Evolutionary Instance Resampling for Difficult Data Sets

by

William Dale Richardson

(Under the Direction of Khaled Rasheed)

Abstract

In the field of machine learning, properties of data sets such as class imbalance and overlap often pose difficulties for classifier algorithms. A number of methods alleviate these difficulties by adjusting the distribution of the training data prior to classifier construction. Resampling is typically effected by weighting, removing, or duplicating instances, but finding a good resampling for the data set is a nontrivial problem. Genetic 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, evolutionary pre-processing is able to produce better classifiers. In particular, an integer-based, one-to-one representation and a cluster-based, real-valued weighting encoding are shown to improve 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

December 2, 2013

Dedication

This thesis is dedicated to Mr. Bryant Ivy Stiles, who taught me a great deal about Latin, history, culture, physical fitness, and life. It is difficult to overstate how much I have benefited from his tutelage and guidance, so at this milestone in my academic career, I would like to acknowledge his contribution. He has taught me the meaning of the phrase, "Et docere et rerum exquirere causas."

Acknowledgments

My mother, Sherry Richardson, and my father, Daniel Richardson, have been a source of unwavering support throughout my life. Needless to say, this thesis would not have been possible without them. At the University of Georgia, I would like to thank Dr. Khaled Rasheed for his direction and support as my academic advisor, and I would like to thank Dr. Walter Potter and Dr. Prashant Doshi for their feedback and patience in serving on my committee. 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. I would also like to express my gratitude toward my instructors at Washington and Lee University who helped prepare me for this work including Dr. Kenneth Lambert, Dr. Simon Levy, Dr. Rance Necaise, Dr. Sara Sprenkle, Dr. Joshua Stough, and Dr. Nathaniel Goldberg. Finally, I would like to thank my friends for their moral support, especially Peter Geiger, Greg Lennon, and E.W. Malachosky.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One objective central to machine learning is developing algorithms that inductively find patterns from examples and use those patterns to accurately predict attributes of unseen examples drawn from the same distribution. If an algorithm is unable to find effective patterns from a data set, the data set is said to be difficult for that method. A number of properties have been identified that can make data sets more difficult to model. One such property that has received a great deal of attention is between-class imbalance, or simply imbalance. A data set is 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 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 is often more disruptive for machine learning algorithms 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 many 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, because they are difficult to distinguish from noise or other statistically insignificant data, especially for methods that are biased toward maximum generality. Finally, the distributions of the training data and the testing data can be different enough to cause learning problems; this phenomenon is known as data fracture. The data sets available in a given problem may suffer from any combination of these features, and these features may appear in varying degrees. They may arise from shortcomings or difficulty 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 both can produce improvements [López et al., 2012]. One advantage of modifying the data set's distribution, however, is that it does not require one to select a machine learning paradigm in

advance. Having altered the data set, one can experiment with different learners to determine which one is best suited for the problem at hand. Modifying the training set's 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. Some methods employ synthetic data points to modify the distribution. The detection and removal of noisy instances has also been the subject of substantial research. 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, directing the inductive behavior of the classifier more 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 original data, and given a high level of noise, it may be impossible to discern an underlying concept at all. Even so, existing pre-processors are capable of data refinement and adjustment in ways that produce substantial classifier improvement, and they operate 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 important for constructing an effective classifier. For a pre-processor applied to such a data set, the inclusion, exclusion, duplication, or weighting of instances becomes more difficult accordingly. Pre-processing algorithms such as One-Sided-Selection, the Neighborhood Cleaning Rule, and SMOTE use simple heuristics to select which points in the data set should be discarded, duplicated, or used to create artificial instances. Random undersampling and random oversampling make these selections with no guidance at all. In either case, the search among 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 possibility space is even larger. As the search space grows, and exhaustive search becomes intractable, it becomes more important that the search method be efficient in navigating that search space.

Genetic algorithms (GAs) 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 distribution to facilitate classifier learning. Emphasizing examples or regions in the instance space can be effected in many different ways, and one strength of the GA is that it can accommodate many different representations within a powerful heuristic search framework. This versatility, however, is not without drawbacks. The genetic encoding and the fitness function determine the shape of the search space; it is important that they create a space that is easy for the algorithm to traverse. Furthermore, genetic algorithms are well-suited to a wide variety of problems, but they often offer poorer performance than custom-tailored algorithms designed for a specific problem. They have no guarantee of finding a global optimum, and they may require substantially more computation time than dedicated methods. Specialized operators that exploit domain-specific information can ameliorate this inefficiency, but ideal applications for genetic algorithms are problems where no adequate method exists. In the context of this research, this means that ideal data sets for genetic pre-processing 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 data set distributions to improve the quality of classifiers. Five different genetic encodings 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

2.1 Imbalance and Difficult Data Sets

Class imbalance has been identified as a hindrance to classifier performance in many real world domains including intrusion detection, oil-spill detection from satellite imagery, and medical diagnosis [Kubat et al., 1997, Autio et al., 2007, Chawla et al., 2008]. Although the negative effects of imbalance 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 subject, Japkowicz and Stephen perform a comprehensive experiment to measure the disruptive effect of imbalance. They study imbalance in isolation and in conjunction with two other properties 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 adjusting misclassification cost of minority examples by the imbalance ratio. Their experiments indicate that, of C5.0 decision trees, multilayer perceptrons, and support vector machines (SVMs), decision trees are the most negatively affected by imbalance, multilayer perceptrons are affected proportionately

