as the dimensionality increases, but this is a well-known effect called the curse of dimensionality. The higher dimensionality creates a higher degree of sparsity given a certain number of data points. This difficulty, however, presents an opportunity for the evolutionary pre-processing methods, and it is well-apparent in Table 5.1 that resampling provides substantial gains for the 3-NN classifiers for the data sets with eight and sixteen dimensions. In a practical application, high dimensionality would likely be remedied by performing feature selection. Although the unassisted 3-NN classifier suffers as the dimensionality increases, it has the advantage of being relatively resistant to both

overlap and imbalance when compared to the other classifiers. For all two-dimensional data sets with an imbalance ratio of 7.0 or 15.0 and relatively high overlap (standard deviations greater than or equal to 3.0), its unaided performance is the best, excepting one case where it is the second best.

The Wilcoxon signed-rank test in Table 5.2 shows the overall patterns over the artificial data sets. Integer-based resampling and WCC offer the greatest improvements for the 3-NN classifier. While WCC tends to offer better gains at lower levels of imbalance,the integral representation is better when the imbalance ratio is very high (15.0). Table 5.2 confirms that the superiority of these methods is significant, though neither has an advantage over the other. Offering a lesser degree of improvement, the binary undersampling and real-valued reweighting are significantly better than the control and ACC, significantly worse than integer-based resampling and WCC, and are approximately as good as each other. Finally, while ACC is worse than all the other evolutionary methods, it is still significantly better than the unassisted 3-NN classifier. Thus evolutionary pre-processing is very successful for the 3-NN classifier as all of the resampling techniques promote the construction of a better classifier.

Table 5.1: 3-NN classifier performance on the artificial data sets

	Control	Binary	Integer	Real	ACC	WCC
IR: 3, Std. Dev.: 2.0, 2 Dimensions	0.99372	0.9739	0.99468	0.9953	0.99244	0.9934
IR: 3, Std. Dev.: 2.0, 8 Dimensions	0.53966	0.64166	0.72846	0.67882	0.7519	0.7755
IR: 3, Std. Dev.: 2.0, 16 Dimensions	0.40598	0.54348	0.61642	0.5635	0.60204	0.65846
IR: 3, Std. Dev.: 3.0, 2 Dimensions	0.88606	0.87476	0.91066	0.90128	0.88778	0.90174
IR: 3, Std. Dev.: 3.0, 8 Dimensions	0.4422	0.5345	0.58512	0.56162	0.52918	0.6357
IR: 3, Std. Dev.: 3.0, 16 Dimensions	0.3132	0.49042	0.50848	0.46776	0.49904	0.56884
IR: 3, Std. Dev.: 4.0, 2 Dimensions	0.691	0.7194	0.76346	0.72378	0.67516	0.74796
IR: 3, Std. Dev.: 4.0, 8 Dimensions	0.37074	0.50728	0.57824	0.5212	0.48452	0.61956
IR: 3, Std. Dev.: 4.0, 16 Dimensions	0.25522	0.45754	0.49014	0.40306	0.37912	0.53992
IR: 3, Std. Dev.: 5.0, 2 Dimensions	0.4972	0.6023	0.62516	0.60566	0.51494	0.66466
IR: 3, Std. Dev.: 5.0, 8 Dimensions	0.20998	0.41544	0.43622	0.37292	0.2603	0.47384
IR: 3, Std. Dev.: 5.0, 16 Dimensions	0.1501	0.37488	0.41158	0.32596	0.2495	0.46024
IR: 7, Std. Dev.: 2.0, 2 Dimensions	0.99184	0.95372	0.98936	0.99252	0.9881	0.98628
IR: 7, Std. Dev.: 2.0, 8 Dimensions	0.30674	0.432	0.56774	0.479	0.4672	0.60482
IR: 7, Std. Dev.: 2.0, 16 Dimensions	0.2638	0.36364	0.41962	0.39374	0.3953	0.44566
IR: 7, Std. Dev.: 3.0, 2 Dimensions	0.75712	0.76816	0.82686	0.7973	0.75486	0.83328
IR: 7, Std. Dev.: 3.0, 8 Dimensions	0.15068	0.34644	0.39536	0.31136	0.3117	0.3869
IR: 7, Std. Dev.: 3.0, 16 Dimensions	0.08994	0.22844	0.23826	0.18654	0.16098	0.25042
IR: 7, Std. Dev.: 4.0, 2 Dimensions	0.65644	0.67686	0.7741	0.71266	0.6377	0.74786
IR: 7, Std. Dev.: 4.0, 8 Dimensions	0.09888	0.23922	0.22092	0.23024	0.19032	0.28896
IR: 7, Std. Dev.: 4.0, 16 Dimensions	0.05014	0.24648	0.24142	0.1608	0.1575	0.26024
IR: 7, Std. Dev.: 5.0, 2 Dimensions	0.25238	0.38756	0.43476	0.36842	0.2467	0.44652
IR: 7, Std. Dev.: 5.0, 8 Dimensions	0.05884	0.23808	0.27546	0.17218	0.14236	0.27734
IR: 7, Std. Dev.: 5.0, 16 Dimensions	0.04378	0.20596	0.22578	0.12592	0.08652	0.21938
IR: 15, Std. Dev.: 2.0, 2 Dimensions	0.97998	0.88026	0.97548	0.9655	0.96634	0.96768
IR: 15, Std. Dev.: 2.0, 8 Dimensions	0.09898	0.2542	0.37616	0.28416	0.22052	0.31044
IR: 15, Std. Dev.: 2.0, 16 Dimensions	0	0.17318	0.17698	0.10186	0.05094	0.12264
IR: 15, Std. Dev.: 3.0, 2 Dimensions	0.74794	0.69246	0.75634	0.75112	0.76886	0.73132
IR: 15, Std. Dev.: 3.0, 8 Dimensions	0.1925	0.23178	0.2596	0.2166	0.19354	0.21248
IR: 15, Std. Dev.: 3.0, 16 Dimensions	0	0.0871	0.0991	0.05538	0.05154	0.07004
IR: 15, Std. Dev.: 4.0, 2 Dimensions	0.30466	0.42982	0.50814	0.42814	0.3305	0.47126
IR: 15, Std. Dev.: 4.0, 8 Dimensions	0.03782	0.09742	0.0935	0.05524	0.04436	0.0677
IR: 15, Std. Dev.: 4.0, 16 Dimensions	0.02252	0.14376	0.17516	0.0625	0.04302	0.07814
IR: 15, Std. Dev.: 5.0, 2 Dimensions	0.40048	0.36116	0.37444	0.4141	0.35558	0.41758
IR: 15, Std. Dev.: 5.0, 8 Dimensions	0.01504	0.15064	0.1379	0.04022	0.04424	0.1132
IR: 15, Std. Dev.: 5.0, 16 Dimensions	0.05084	0.14002	0.14506	0.07674	0.05514	0.1333