to the degree of imbalance, and SVMs are virtually unaffected. Among remedial methods, oversampling the minority class 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 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 real-world data sets and gauge the benefit of a number of pre-processors. The authors find that, in general, oversampling is more effective than undersampling. They also find that imbalance is most problematic in conjunction with other complicating factors. 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 is 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 data pre-processing method that works by generating artificial examples to change the local distribution of instances. It was developed specifically to operate on imbalanced data sets as a form of oversampling that would improve recall without causing overfitting in the classifier. The algorithm operates by randomly selecting minority-class instances and interpolating at random intervals between those instances and their nearest minority-class neighbors. Its primary parameter is 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 the ROC 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, Batista et al., 2005, Han et al., 2005]. It has been widely adapted likely because introducing synthetic data points is 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 evolutionary methods examined in this paper is that data pre-processing can be applied in a wrapper framework so that the resampling is guided by some heuristic for classifier performance. Chawla et al. propose a SMOTE-based wrapper method to pre-process training 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. 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. Furthermore, wrapper adaptations of any method naturally require more computational time than the method itself. To offset this increased cost, the authors suggest that parallelization can be used to reduce execution time, if not the total computational burden, as the problem lends itself well to division (e.g. each fold can 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 its β parameter varies with the relative cost/imbalance. They find that the cost-insensitive measures produce good improvements, which is valuable because these methods are available in domains where the cost matrix is not known *a priori* or the misclassification costs changes over time. Among the cost-insensitive measures, the AUC performs at least as well as the F-measure, and Chawla et al. recommend it for guiding wrapper-based pre-processors [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. The authors 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 valuable to this thesis 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 in artificial data sets. These data sets consist of five-dimensional data points (not including the label) in two classes drawn from Gaussian clusters with a standard deviation of 1.0. Across these sets, the imbalance ratio varies from ninety-nine to one, and the distance between the means varies from nine standard deviations to zero (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 is more acute in the presence of imbalance [Prati et al., 2004]. Imbalance alone also hinders effective tree construction, but its effect in isolation is less pronounced than that of overlap. Since this thesis attempts to mitigate the difficulty of data sets, this article's insight into the relationship between two complicating features is very valuable.

García et al. empirically compare classifier paradigms' performance on imbalanced data sets with varying degrees of overlap. Their experiment also investigates the effects of having a globally imbalanced minority class be dominant within an overlapping region. Using a single nearest neighbor (1-NN) classifier, a radial basis function (RBF) network, a naïve Bayes classifier, a multilayer perceptron, a C4.5 tree learner, and support vector machines (SVMs), they examine the effect of increasing overlap with a constant degree of imbalance and training set size. To control the amount of overlap, the authors use artificial data sets: instances are generated from a uniform distribution covering rectangular areas. 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 [García et al., 2007]. When the distribution of data in the overlapping region is 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 properties 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 intelligently at a local scale.

Focusing on the behavior of SVM classifiers, Denil and Trappenberg investigate the effects of varying data set size, class imbalance, and overlap. They hypothesize that imbalance and overlap operate independently to inhibit SVM learning. From their experiments, however, 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 disrupt 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, small disjuncts and their effect 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 generality, 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 decided by 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 feature 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 simultaneously balances within-class imbalance and between-class imbalance. Nickerson et al. extend this method by augmenting 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 incorporation 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 is still 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 backpropagation 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]. The authors 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 between-class imbalance [Jo and Japkowicz, 2004]. They argue from this that remedying the small disjuncts problem should be the focus of research on improving difficult data sets. These works elucidate the relationship between imbalance and rare cases, and they are especially relevant to this thesis because using clustering to identify sub-concepts is essential to two of the pre-processing methods proposed 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. Data fracture is not a feature of any data set itself, but the authors argue that its effect can be especially pronounced in imbalanced domains [Moreno-Torres and Herrera, 2010]. They propose a feature extraction method for imbalanced data sets described in Section 2.3. Data fracture, also referred to as data-shift, receives relatively little attention compared to imbalance, overlap, and rare cases.

A number of studies compare the effectiveness of various pre-processing methods at countering complicating data features. 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 combined 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, SMOTE and SMOTE+ENN offers the best performance; in particular, SMOTE+ENN produces great improvement in overlapping domains. The authors suggest that the latter excels in overlapping sets because the application of ENN allows it to "clean" noisy border areas, making class boundaries easier to define.

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 (OSS), 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 network, RIPPER, a random forest classifier, a logistic regression learner, and a naïve Bayes classifier. Each classifier is used in conjunction with each pre-processor on every data set. The authors find that the resampling methods' results vary depending on the classifier used. They also find that the measures of merit rank the performances differently. In summary, though, they conclude that random undersampling is 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 pre-processing control). The failure of the non-random methods is remarkable because they were developed to address shortcomings in random oversampling and undersampling. This study covers a substantial group of data sets, but it does not give a thorough account of those sets' properties: 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 meth-

ods. If nothing else, 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. Furthermore, the existing "intelligent" operators have room for improvement.

López et al. present a comprehensive analysis of data pre-processing, cost-sensitive learners, and a combination of the two as 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 of SMOTE. When pre-processing and cost-sensitive methods are combined, the pre-processing techniques are applied to the data sets before they are modeled by 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 over either individually. As an additional contribution, the authors provide an extremely thorough review of the prior research on the relationship between imbalance and other aspects of data sets such as overlap and data-shift (previously referred to as data fracture here). 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. the authors observed that 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 performs a thorough experiment varying data set size, level 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 artificial data sets across all four variables and examines two other factors: boundary shape (linear and nonlinear) and rare cases located within majority-class regions. 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 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 because many artificial data sets used in previous experiments have linear class/concept borders [Japkowicz and Stephen, 2002, García et al., 2007, Denil and Trappenberg, 2010].

In the study's second section, Stefanowski evaluates the performance of various pre-processing methods for improving classifier performance and compares them to a proposed method. 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 can 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 pre-processors 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 pre-processors examined are capable of producing improvements. While random oversampling and cluster oversampling provide stronger performance on relatively simple data sets, NCR and SPIDER perform better on the more difficult data sets containing rare examples or overlap [Stefanowski, 2013]. In particular, SPIDER provides the best performance on difficult data sets. SPIDER and NCR both provide substantial gains in sensitivity, but SPIDER has a less adverse effect on specificity. SPIDER's superiority in this experiment 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.

2.2 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. More specifically, most references to "boosting" refer to extensions and adaptations of one metalearning method that assigns weights to instances: AdaBoost. Freund 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]. 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 hypothesis, 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 by the inductive method. 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. Freund and Schapire demonstrate AdaBoost effectiveness by applying it to prototype selection in the domain of character recognition. Boosting is relevant to this thesis because improving classification on difficult concepts and instances by resampling them is the purpose of this research.

In imbalanced domains, cost-sensitive classifiers can outperform their naïve counterparts [Japkowicz and Stephen, 2002, Chawla et al., 2008, López et al., 2012]. Fan et al. propose AdaCost, an augmentation of AdaBoost designed to provide improved classification in domains where misclassifications of some data are considered more costly (less desirable) than others; the domain of interest is the detection of credit card fraud. AdaCost proves effective, as it is consistently able to lower classification costs over the data without using additional computational power [Fan et al., 1999]. Since the objective is framed in terms of lowering the total misclassification cost, traditional classifier performance measures such as the AUC, F-measure, or G-mean, are not considered, but it is certainly possible to devise cost matrices to maximize those measures. The success of approaches that use cost-sensitivity to adjust for imbalance suggests 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 discourages learning models that include false negatives more than models that include false positives, resulting in better recall for the minority class. High minority-class recall contributes to the ultimate goal in this experiment: maximizing 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) classifier. To evaluate these boosting techniques, they apply them to four data sets from medical domains. Sun et al. observe that AdaC2 generally performs better than the other methods evaluated, and that it is more sensitive to the costs applied, making it more suitable for imbalanced domains [Sun et al., 2007]. This method is relevant because it is an application of boosting specifically for data sets suffering from imbalance.

Supporting the idea that boosting can improve classifier performance on imbalanced data, Chawla et al. augment AdaBoost with SMOTE to form SMOTEBoost [Chawla et al., 2003]. SMOTE, as described above, is a synthetic data generation technique for alleviating 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 is generated. The authors demonstrate that their method directs 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 performs better in the imbalanced intrusion detection domain than a 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 experiment demonstrates the effectiveness of boosting combined with pre-processing 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 with and without replacement. Two post-correction levels of imbalance are examined, so that after application, the data set is composed of either 35% or 50% minority examples for each boosting method. Thus eight different methods are used. By artificially altering the global level of noise, the imbalance level, and the class distribution of the noise in four UCI data sets, the authors compare their per-

formance across these factors. They 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 also find that 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 is virtually identical, though the versions without replacement offered better classification results than the versions with replacement. Between the two boosting methods, RUSBoost generally yields better results than SMOTEBoost. The authors explain the effect of noise on boosting by pointing out that noisy points typically remain difficult to classify, so as boosting progresses, they are given increasing amounts of weight. To demonstrate this effect, they present tables that show the progression of noisy examples' weights over time. From these results, 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 boosting literature to the evolutionary methods in this thesis is whether it is better to effect instance weights through direct weighting or probabilistic resampling. In the context of this paper, resampling does not take the general meaning used elsewhere in this thesis: here, each example has a probability of inclusion proportional to its assigned value. Seiffert et al. present an empirical study of this question, comparing a variety of boosting methods using both resampling and direct reweighting. The boosting algorithms examined are AdaBoost, AdaCost, AdaC1, AdaC2, AdaC3, CSB0, CSB1, CSB2, RareBoost, and SMOTEBoost. Since SMOTEBoost, as originally defined, does not have an apparatus for applying weights directly, the authors devise an adaptation. The area under receiver-operating characteristic curve and the area under the precision/recall curve are examined across fifteen data sets, and four classifiers are used (two configurations of C4.5, RIPPER, and a naïve 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 important 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, indicates 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 genetic algorithm (CHC) to evolve inclusion/exclusion of majority class instances for each boosting iteration. The fitness criterion for the genetic algorithm is a function of the G-Mean of a 1-NN classifier built from the included majority examples, 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 instance used in previous iterations of boosting and taking the maximum distance among these. The method is also applied without any diversity maintenance mechanism included at all. These three configurations of EUSBoost are compared, 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 is worse than EUSBoost, the difference is not statistically conclu-

sive 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). The kappa-error diagrams allow them to determine that while EUSBoost tends to produce better classification results, it does so at the expense of classifier diversity, at least 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) that modifies instance weights and directs classifier learning emphasis.

2.3 Evolutionary Methods for Improving Classification

A variety of existing approaches employ 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 here 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 because 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 called a prototype. In one early approach, Kuncheva uses a GA to select a reference set for a k-NN classifier from a set of training data. The primary objective is to remove redundant examples and thereby reduce the computational burden of classification.

The author uses an elitist generational genetic algorithm to evolve the inclusion/exclusion of each instance with a binary gene encoding [Kuncheva, 1995]. As her genetic operators, Kuncheva applies single point crossover to every mating pair and uses bitflip mutation to each gene with probability 0.05. Mating pairs are selected by choosing individuals randomly from the most fit half of the population, and survivors are chosen from both the offspring and the parents. She examines two different fitness criteria for constructing the prototypes. One is the error rate of a k-NN classifier using a leave-one-out validation scheme: the training set is also the validation set, but instances in the reference set are not allowed to classify themselves. The other fitness function is based on the sum of the number of neighbors that contribute to correctly classified instances' classification. She compares genetic algorithms using each fitness measure against two pre-existing prototype selection techniques: Wilson's editing and MULTIEDIT. Kuncheva shows that the GA guided by the second fitness function produces small improvements over the other methods. The genetic algorithm using the first fitness function also produces a prototype with better accuracy than non-evolutionary prototype-selection, but its effectiveness in guiding the GA decreases in later generations. The improvements from the evolutionary techniques are not deemed statistically significant, but they are an early demonstration that genetic algorithms could be suitable for the prototype selection problem, and this investigation forms the basis of later work.

Ishibuchi and Nakashima use a genetic algorithm to select instances and features simultaneously. Their goal is to find a reference set for a 1-NN classifier that minimizes the number of instances and attributes used while maximizing classifier accuracy. To this end, they propose a binary encoding such that each of the n features and each of the m instances in the training set are assigned a gene in a chromosome of length $m + n$. The genes that determine the features' inclusion occupy the first n positions. The authors evolve these chromosomes using a genetic algorithm with random mating selection, uniform crossover, and bitflip mutation. They determine by experimentation that biasing mutation towards exclusion (i.e.

making the transition from "1" to "0" more probable than that from "0" to "1") improves reference set reduction without compromising classification accuracy. Selection pressure is applied only through survivor selection, in which the most fit parents and children survive in a strategy analogous to $(\mu+\lambda)$ of evolution strategies. The authors compare two fitness measures. Both functions contain weighted terms for classifier performance, number of instances, and number of features; they differ in how they calculate the 1-NN classifier's performance. Performance is given by the accuracy in classifying each member of the training set using the reference set (and feature set) encoded in the chromosome. For the first fitness function, each instance in the training set is considered its own nearest neighbor if it is included in the reference set. In the other measure, instances cannot classify themselves (leave-one-out validation). Ishibuchi and Nakashima find that while the first measure of merit yields higher accuracy on the training data, the second produces reference sets that offer better accuracy for previously unseen data [Ishibuchi and Nakashima, 2000]. This suggests that using the training set as the validation set that guides the GA leads to overfitting, and the authors agree, concluding that the second fitness function offers better generalization. They compare their methods' evolved reference/feature sets against the full training sets with all attributes, and they observe that the GA using leave-one-out validation improves classifier accuracy. Additionally, by changing the weight terms in the fitness function, Ishibuchi and Nakashima find that the GA can effectively improve data set/feature set reduction at the expense of classifier accuracy, providing versatility in optimizing the objectives.