Table 5.2: Wilcoxon signed-rank test for the 3-NN classifier's performance on the artificial data set

	Control	Binary	Integer	Real	ACC	WCC
Control		-	-	-	-	-
Binary	+		-	=	+	-
Integer	+	+		+	+	=
Real	+	=	-		+	-
ACC	+	-	-	-		-
WCC	+	+	=	+	+	

5.2 J48 Classifier Performance

The J48 implementation of C4.5 is very competitive compared to the other unassisted classifiers. One of its strengths is its relative resistance to degradation as the dimensionality increases. As noted above, the dimensionality of the artificial data sets is increased by adding attributes which do not affect the classification boundaries. For those attributes, majority and minority classes' samples are generated from the same distribution. Since the inductive tree classifier works by greedily selecting attributes which produce which contain the most information, they are able to ignore these extraneous features without any difficulty. In contrast, the 3-NN classifier used here must account for all features, and differences in the noisy features contribute to the distances between instances. Additionally, J48 is relatively resistant to the negative effects of imbalance and overlap, at least as they appear in the artificial data sets and in comparison with support vector machines and the multilayer perceptron.

Examining the performance increases offered by evolutionary pre-processing, it is clear that the inductive tree classifier benefits less than the nearest-neighbor approach. One reason for this is likely that the GA is guided by the performance of a nearest-neighbor algorithm, and thus the resampling developed may not be optimal for J48. This is also the case for

the multilayer perceptron and support vector machines. Comparing the relative value of the evolutionary resampling methods, once again the integer-based resampling approach and WCC emerge as the best methods. The Wilcoxon signed-rank tests in table 5.4 show that WCC is better than the control and all of the other methods except integer-based resampling by a significant margin. For J48, though, integer-based resampling is not as good as WCC. While binary undersampling and evolutionary real-valued weighting are, as stated above, significantly worse than WCC, they are not worse than integer-based resampling by a significant margin. Thus, among these four pre-processing methods, only WCC stands apart, but all of these methods are significantly better than the unaided classifier and ACC. ACC is able to offer relatively good improvements for some data sets (e.g. the two-dimensional data set with an IR of 15.0 and a cluster standard deviation of 5.0), but overall it offers comparable performance to the unmodified data set at considerable computational expense.