Cano et al. also propose and evaluate a number of evolutionary instance selection methods. In 2003, they examine a variety of non-evolutionary prototype selection algorithms as well as four different evolutionary algorithms: a generational GA, a steady-state GA, an algorithm called heterogeneous recombination and cataclysmic recombination (CHC), and population-based incremental learning (PBIL). They compare the amount of time required to pre-process the data, the percentage reduction the data set experienced, and the resulting

accuracy of classifiers built from the resampled data sets. Resampling is performed in the context of prototype selection (i.e. constructing a reference set for a 1-NN classifier) and in the context of training set selection for a C4.5 learner, a distinction made by the authors. Like Ishibuchi and Nakashima, Cano et al. use a fitness function which encourages data set reduction and classifier accuracy, though here it is restructured so that a single parameter determines the relative emphasis placed on the objectives. They also assess accuracy using a 1-NN classifier to assign labels to each instance in the training set with leave-one-out validation. From a variety of UCI data sets, the authors find that CHC is the most suitable technique for improving the performance of the 1-NN classifier and the C4.5 induction algorithm [Cano et al., 2003]. This algorithm is the fastest of the evolutionary methods and typically offers the best trade-off between accuracy and data reduction. CHC performs well for both prototype selection and training set selection tasks, while the traditional genetic algorithms experience difficulty for some of the larger training set selection problems. Some of the prototype selection methods produce reference sets with higher accuracies than CHC's, but they do so at a disproportionately high cost to data reduction. The authors also find that the non-evolutionary prototype selection methods do not perform well on the training set selection task. Thus an evolutionary technique provides greater versatility as well as better accuracy gains. Finally, in an analysis of the methods' behavior, Cano et al. make an important observation: while the non-evolutionary methods remove instances in accordance with underlying assumptions of their designers, CHC pragmatically selects those examples which are most important to raising the accuracy. Principled approaches maintain superfluous instances or discard relevant instances as their pre-defined emphasis on borderline or core instances dictate, often to the detriment of the reference set. The authors conclude that CHC demonstrates superior performance because it uses evolutionary principles to search among the possible subsets in a way that is unbiased by the geometry of the instance space [Cano et al., 2003]. This is one of the strongest arguments in the literature for the application

of evolutionary algorithms to resampling/instance selection.

García et al. also use a binary genetic algorithm for prototype selection; their work extends that of Cano et al. by applying it to imbalanced domains. They present a method called Evolutionary Undersampling (EUS) which evolves the inclusion/exclusion of each instance as a binary allele. They compare two fitness functions: one is based upon the percentage of reduction and classifier accuracy of a 1-NN learner and the other is a factor of the relative balance of the prototype and the G-mean of a 1-NN classifier. The G-mean is the geometric mean of the sensitivity and the specificity. This measure is commonly used in the context of imbalance because it is relatively unbiased by the class ratio. The authors use two different evolutionary algorithms: CHC and PBIL, 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 algorithm guided by 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 method at reducing the size of the data set [García et al., 2006]. The single method that provides better concentration is the CHC technique whose fitness function selects for greater reduction. This work demonstrates that evolutionary techniques can provide very strong performance in imbalanced domains.

One drawback of EUS is that it does not scale well. As the training set grows, the number of possible subsets increases exponentially. Furthermore, the length of the chromosome grows, which makes convergence more difficult for the evolutionary algorithms. To counter these effects, García et al. develop an evolutionary method that employs local search in a combination they term a "memetic algorithm." Once again, the domain is prototype selection, and so the objective remains simultaneously decreasing the size of the reference set and increasing accuracy for a 1-NN classifier. The local optimization is performed by search-

ing among the included instances in the chromosomes (those which have an allele value of "1") and examining the effect of excluding each of them in turn. Whether an instance is excluded (i.e. the chromosome is altered by optimization) is determined by the stage of the genetic algorithm; the local optimization alternates between two objectives during the evolutionary search. In one stage, the algorithm seeks to maximize classification accuracy and will not accept alterations that would diminish the fitness. In the other stage, the memetic component accepts small decreases in fitness to maintain genetic diversity and encourage data set reduction. The authors compare a steady-state memetic algorithm (SSMA) with a number of non-memetic evolutionary approaches: a generational GA, a steady-state GA, CHC, PBIL, and a generational GA with an intelligent crossover operator (IGA). Compared against these methods, SSMA produces better testing set accuracy, and García et al. attribute this difference to overfitting. For smaller data sets, CHC is a strong competitor taking into account both data reduction and test set accuracy measures, but SSMA offers much stronger performance on large data sets [García et al., 2008]. Additionally, the time required for pre-processing is drastically smaller for the SSMA than the all of the other evolutionary methods. The authors compare SSMA with several conventional prototype selection algorithms as well. This comparison shows that SSMA produces excellent test set accuracy; it performs significantly better than all methods in this respect except one: Explore. Compared to Explore, however, CHC offers much greater data set reduction. This study is important because it shows that while data set size can produce substantial difficulties for evolutionary prototype selection, the problem of scalability can be circumvented. Furthermore, evolutionary prototype selection techniques offer substantial improvements over conventional pre-processing.

Kim evolves the set of training instances and the connection weights of an artificial neural network simultaneously [Kim, 2006]. The genetic algorithm is applied to instance selection in the domain of financial forecasting. Instance selection here is intended to mitigate the

effect of noisy instances and shorten training times. Financial forecasting is an especially noisy domain, and the time required to train artificial neural networks is one of their greatest disadvantages as a machine learning paradigm. The connection weights are evolved using a genetic algorithm because it is a global search technique that may avoid local optima that can challenge locally-guided training algorithms like gradient descent. The author reports that the incorporation of instance selection into the genetic algorithm evolving the connection weights improved the neural network's accuracy for both the training and testing sets by a statistically significant margin. This study provides additional evidence that evolutionary instance resampling (specifically, instance selection) can be very beneficial in the field of machine learning.