Table 5.3: J48 classifier performance on the artificial data sets

	Control	Binary	Integer	Real	ACC	WCC
IR: 3, Std. Dev.: 2.0, 2 Dimensions	0.93366	0.95208	0.95926	0.96184	0.96408	0.95384
IR: 3, Std. Dev.: 2.0, 8 Dimensions	0.95664	0.9479	0.96748	0.95036	0.97652	0.98656
IR: 3, Std. Dev.: 2.0, 16 Dimensions	0.93364	0.93342	0.91972	0.93568	0.91392	0.93442
IR: 3, Std. Dev.: 3.0, 2 Dimensions	0.86448	0.86068	0.86526	0.86384	0.84332	0.8569
IR: 3, Std. Dev.: 3.0, 8 Dimensions	0.84836	0.84394	0.8416	0.84856	0.80676	0.8473
IR: 3, Std. Dev.: 3.0, 16 Dimensions	0.87508	0.86614	0.86468	0.84832	0.86774	0.87184
IR: 3, Std. Dev.: 4.0, 2 Dimensions	0.64292	0.72908	0.6739	0.72768	0.63736	0.73252
IR: 3, Std. Dev.: 4.0, 8 Dimensions	0.73318	0.76118	0.736	0.73618	0.74514	0.75572
IR: 3, Std. Dev.: 4.0, 16 Dimensions	0.71176	0.74742	0.67158	0.71994	0.69878	0.74212
IR: 3, Std. Dev.: 5.0, 2 Dimensions	0.54308	0.62078	0.52418	0.60896	0.55088	0.68936
IR: 3, Std. Dev.: 5.0, 8 Dimensions	0.60662	0.6349	0.55228	0.60518	0.55388	0.65104
IR: 3, Std. Dev.: 5.0, 16 Dimensions	0.51524	0.58144	0.54762	0.57338	0.55858	0.61018
IR: 7, Std. Dev.: 2.0, 2 Dimensions	0.94492	0.93938	0.94062	0.95516	0.9489	0.9552
IR: 7, Std. Dev.: 2.0, 8 Dimensions	0.91708	0.92476	0.93752	0.92062	0.91452	0.92892
IR: 7, Std. Dev.: 2.0, 16 Dimensions	0.95948	0.96212	0.97012	0.95692	0.96452	0.96298
IR: 7, Std. Dev.: 3.0, 2 Dimensions	0.7144	0.7683	0.78252	0.7349	0.7267	0.785
IR: 7, Std. Dev.: 3.0, 8 Dimensions	0.76434	0.7219	0.7192	0.73936	0.74406	0.74286
IR: 7, Std. Dev.: 3.0, 16 Dimensions	0.69664	0.6827	0.67978	0.65574	0.66144	0.66738
IR: 7, Std. Dev.: 4.0, 2 Dimensions	0.58412	0.63986	0.62078	0.6376	0.62954	0.6694
IR: 7, Std. Dev.: 4.0, 8 Dimensions	0.5674	0.58428	0.62654	0.55956	0.57096	0.56596
IR: 7, Std. Dev.: 4.0, 16 Dimensions	0.53842	0.58884	0.4881	0.53618	0.5054	0.55566
IR: 7, Std. Dev.: 5.0, 2 Dimensions	0.266	0.36254	0.38654	0.38128	0.294	0.4557
IR: 7, Std. Dev.: 5.0, 8 Dimensions	0.48506	0.47828	0.49014	0.44542	0.4297	0.48394
IR: 7, Std. Dev.: 5.0, 16 Dimensions	0.31504	0.385	0.35492	0.35894	0.33964	0.38488
IR: 15, Std. Dev.: 2.0, 2 Dimensions	0.93404	0.9036	0.9271	0.9255	0.92152	0.90968
IR: 15, Std. Dev.: 2.0, 8 Dimensions	0.84882	0.87258	0.87352	0.87368	0.8491	0.85402
IR: 15, Std. Dev.: 2.0, 16 Dimensions	0.94022	0.90374	0.9192	0.90208	0.92768	0.91506
IR: 15, Std. Dev.: 3.0, 2 Dimensions	0.65176	0.69066	0.7154	0.70682	0.75294	0.7042
IR: 15, Std. Dev.: 3.0, 8 Dimensions	0.8292	0.73704	0.80458	0.76694	0.78478	0.7403
IR: 15, Std. Dev.: 3.0, 16 Dimensions	0.54916	0.56772	0.59472	0.58082	0.58566	0.58314
IR: 15, Std. Dev.: 4.0, 2 Dimensions	0.23034	0.5013	0.53994	0.48898	0.3991	0.47794
IR: 15, Std. Dev.: 4.0, 8 Dimensions	0.15408	0.30052	0.3515	0.26704	0.20398	0.31356
IR: 15, Std. Dev.: 4.0, 16 Dimensions	0.47464	0.42614	0.41336	0.48688	0.42686	0.45814
IR: 15, Std. Dev.: 5.0, 2 Dimensions	0.27942	0.40458	0.41946	0.42522	0.33114	0.3722
IR: 15, Std. Dev.: 5.0, 8 Dimensions	0.18774	0.19374	0.30734	0.21866	0.22622	0.23922
IR: 15, Std. Dev.: 5.0, 16 Dimensions	0.1818	0.18264	0.25572	0.1268	0.1692	0.22368

Table 5.4: Wilcoxon signed-rank test for the J48 classifier's performance on the artificial data sets

	Control	Binary	Integer	Real	ACC	WCC
Control		-	-	-	=	-
Binary	+		=	=	+	-
Integer	+	=		=	+	=
Real	+	=	=		+	-
ACC	=	-	-	-		-
WCC	+	+	=	+	+	

5.3 Multilayer Perceptron Classifier Performance

The unassisted mulitlayer perceptron classifier offers very competitive performance on the mildly imbalanced data sets, but for data sets where the imbalance ratio is above 3.0, its performance diminishes in comparison with support vector machines and J48. Like J48, it is less affected by increases in dimensionality than 3-NN. By reducing the weights on the input nodes corresponding to the noise dimensions, the multilayer perceptron is able to effectively mitigate their effect over the course of training. Ultimately, its difficulty with the higher-imbalance data sets (i.e. two thirds of the artificial data) means that it is not very strong as an unaided method in this experiment.

The MLP classifier's performance on the artificial data sets without resampling was lackluster compared to support vector machines and J48, evolutionary pre-processing improved its capabilities substantially. Two notable cases are the data sets with an imbalance ratio of 15.0 and a cluster standard deviation of 5.0 in two and sixteen dimensions. These two data sets are relatively challenging: they represent the extremes of both overlap and imbalance. Among the unaided classifiers, MLP achieves the second-lowest mean F2-measure. Pre-processing the data using integer-encoded resampling, however, yields the best results out of all classifiers and all pre-processing methods. While these examples demonstrate ex-

Table 5.5: Wilcoxon signed-rank test for the MLP classifier's performance on the artificial data sets

	Control	Binary	Integer	Real	ACC	WCC
Control		-	-	-	=	-
Binary	+		-	=	+	-
Integer	+	+		+	+	+
Real	+	=	-		+	-
ACC	=	-	-	-		-
WCC	+	+	-	+	+	

ceptionally dramatic increases in the F2-score, the multilayer perceptron generally benefits from the application of evolutionary pre-processing. Comparing the various evolutionary techniques, it is clear that integer-encoded resampling produces the best results by far. This is especially true for the highly imbalanced data sets. As in the case of the other two classifiers, WCC also provides relatively large gains compared to the other pre-processing methods and the control. For the MLP classifier, table 5.5 shows that the integer-encoded method is significantly better than WCC, and WCC is significantly better than the other methods. Real-valued evolutionary weighting and binary resampling offer commensurate, modest benefits for this classifier, and as in the case of J48, ACC does not produce a significant benefit for the classifier when analyzed in the context of the entire artificial set.

5.4 Support Vector Machines Classifier Performance

The support vector machines classifier performed relatively well on the eight and sixteen dimensional artificial data sets compared to the 3-NN and multilayer perceptron algorithms. Among the two-dimensional data sets, the SVM classifier did very, very poorly when the imbalance and overlap were both relatively high. One example of this is the two-dimensional data set with an imbalance ratio of 7.0 and a standard deviation of 5.0. For those data

sets on which the SVM classifier exhibited exceptionally poor performance, however, higher-dimensionality data sets with the same amount of imbalance and overlap did not pose exceptional difficulty. In fact, the SVM offered very competitive performance (e.g. the sixteen-dimensional data set with an imbalance ratio of 7.0 and a standard deviation of 5.0). This difference is likely attributable to the use of the RBF kernel function for the two-dimensional data sets and the polynomial kernel function for the higher dimensionality data sets. The fact that this effect is especially pronounced in the presence of both imbalance and overlap (especially overlap) suggests that the previously documented synergy between overlap and imbalance for SVM classifiers may also be partially responsible.

The SVM method did not benefit greatly from evolutionary pre-processing compared to the multilayer perceptron or the nearest-neighbor classifier. Small gains in performance were the norm, and the only large gains were in the aforementioned data sets where unassisted support vector machines failed to produce an acceptable classification. This is visible in table 5.6. Having noted this fact, comparing the methods' relative performance is still valuable. Here, as for the other classifiers, integer-based resampling is a relatively good pre-processing method. Although within-class cluster weighting was able to produce significant improvements for the other machine learning paradigms, SVM with WCC is not significantly better than unassisted SVM when examined over the array of artificial data sets. Table 5.7 shows that, for support vector machines, only binary undersampling and integer resampling are able to produce a significant improvement in classification in general, and neither of these two is significantly better than the other. Fortunately, none of the resampling methods significantly degraded the performance of the SVM classifier.

Table 5.6: SVM classifier performance on the artificial data sets

	Control	Binary	Integer	Real	ACC	WCC
IR: 3, Std. Dev.: 2.0, 2 Dimensions	0.99246	0.99624	0.98874	0.98874	0.9912	0.99246
IR: 3, Std. Dev.: 2.0, 8 Dimensions	0.98684	0.98258	0.98436	0.98464	0.98212	0.98936
IR: 3, Std. Dev.: 2.0, 16 Dimensions	0.94616	0.92564	0.94926	0.94902	0.9493	0.94528
IR: 3, Std. Dev.: 3.0, 2 Dimensions	0.83252	0.86392	0.86384	0.83518	0.8368	0.83356
IR: 3, Std. Dev.: 3.0, 8 Dimensions	0.70956	0.76896	0.72724	0.70386	0.71222	0.7138
IR: 3, Std. Dev.: 3.0, 16 Dimensions	0.8673	0.84778	0.86508	0.86116	0.8588	0.85964
IR: 3, Std. Dev.: 4.0, 2 Dimensions	0.57636	0.64972	0.63138	0.55474	0.55316	0.55848
IR: 3, Std. Dev.: 4.0, 8 Dimensions	0.62378	0.68744	0.62916	0.59144	0.61152	0.6218
IR: 3, Std. Dev.: 4.0, 16 Dimensions	0.6428	0.70082	0.6587	0.63744	0.65056	0.6417
IR: 3, Std. Dev.: 5.0, 2 Dimensions	0.35552	0.47968	0.47964	0.3509	0.37526	0.36812
IR: 3, Std. Dev.: 5.0, 8 Dimensions	0.4693	0.53768	0.47134	0.46392	0.44914	0.44948
IR: 3, Std. Dev.: 5.0, 16 Dimensions	0.47834	0.5765	0.48302	0.4765	0.46076	0.4722
IR: 7, Std. Dev.: 2.0, 2 Dimensions	0.9718	0.93974	0.96164	0.96672	0.96926	0.9724
IR: 7, Std. Dev.: 2.0, 8 Dimensions	0.94234	0.93824	0.9381	0.94838	0.95278	0.95032
IR: 7, Std. Dev.: 2.0, 16 Dimensions	0.9144	0.928	0.92886	0.92144	0.93346	0.929
IR: 7, Std. Dev.: 3.0, 2 Dimensions	0.60896	0.64384	0.701	0.5885	0.59406	0.60212
IR: 7, Std. Dev.: 3.0, 8 Dimensions	0.60978	0.6781	0.62958	0.60402	0.62164	0.62796
IR: 7, Std. Dev.: 3.0, 16 Dimensions	0.67866	0.62764	0.65478	0.6563	0.68094	0.66696
IR: 7, Std. Dev.: 4.0, 2 Dimensions	0.39956	0.43766	0.5472	0.40724	0.40882	0.40586
IR: 7, Std. Dev.: 4.0, 8 Dimensions	0.37526	0.49266	0.42028	0.38152	0.38222	0.38846
IR: 7, Std. Dev.: 4.0, 16 Dimensions	0.47592	0.53176	0.50858	0.4823	0.4676	0.48724
IR: 7, Std. Dev.: 5.0, 2 Dimensions	0.00764	0.11316	0.28174	0.0152	0.0076	0
IR: 7, Std. Dev.: 5.0, 8 Dimensions	0.3272	0.39806	0.36266	0.37294	0.36868	0.36198
IR: 7, Std. Dev.: 5.0, 16 Dimensions	0.25218	0.22548	0.26842	0.25124	0.2298	0.23486
IR: 15, Std. Dev.: 2.0, 2 Dimensions	0.75028	0.70446	0.7616	0.7616	0.75594	0.7616
IR: 15, Std. Dev.: 2.0, 8 Dimensions	0.90736	0.87114	0.926	0.90564	0.9096	0.90158
IR: 15, Std. Dev.: 2.0, 16 Dimensions	0.8341	0.74266	0.83646	0.84014	0.83726	0.81664
IR: 15, Std. Dev.: 3.0, 2 Dimensions	0.30286	0.22728	0.40688	0.28978	0.296	0.28914
IR: 15, Std. Dev.: 3.0, 8 Dimensions	0.67918	0.66652	0.68698	0.64836	0.68838	0.67602
IR: 15, Std. Dev.: 3.0, 16 Dimensions	0.42652	0.42042	0.41238	0.4071	0.42488	0.39424
IR: 15, Std. Dev.: 4.0, 2 Dimensions	0	0.00758	0.1145	0	0	0
IR: 15, Std. Dev.: 4.0, 8 Dimensions	0.24688	0.28746	0.24194	0.23714	0.24628	0.23834
IR: 15, Std. Dev.: 4.0, 16 Dimensions	0.22602	0.1483	0.19752	0.234	0.21994	0.1786
IR: 15, Std. Dev.: 5.0, 2 Dimensions	0.11378	0.09792	0.13242	0.11344	0.11378	0.10614
IR: 15, Std. Dev.: 5.0, 8 Dimensions	0.25094	0.2811	0.24208	0.25062	0.21676	0.209
IR: 15, Std. Dev.: 5.0, 16 Dimensions	0.14078	0.15236	0.1504	0.1167	0.12822	0.13112