Byeon et al. use genetic algorithms to help identify and remove noisy instances in their GAPS approach, allowing classifiers to achieve better accuracy [Byeon et al., 2008]. GAPS uses a binary genetic algorithm to evolve a subset of the instances which are likely noise. The authors then apply more conventional prototype selection to confirm which instances should be removed. To guide the genetic algorithm, Byeon et al. rely on the assumption that a model which represents the underlying concept will misclassify noisy instances. The fitness function uses the instances selected for inclusion to construct a C4.5 classifier; the fitness of each individual is proportional to the percentage of the supposedly noisy examples misclassified by the induction tree learner. To test the effectiveness of GAPS, the authors modify two data sets from the University of California, Irvine (UCI) machine learning repository, generating new labels for the instances based on a randomly constructed function. Noise is then introduced into these artificial labels, and GAPS is used to determine which of the instances are noisy. GAPS is compared against the RemoveMisclassified pre-processor included in the Weka machine learning suite; the two methods are used to filter data sets with six different levels of noise, and Platt's Sequential Minimal Optimization (SMO) learner is used to classify a test set from the same distribution. The authors show that their method is much

better at detecting noisy instances than RemoveMisclassified, especially at higher levels of noise. Later, the authors extend this technique into a comprehensive approach called NDFS [Byeon, 2009]. NDFS improves upon GAPS by simultaneously evolving a set of features using a separate GA. At regular intervals, the feature selection and noise selection methods exchange information. This exchange is important because as the feature set changes, so too may the appearance of noise for some instances, and which instances are included in the data set affects which features are superfluous. Examining this method in the context of both classification and regression, the Byeon finds that NDFS increases machine learning performance substantially by removing noise and eliminating superfluous attributes in the data.

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 attempt to use a straightforward real-value GA representation and report small improvements in classifier performance [Kelly Jr and Davis, 1991]. Later, Punch et al. implement a similar approach, although theirs incorporates a simultaneous feature extraction method in which three feature pairs are multiplied to produce new attributes [Punch III et al., 1993]. Which pairs of attributes were chosen, as well as weighting factors for those products, are evolved by the GA simultaneously. Using this method, they are able to classify a real-world data in a difficult, real-world domain (soil sample classification) with better accuracy than either naïve k-NN classification or k-NN classification with binary GA feature selection. The second point is important because the objective of this experiment is to determine whether diminishing or amplifying the influence of an instance to a variable degree might provide better performance than disregarding it entirely.

Evolutionary methods are also used for dedicated 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 easier classifier learning. Additionally, the new feature space is two-dimensional to provide easier visualization of the transformed instance space [Moreno-Torres and Herrera, 2010]. While the purpose of Moreno-Torres and Herrera's work is to propose a general technique, 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 for seismogram classification because not only are the classes difficult to distinguish, but the instances have a waveform representation that poses challenges for conventional machine learning without some form of processing. Although their method uses a fixed-length, binary-encoded GA, they produce behavior similar to that of 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 uniform weight that is modified as the GA runs, emphasizing difficult examples.

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 because this research is an extension of García et al.'s earlier work and because the underlying task falls within the framework, like prototype selection, of 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. The fitness of each individual is calculated as a function of the AUC and the number of generalized examples included so that models which produce a high AUC and use few exemplars are considered the most fit. 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 the results are tested for statistical significance using the Wilcoxon signed-rank test. The authors find that EGIS-CHC is able to achieve a better AUC measure than the other methods evaluated, regardless of whether SMOTE is used. Furthermore, EGIS-CHC is 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, unlike the other techniques examined in the article, the performance of EGIS-CHC is hampered by the application of SMOTE. The proposed method in this paper provides strong evidence that evolutionary algorithm-based approaches can effectively partition instance spaces and locate concepts present in data sets.

2.4 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

identifying outliers is a nontrivial problem, especially in domains with high dimensionality [Aggarwal and Yu, 2005]. Accordingly, outlier detection has received substantial attention, and it is worthwhile to examine some of the methods developed. Of especial interest are techniques that incorporate evolutionary components to guide the search for outliers.

Brodley and Friedl propose a relevant approach for detecting outliers or mislabeled data. They frame their research specifically in the context of machine learning, filtering the data with machine learning techniques and then using the accuracy of classifiers built on that 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, individual techniques are used to filter the data by classifying it. Misclassified instances are removed from the data set, and then a new classifier is constructed from the filtered data set. The other methods use ensembles in which all of the classifiers construct a model and remove instances according to either a majority vote or a unanimous misclassification. As might be expected, Brodley and Friedl find that the majority misclassification scheme is more likely to remove erroneous instances (as well as non-noisy ones), and the consensus scheme is less likely to remove any instance from consideration. Additionally, an ensemble classifier consisting of the three learners is used as the classification method, with filtering performed by consensus and majority vote to determine whether an ensemble for classification could replace a filter method. As a control, the classifier methods were used to learn from the data with no filtering. The authors use a number of real-world data sets with varying amounts of artificially-introduced noise to evaluate outlier detection. They find that filtering can be very beneficial for improving classifier accuracy in the presence of noise. While filtering does not produce improvements from data sets without artificial noise, it can assist the classification of noisy data sets, especially as the amount of noise increases [Brodley and Friedl, 1999]. Furthermore, altered data sets produce decision trees with substantially fewer nodes; for the data sets examined, the number is often reduced by

at least a factor of two. Majority-vote filters generally offer the best performance from the classifiers built on their data; the authors suggest that this is because it is generally more important to exclude erroneous examples than it is to include valid ones. The authors also find that the use of an ensemble classifier is not sufficient to replace filtering: not only do filtered ensemble learners perform better than unfiltered learners, but the filtered single-method learners outperform the unfiltered ensemble learners in many cases. As a side note, the authors do note some effects of class imbalance in the behavior of their techniques, but they do not take any measures to correct it, which is important because they use accuracy as their classifier performance measure. This experiment is relevant because it is an application of outlier detection to (and by) machine learning that demonstrates great potential gains, particularly for noisy data sets.