Table 5.7: Wilcoxon signed-rank test for the SVM classifier's performance on the artificial data sets

SVM	Control	Binary	Integer	Real	ACC	WCC
Control		-	-	=	=	=
Binary	+		=	+	+	+
Integer	+	=		+	+	+
Real	=	-	-		=	=
ACC	=	-	-	=		=
WCC	=	-	-	=	=	

5.5 Effects of Imbalance and Overlap

The main reason that an array of artificial data sets was employed here was to be able to evaluate each evolutionary pre-processing method and each classifier paradigm while controlling data set features which have been shown to make constructing effective classifiers more difficult. This would allow one to determine where, if anywhere, each pre-processing method is most beneficially applied. Unfortunately, no overarching trend is easily discernible with respect to the relative benefit of pre-processing and either overlap or imbalance. In general, imbalance and overlap hinder classifier performance. Which classifier method will offer the greatest gains, however, does not seem to be predictable from the relative quantities of these two features in the artificial data sets. It may be the case that trends would emerge by examining more data sets with different distributions. It is possible that the absolute rarity of the minority class and the stochastic process of generating the data determine which data sets are difficult individually and which pre-processing methods work well on each individual data set. In that case, there may be larger trends with respect to the efficacy of each pre-processing method and the levels of imbalance and overlap that are simply not visible here.

Table 5.8: 3-NN classifier performance on the UCI data sets

	control	binary	integer	real	ACC	WCC
Ecoli	0.94274	0.92474	0.95228	0.95874	0.93656	0.94704
Iris	0.93932	0.91586	0.94444	0.9463	0.93676	0.94188
SPECTF	0.34264	0.42626	0.5031	0.4318	0.31666	0.55432
Wdbc	0.94532	0.92656	0.95026	0.94312	0.94864	0.95196
Yeast	0.52758	0.54132	0.59564	0.57464	0.53112	0.6007

Table 5.9: J48 classifier performance on the UCI data sets

	control	binary	integer	real	ACC	WCC
Ecoli	0.94374	0.94716	0.93328	0.94438	0.92594	0.94438
Iris	0.9224	0.919	0.9185	0.92522	0.90932	0.93412
SPECTF	0.3719	0.45304	0.40096	0.34822	0.34414	0.4975
Wdbc	0.92168	0.91326	0.92608	0.9263	0.91248	0.92876
Yeast	0.5122	0.54854	0.52146	0.53934	0.47198	0.58828

5.6 UCI Data Set Performance

The results on the UCI data sets resemble those from the artificial data sets. This is encouraging, as it suggests that the conclusions drawn from the analysis above may be applicable outside the scope of data sets resembling Gaussian clusters in an XOR pattern. When the unassisted classifier is able to achieve high performance, there is little, if any, gain from pre-processing. This is demonstrated by the Ecoli, Iris, and Wdbc data sets. On more difficult SPECTF set, all four classifier methods are improved by the use of pre-processing, and the greatest performance gains occur when binary undersampling, integer-based resampling, and WCC are used. For the yeast data set, which the unassisted classifiers also had difficulty, smaller gains are also visible, particularly from WCC and and integer resampling. Another notable feature of this set is the total failure of all SVM applications excepting the integer-based approach. This is almost certainly not a feature of the classifier but the result of poor SVM configuration for the data set.

Table 5.10: MLP classifier performance on the UCI data sets

	control	binary	integer	real	ACC	WCC
Ecoli	0.94846	0.94202	0.94672	0.94354	0.95832	0.94328
Iris	0.93254	0.947	0.94706	0.95296	0.93358	0.94182
SPECTF	0.35682	0.53638	0.48008	0.36804	0.43244	0.51274
Wdbc	0.95366	0.95694	0.95892	0.95722	0.95056	0.96042
Yeast	0.52974	0.57774	0.6245	0.5658	0.5233	0.61232

Table 5.11: SVM classifier performance on the UCI data sets

	control	binary	integer	real	ACC	WCC
Ecoli	0.78346	0.7426	0.92482	0.78656	0.78344	0.77876
Iris	0.96262	0.95454	0.94036	0.9626	0.94166	0.95686
SPECTF	0.3845	0.54196	0.44112	0.41506	0.42134	0.39616
Wdbc	0.69068	0.8445	0.66986	0.67418	0.68974	0.66714
Yeast	0	0	0.38392	0	0	0

Applying the Wilcoxon signed-rank test, no significant differences in classifier performance can be noted for any pre-processing method using either the unaided multilayer perceptrons the or unaided support vector machines. The former is more surprising than the latter, and it suggests that the multilayer perceptron may have had exceptional difficulty with the artificial data sets due to some property of the instance space. The 3-NN classifier, on the other hand, does benefit significantly from multiple methods as shown in table 5.12. As in the artificial data sets, integer resampling and WCC provide the best improvements: they are significantly better than the unaided classifier, ACC, and binary undersampling. The results of the Wilcoxon signed-rank test on tthe J48 classifier's results are shown in table 5.13. The J48 classifier benefits from within-class cluster weighting, which is able to augment the unaided classifier and offer significantly better performance than integer-based resampling. In contrast, across-class cluster weighting poses a significant hindrance to the classifier. The differences in the remaining methods' performances were not deemed significant.

Table 5.12: Wilcoxon signed-rank test for the 3-NN classifier's performance on the UCI data sets

	Control	Binary	Integer	Real	ACC	WCC
Control		=	-	=	=	-
Binary	=		-	-	=	-
Integer	+	+		=	+	=
Real	=	+	=		=	=
ACC	=	=	-	=		-
WCC	+	+	=	=	+	

Table 5.13: Wilcoxon signed-rank test for the J48 classifier's performance on the UCI data sets

	Control	Binary	Integer	Real	ACC	WCC
Control		=	=	=	+	-
Binary	=		=	=	+	=
Integer	=	=		=	+	-
Real	=	=	=		+	=
ACC	-	-	-	-		-
WCC	+	=	+	=	+	

Chapter 6

Conclusions and Future Directions

This thesis contributes to the development of evolutionary data resampling methods by proposing a variety of novel representations and evaluating their performance across data sets with varying dimensionality, imbalance, and overlap. Binary undersampling, integer resampling, real-valued weighting, and cluster-based weighting for clusters generated within and across class boundaries were compared against a baseline. For many data sets, especially data sets with high degrees of imbalance and overlap, training the classifier with the pre-processed data sets yielded dramatic improvements. The experiment presented in this thesis sheds light onto how these pre-processing methods are best applied, and it suggests some avenues for future research.

Among the evolutionary resampling methods evaluated here, the integer representation generally provides the greatest and most consistent improvements for classifiers trained on the artificial data sets. The benefits of this technique are particularly notable for 3-NN classifiers and multilayer perceptrons. Although one might anticipate that this method would cause detrimental over-fitting (as random oversampling does), this does not appear to be the case. Furthermore, integer-based resampling is successful despite the fact that it lacks any domain-specific operators. It is important to note, however, that using this

method causes the size of the training data to increase substantially. Consequently, the time required for classification increases for the 3-NN classifier, and the time required to construct (i.e. train) a model of the instance space increases for the multilayer perceptron. While this growth in the size of the training data imposes greater computational costs, it may be the source of the representation's success: by increasing the number of minority class samples, it eliminates the problem of absolute rarity identified by Japkowicz and Stephen as the most detrimental aspect of between-class imbalance [Japkowicz and Stephen, 2002]. The success of such a simple method lacking domain-specific operators is very encouraging, as there are many possible improvements to investigate. In order to avoid overfitting, instances could be oversampled not by direct duplication but by small, random perturbations. To reduce dimensionality, one could consider for resampling only those elements that are in a borderline region. Addressing only elements at risk for misclassification is a common tactic for pre-processing methods. To answer the problem of data set growth, it is possible to incorporate a term in the fitness function which penalizes exceedingly large data sets, introduce a local search element to attempt to reduce the sum of allele values without degrading the fitness, or introduce genetic operators which cause large allele values to decay if there is not strong selection pressure to maintain their presence.

Weighting the instances based on within-class cluster weighting also produced substantial gains for all of the classifiers except support vector machines. The 3-NN classifier especially benefits from WCC, particularly in learning the UCI data sets. The principal advantage of this method is the the size of the genetic representation, which allows for fast convergence and better efficiency. Unfortunately, the effectiveness of this method is limited by the quality of the clusters used to reduce the complexity of the data set, and producing representative clusters is itself a nontrivial problem. The optimal number of clusters and the placement of those clusters must be determined in advance, and there is no guarantee that a process which maximizes typical metrics of cluster quality (e.g. the Davies-Bouldin

index) will also maximize the quality of clusters for the purpose of assigning weights to their constituent instances. Hence, cluster-based representations present a promising direction for further development both in developing new representations and genetic operators, and also in improving the clustering process to be less reliant upon *a priori* information.

Determining the inclusion/exclusion of each instance using a binary evolutionary approach has also proven effective in the experiments here. In particular, this technique was beneficial for the SVM classifier where most other methods failed. The success of the binary representation is not entirely surprising since, despite its simplicity, similar techniques have proven very successful in other work [Cano et al., 2003, García et al., 2006, Byeon et al., 2008]. Furthermore, and in contrast with the integer-encoded genetic algorithm described above, any performance improvement from binary undersampling is accompanied by the secondary benefits of a smaller data set. For this reason, the binary one-to-one approach would be preferable to integer-encoded resampling for improving the performance of an SVM classifier or a C4.5 decision tree constructor, since they produce a commensurate improvement in the F2-measure. Given their inherent simplicity and the amount of previous work done with them, binary one-to-one representations offer relatively little in the way of further improvement, though this is not to say that they have been exhausted.