Crawford and Wainwright investigate using a genetic algorithm to search for a set of outliers. Highlighting the combinatorial explosion of possible noisy subsets, the authors hypothesize that a GA could offer efficient and effective search. They experiment using different fitness functions to guide the genetic algorithm: three 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. Crawford and Wainwright use an integer representation such that each integer corresponds to an index in the data set. The chromosome length is the number of suspected outliers, and the indices in each chromosome are sorted. They effect recombination by uniform, order-based crossover and replace duplicates with unique random values. Uniform mutation modifies the offspring so that each gene selected for mutation takes a new value. The authors find that Cook's Squared Distance formula for multiple-case diagnostics generally produces better performance than Least-squares or Andrews and Pregibon's diagnostic, but it has the significant drawback of being unable to address cases in which there are no outliers [Crawford and Wainwright, 1995]. From their results, the authors claim that genetic algorithms are a promising method for outlier detection.

In similar research, Tolvi proposes a genetic algorithm guided by an information-based heuristic called BIC. Unlike Crawford and Wainwright's methods, 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. His experiment evaluates the algorithm's performance and its scalability in terms of the number of data attributes and the size of the data set. The former tests indicate that the GA is very effective [Tolvi, 2004]. It located the global optimum every time, though only two data sets are used. The scalability experiments indicate 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 determines the chromosome length, which in turn determines the amount of time required for the GA to converge. In spite of its small array of data sets, Tolvi's experiment lends 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 heuristic for finding outliers in practice. Unfortunately, as the dimensionality increases, the number of possible projections undergoes a combinatorial explosion, making exhaustive search 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 problems, so the authors select the GA to search for projections that are likely to contain outliers. Each individual represents one possible region of the grid, each gene 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 regions overall are returned when 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 for this experiment indicate that not only is the algorithm effective in finding regions which actually contains outliers, but its required time also scales approximately linearly in the number of dimensions, which is very good considering how rapidly 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 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, five different evolutionary methods resample data sets so that each instance in the training set is assigned influence proportionate to its importance to a classifier. Three of these methods evolve weights for individual examples. The other two evolve and apply weights on the basis of cluster membership. While the former group allows finer tuning within the resampling space, it results in a much larger 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 resampling 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 all of these minority instances correctly is vital to constructing an effective model of the data. Under the same conditions, 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 concept definition. As an additional benefit, majority undersampling reduces the storage and classification time required for k-NN classifiers and reduces the training/construction time for other machine learning paradigms. In domains that also exhibit 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 where sufficient information in the validation set to do so. By amplifying or diminishing the weight accordingly, the classifier's model will reflect 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 provides for 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 under-represented classes and concepts. Intuitively, it may be more beneficial to mitigate an instance's influence than to remove 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 impor-

tance, β is assigned a value of 1.0; this has been taken as a default value by many researchers, 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 (F2-measure), 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-score 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 refers specifically to the F2-measure.

$$\text{F-measure} = (1 + \beta^2)\frac{Pr \cdot Re}{(\beta^2 \cdot Pr) + Re}$$

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 that offers the least biased measure of performance.

Since the F-measure is an indicator of a classifier's ability to correctly classify data, a classifier must be constructed from a resampled data set to evalutate it with the F-measure. A 3-NN classifier using the Mahalanobis distance is used to assign a fitness value to each individual (i.e. sampling) in the population. The classifier is constructed and evaluated using a two-fold stratified sampling of the training data. Each fold is used as the training set and

the testing set in turn, and the F-score 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. 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. The nearest neighbor paradigm was chosen for the classifier because it is simple, does not require any training time, has reasonable classification time, and is used in similar work [Cano et al., 2003]. Another good classifier for guiding the search would have been a decision tree learner due to their fast training and classification times [Byeon et al., 2008].

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; these are referred to as the 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 encodings 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 indicates that the corresponding instance should be excluded from training while a "1" indicates that it should be used to construct the final classifier. This is a genetic implementation of undersampling, a technique which has yielded substantial improvements for classification 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. The fitness function is influenced more by recall than precision, so eliminating non-noisy minority instances is discouraged by the fitness function.

The two novel one-to-one representations employed are integer and real-valued representations for emphasizing examples in the training set. These methods represent different resampling approaches. The integer representation is an extension of the binary method: 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 instance-by-instance 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 because a much larger portion of the allele range is devoted to oversampling. 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 values. As it is 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 either accept weights by default or have variants that can. Being able to fine-tune the influence of each instance in the data set allows all of the examples to contribute to classifier training in a more nuanced way. 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 represen-

tation, but with this expressiveness comes a substantially larger search space. Worse, large portions of this search space are likely to 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 in Section 3.4.

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 executing 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 cluster 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 differ 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 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 one-to-one genetic representations' chromosome length. Seeding can have a detrimental effect on the genetic algorithm's ability to traverse the search space by reducing genetic diversity prematurely. It is applied 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. The seeding is designed to produce a population with as much diversity as possible while directing the search away 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 primary objective of these methods is to improve classification performance on minority-class examples, weights on the minority class instances are initially set higher than those on 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/exclusion of majority-class examples is decided randomly with even probability. For both integer-valued and real-valued, one-to-one genetic algorithms, the populations are 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 is higher than the mean majority instance weight, but the initial population should contain a variety of values for each locus. 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 proliferate, and mutation allows 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.

3.4 Genetic Algorithm Configuration and Operators

The mating selection, survivor selection, and termination conditions are the same for all five genetic representations. A population of four hundred individuals is evolved in each run. This length was chosen to ensure that there would be sufficient genetic diversity for the GA representations with long encodings. To promote convergence and protect good solutions, the methods employ a one-fifth generation gap genetic algorithm so that 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, 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 (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

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

integer-valued GA also uses two-point crossover, but it employs creep mutation, changing the value of mutated genes by unit increments. Creep mutation was chosen so that applications of the operator would produce relatively small changes in the weights (and thus in the classifiers built using 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 applied. 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. Extending the α range beyond [0.0, 1.0] encourages exploration of the search space. Alternately, two-point crossover recombines the genes by position, preserving the allele values. The mutation operator for the real-valued representation is an ensemble method composed of three different kinds of mutation, of which one is chosen at random for each individual. The three types of mutation are creep, uniform, and a novel operator called borderline mutation. Creep operators alter the value of a gene by incrementing or decrementing the allele by a relatively small amount. Creep mutation is the most probable operation by a wide margin as it is neither disruptive nor computationally expensive. The amount of creep is subject to varying probability, as shown in Table 3.2: small mutations are typically more probable than large ones. Uniform

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%

mutation is given a very low probability which, combined with the low mutation rate, prevents this disruptive operator from being applied excessively. Borderline mutation is also applied sparingly because it is computationally intensive and because its influence might cause the GA to founder on a local optimum.