In spite of its flexibility and representational power, the real-valued weighting method did not prove particularly effective. This may be due in part to the finding that in some cases, resampling can be more effective than equivalent re-weighting, and it may be also due to a sub-optimal traversal of the real-valued, high dimensional search space (i.e. it could offer better performance with different genetic operators). At this point does not appear that real-valued one-to-one pre-processing methods are an especially promising area of research. Of course, the performance one real-valued GA scheme with one set of parameters is not a sufficient basis to infer anything about the effectiveness of methods with similar representations. The effectiveness of smaller and simpler representations, however, seems to

indicate that the greater expressiveness of this representation is not only unnecessary but may also be detrimental to converging on a useful resampling of the data.

In general, these experiments suggest that the evolutionary pre-processing techniques presented here are most applicable when the unaided classifier is unable to construct an effective model. The evolutionary methods examined above are not well-suited to correcting small failures. For data sets where the unaided classifier attained perfect or nearly-perfect classification results, resampled data sometimes produced less effective classifiers. Furthermore, even in cases where they performed poorly, support vector machines were not greatly improved by the pre-processing techniques proposed here. On the other hand, the methods showed that resampling is able to compensate for classifier failure attributable to a variety of causes. The pre-processors are able to compensate for poor classifier configuration, the curse of dimensionality, and difficulties which appeared to arise from the data set itself. This is encouraging, as configuring machine learning methods is often accomplished by trial and error, and methods that reduce the effect of poor configuration could prove very useful.

Finally, it is of note that in experiments such as the one presented here, where a variety of sophisticated methods are applied on many different problem instances, parameter configuration becomes a significant problem. With respect to the machine learning methods, the best configuration may vary from data set to data set. Similarly, the ability of genetic algorithms can be heavily reliant upon balancing exploration and exploitation with respect to the fitness landscape. In the former case, reasonably good configurations were sought by trial and error in preliminary experiments, but the search was not exhaustive and it did not address each data set individually. With respect to the evolutionary parameters, time did not permit extensive parameter tuning. Any of the evolutionary methods described above could, however, be extended by applying some form of parameter control. In short, any of the evolutionary techniques or any single application of a machine learning method may have suffered here from poor configuration.

Bibliography

[Aggarwal and Yu, 2005] Aggarwal, C. C. and Yu, P. S. (2005). An effective and efficient algorithm for high-dimensional outlier detection. *The VLDB journal*, 14(2):211–221.

[Autio et al., 2007] Autio, L., Juhola, M., and Laurikkala, J. (2007). On the neural network classification of medical data and an endeavour to balance non-uniform data sets with artificial data extension. *Computers in Biology and Medicine*, 37(3):388–397.

[Batista et al., 2004] Batista, G. E., Prati, R. C., and Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20–29.

[Batista et al., 2005] Batista, G. E., Prati, R. C., and Monard, M. C. (2005). Balancing strategies and class overlapping. In *Advances in Intelligent Data Analysis VI*, pages 24–35. Springer.

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

[Byeon, 2009] Byeon, B. (2009). *Enhancing the quality of high dimensional noisy data for classification and regression problems*. PhD thesis, University of Georgia.

[Byeon et al., 2008] Byeon, B., Rasheed, K., and Doshi, P. (2008). Enhancing the quality of noisy training data using a genetic algorithm and prototype selection. In *IC-AI*, pages 821–827.

[Cano et al., 2003] Cano, J. R., Herrera, F., and Lozano, M. (2003). Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study. *Evolutionary Computation, IEEE Transactions on*, 7(6):561–575.

[Chang and Lin, 2011] Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[Chawla et al., 2002] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

[Chawla et al., 2008] Chawla, N. V., Cieslak, D. A., Hall, L. O., and Joshi, A. (2008). Automatically countering imbalance and its empirical relationship to cost. *Data Mining and Knowledge Discovery*, 17(2):225–252.

[Chawla et al., 2003] Chawla, N. V., Lazarevic, A., Hall, L. O., and Bowyer, K. W. (2003). Smoteboost: Improving prediction of the minority class in boosting. In *Knowledge Discovery in Databases: PKDD 2003*, pages 107–119. Springer.

[Crawford and Wainwright, 1995] Crawford, K. D. and Wainwright, R. L. (1995). Applying genetic algorithms to outlier detection. In *ICGA*, pages 546–550.

[Denil and Trappenberg, 2010] Denil, M. and Trappenberg, T. (2010). Overlap versus imbalance. In *Advances in Artificial Intelligence*, pages 220–231. Springer.

[Drummond et al., 2003] Drummond, C., Holte, R. C., et al. (2003). C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on Learning from Imbalanced Datasets II*, volume 11. Citeseer.

[Fan et al., 1999] Fan, W., Stolfo, S. J., Zhang, J., and Chan, P. K. (1999). Adacost: misclassification cost-sensitive boosting. In *ICML*, pages 97–105. Citeseer.

[Forman and Scholz, 2010] Forman, G. and Scholz, M. (2010). Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD Explorations Newsletter*, 12(1):49–57.

[Freund et al., 1996] Freund, Y., Schapire, R. E., et al. (1996). Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156.

[Galar et al., 2013] Galar, M., Fernández, A., Barrenechea, E., and Herrera, F. (2013). Eusboost: Enhancing ensembles for highly imbalanced data-sets by evolutionary undersampling. *Pattern Recognition*.

[García et al., 2006] García, S., Cano, J. R., Fernández, A., and Herrera, F. (2006). A proposal of evolutionary prototype selection for class imbalance problems. In *Intelligent Data Engineering and Automated Learning–IDEAL 2006*, pages 1415–1423. Springer.

[García et al., 2008] García, S., Cano, J. R., and Herrera, F. (2008). A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41(8):2693–2709.

[García et al., 2012] García, S., Derrac, J., Triguero, I., Carmona, C. J., Herrera, F., et al. (2012). Evolutionary-based selection of generalized instances for imbalanced classification. *Knowledge-Based Systems*, 25(1):3–12.