Borderline mutation is a problem-specific operator implemented to help the genetic algorithm navigate the deep space of real-valued weights. This method is based on the use of Tomek links 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 the instance's nearest neighbor. If the nearest neighbor is of the opposite class, then 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 neighbor is of the same class, the mutation operator does nothing. The borderline pairs selected in this manner are not necessarily Tomek links, but the intuition behind this operator is that the borderline pairs will work in a similar fashion. In principle, the boundary between the positive and negative classes is shifted to enlarge the influence of positive class without diminishing the influence of the negative example too severely.

The operators for the cluster-based operators vary in sophistication. For both clustering

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%

genetic algorithms, crossover is performed by the same crossover method as the real-valued, one-to-one GA: a simple ensemble of two which provides both positional and value-based recombination. As above, the more difficult representation is given a more sophisticated mutation operator. For WCC, a single-valued, floating-point creep mutation is used. ACC employs an ensemble similar to the one used for the real-valued, one-to-one GA encoding. The operation probabilities for ACC's mutation operator are shown in Table 3.3. Here, as there, it includes numerous creep increments and a specialized, representation-specific mutation operator. Instead of using borderline mutation, however, clusters' weights are altered depending on whether they consist primarily of majority-class instances or minority-class instances. For each gene, with probability equal to the mutation rate, if the corresponding cluster is composed of mostly majority-class instances, its weight is decreased by a factor of 0.90. If it is a predominantly minority-class cluster, its weight is increased by 1.20. This places more emphasis on the underrepresented class instances.

Chapter 4

Experimental Setup

To conclude that the proposed methods are effective, it must be shown that classifiers built from pre-processed data are more effective than classifiers trained on the raw data set. Four algorithms representing different machine learning paradigms are used to model numerous data sets, and the performance of these unaided learners forms a baseline. The data sets are then pre-processed using evolutionary methods, and the same four algorithms are trained using the altered data. In both cases, examples from the original data set are withheld from training (and pre-processing) to be used as a testing set. The examples in the testing set ensure that the algorithm has learned the underlying distribution from the original data set as a whole (as opposed to that of the training set). The models' ability to predict these instances' classes is used to estimate how well they will classify other unseen examples. For these estimations to be useful, the data sets must resemble problems to which the pre-processing methods might be applied. Artificial data sets are used in this experiment to evaluate pre-processor performance while controlling complicating factors: imbalance, overlap, and dimensionality. Data sets taken from other fields provide a glimpse into the methods' performance in real-world domains and a basis for comparison with other techniques.

4.1 Artificial Data Sets

The artificial data sets in this experiment are generated from a simple pattern: four coplanar Gaussian multivariate clusters forming an equilateral diamond with one corner at the origin. Gaussian distributions are used to generate the sub-concepts because they appear frequently in artificial data sets throughout the literature and correspond to many observed phenomena [Ishibuchi and Nakashima, 2000, Japkowicz, 2001, Prati et al., 2004]. Each 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. Essentially, the clusters form a classic XOR pattern in the lowest two dimensions; Japkowicz uses similar XOR patterns in one study on within-class and between-class imbalance [Japkowicz, 2001]. In two dimensions, the means of the clusters are (0,0), (0,16), (8, 8), and (-8, 8). For higher dimensionality sets, the remaining attributes are generated about 0.0. A more traditional XOR pattern is aligned with the axes, but C4.5 classifiers can encounter unnecessary difficulty if the information gain along each axis is zero. Thus the pattern is rotated by 45° to avoid setting the inductive tree learner at an unfair disadvantage. The data sets were generated using the R statistical suite and the MASS package for R [Venables and Ripley, 2002, R Core Team, 2013].

In previous research that also used artificial data, one alternative to the XOR shape is the "backbone" pattern. In the "backbone" pattern, class concepts form stripes that alternate by class along a single axis of the instance space [Japkowicz and Stephen, 2002, Denil and Trappenberg, 2010]. This arrangement is favorable because it allows the researchers to easily and intuitively vary overlap, imbalance, class size, and concept complexity. Its greatest drawback is its distinctly artificial quality. Linear decision boundaries are not expected in real-world domains, and research has shown that nonlinear boundaries are typically more difficult for classifiers to learn [Stefanowski, 2013]. The Gaussian XOR pattern

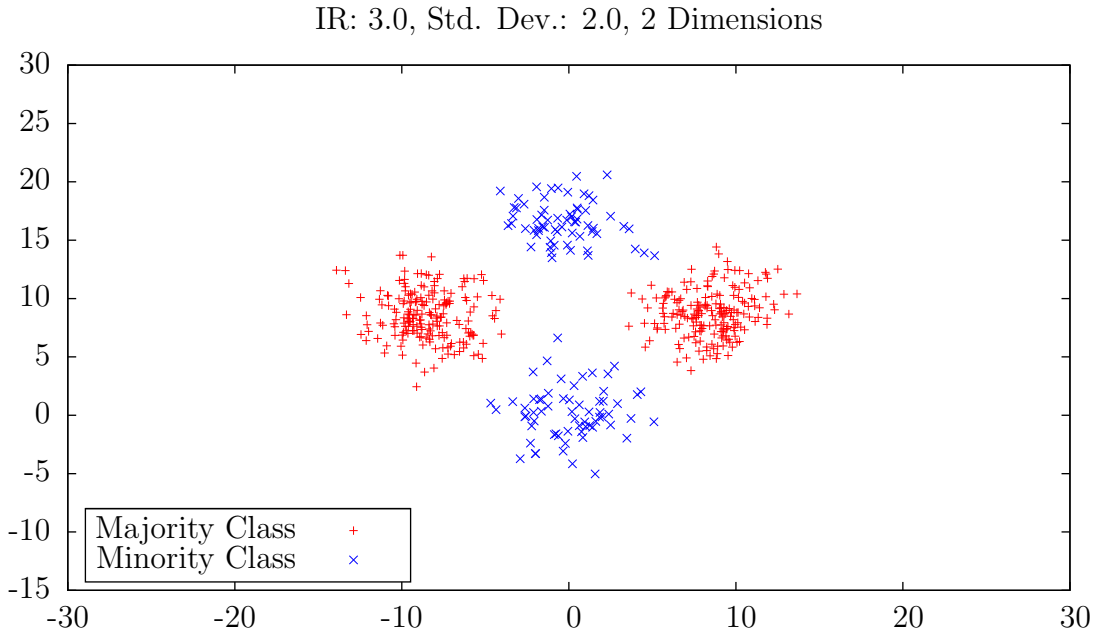


Figure 4.1: "Easy" artificial data set

was selected because it also provides simple control over the relevant factors for the experiment (overlap and imbalance) while arguably resembling a real-world distribution more than the "backbone" pattern.