[García et al., 2007] García, V., Sánchez, J., and Mollineda, R. (2007). An empirical study of the behavior of classifiers on imbalanced and overlapped data sets. In *Progress in Pattern Recognition, Image Analysis and Applications*, pages 397–406. Springer.

[Hall et al., 2009] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

[Han et al., 2005] Han, H., Wang, W.-Y., and Mao, B.-H. (2005). Borderline-smote: A new over-sampling method in imbalanced data sets learning. In *Advances in intelligent computing*, pages 878–887. Springer.

[He and Garcia, 2009] He, H. and Garcia, E. A. (2009). Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284.

[Hoffmann, 2001] Hoffmann, F. (2001). Boosting a genetic fuzzy classifier. In *IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th*, volume 3, pages 1564–1569. IEEE.

[Hoffmann, 2004] Hoffmann, F. (2004). Combining boosting and evolutionary algorithms for learning of fuzzy classification rules. *Fuzzy Sets and Systems*, 141(1):47–58.

[Holte et al., 1989] Holte, R. C., Acker, L., and Porter, B. W. (1989). Concept learning and the problem of small disjuncts. In *IJCAI*, volume 89, pages 813–818. Citeseer.

[Japkowicz, 2001] Japkowicz, N. (2001). Concept-learning in the presence of between-class and within-class imbalances. In *Advances in Artificial Intelligence*, pages 67–77. Springer.

[Japkowicz and Stephen, 2002] Japkowicz, N. and Stephen, S. (2002). The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449.

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

[Kelly Jr and Davis, 1991] Kelly Jr, J. D. and Davis, L. (1991). A hybrid genetic algorithm for classification. In *IJCAI*, volume 91, pages 645–650.

[Khoshgoftaar et al., 2011] Khoshgoftaar, T. M., Van Hulse, J., and Napolitano, A. (2011). Comparing boosting and bagging techniques with noisy and imbalanced data. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(3):552–568.

[Kim, 2006] Kim, K.-j. (2006). Artificial neural networks with evolutionary instance selection for financial forecasting. *Expert Systems with Applications*, 30(3):519–526.

[Kubat et al., 1997] Kubat, M., Matwin, S., et al. (1997). Addressing the curse of imbalanced training sets: one-sided selection. In *ICML*, volume 97, pages 179–186.

[Kuncheva, 1995] Kuncheva, L. I. (1995). Editing for the¡ i¿ k¡/i¿-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, 16(8):809–814.

[López et al., 2012] López, V., Fernández, A., Moreno-Torres, J. G., and Herrera, F. (2012). Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics. *Expert Systems with Applications*, 39(7):6585–6608.

[Moreno-Torres and Herrera, 2010] Moreno-Torres, J. G. and Herrera, F. (2010). A preliminary study on overlapping and data fracture in imbalanced domains by means of genetic programming-based feature extraction. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pages 501–506. IEEE.

[Nickerson et al., 2001] Nickerson, A., Japkowicz, N., and Milios, E. (2001). Using unsupervised learning to guide resampling in imbalanced data sets. In *Proceedings of the Eighth International Workshop on AI and Statitsics*, pages 261–265.

[Orlic and Loncaric, 2010] Orlic, N. and Loncaric, S. (2010). Earthquakeexplosion discrimination using genetic algorithm-based boosting approach. *Computers & geosciences*, 36(2):179–185.

[Özyer et al., 2007] Özyer, T., Alhajj, R., and Barker, K. (2007). Intrusion detection by integrating boosting genetic fuzzy classifier and data mining criteria for rule pre-screening. *Journal of Network and Computer Applications*, 30(1):99–113.

[Prati et al., 2004] Prati, R. C., Batista, G. E., and Monard, M. C. (2004). Class imbalances versus class overlapping: an analysis of a learning system behavior. In *MICAI 2004: Advances in Artificial Intelligence*, pages 312–321. Springer.

[Punch III et al., 1993] Punch III, W. F., Goodman, E. D., Pei, M., Chia-Shun, L., Hovland, P. D., and Enbody, R. J. (1993). Further research on feature selection and classification using genetic algorithms. In *ICGA*, pages 557–564.

[R Core Team, 2013] R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

[Seiffert et al., 2008] Seiffert, C., Khoshgoftaar, T. M., Van Hulse, J., and Napolitano, A. (2008). Resampling or reweighting: A comparison of boosting implementations. In *Tools with Artificial Intelligence, 2008. ICTAI'08. 20th IEEE International Conference on*, volume 1, pages 445–451. IEEE.

[Stefanowski, 2013] Stefanowski, J. (2013). Overlapping, rare examples and class decomposition in learning classifiers from imbalanced data. In *Emerging Paradigms in Machine Learning*, pages 277–306. Springer.

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

[Tolvi, 2004] Tolvi, J. (2004). Genetic algorithms for outlier detection and variable selection in linear regression models. *Soft Computing*, 8(8):527–533.

[Tomek, 1976] Tomek, I. (1976). Two modifications of cnn. *Systems, Man, and Cybernetics, IEEE Transactions on*, 6(11):769–772.

[Turney, 1995] Turney, P. (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research (JAIR)*, 2.

[Van Hulse et al., 2007] Van Hulse, J., Khoshgoftaar, T. M., and Napolitano, A. (2007). Experimental perspectives on learning from imbalanced data. In *Proceedings of the 24th international conference on Machine learning*, pages 935–942. ACM.

[Venables and Ripley, 2002] Venables, W. N. and Ripley, B. D. (2002). *Modern Applied Statistics with S*. Springer, New York, fourth edition. ISBN 0-387-95457-0.

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

Appendices

Appendix A

Data Resampling Techniques

[To be finished]

Appendix B

Data Set Performance Charts

[To be finished]