In total, thirty-six different artificial data sets are generated with varying degree of overlap, degree of imbalance, and dimensionality. Overlap is varied by setting the standard deviation of all the clusters uniformly; in the course of the experiment, values of 2.0, 3.0, 4.0, and 5.0 are used. Smaller standard deviations produce tighter clusters with cleanly separable boundaries while the larger values produce diffuse clusters with boundaries that are difficult to discern. Imbalance is measured by the Imbalance Ratio (IR), which is the ratio of the number of majority-class examples to the number of minority-class examples; sets are generated with imbalance ratios of 3.0, 7.0, and 15.0. Evenly balanced class distributions (where the IR is 1.0) are not used at all because the purpose of the experiment is primarily to

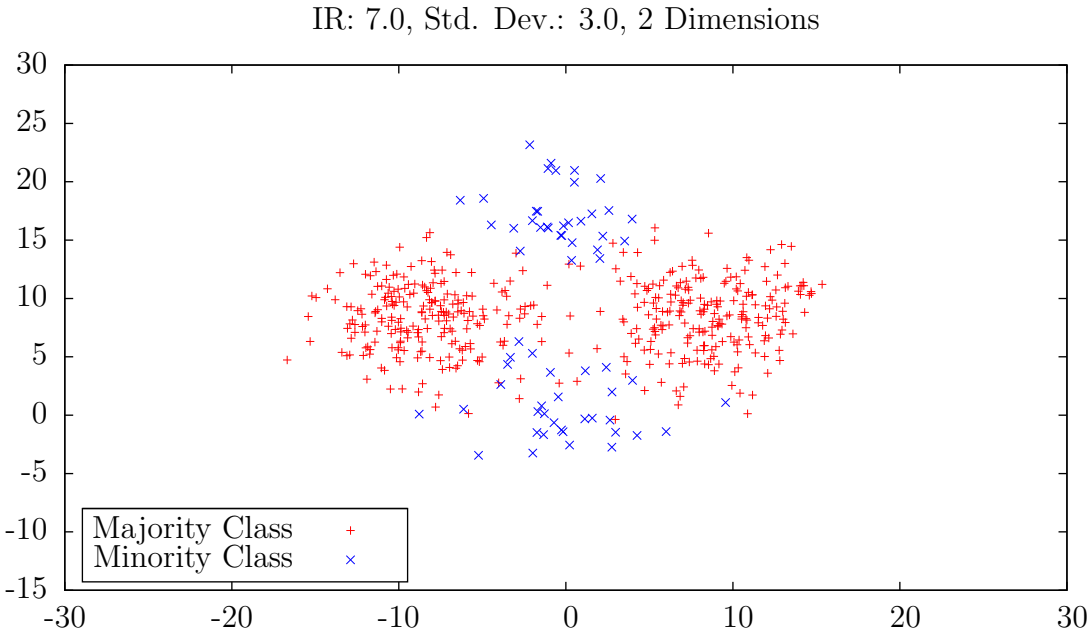


Figure 4.2: "Moderate" artificial data set

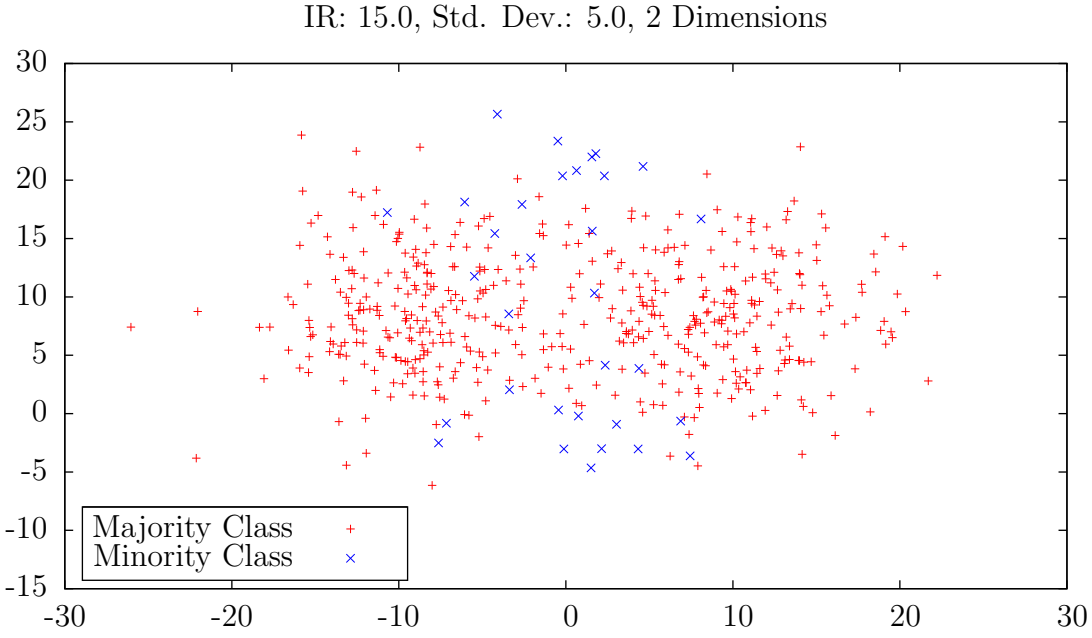


Figure 4.3: "Difficult" artificial data set

Table 4.1: Artificial data set 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

classify the experimental methods' performance on imbalanced data sets. Finally, data sets are 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 because this size accommodates high levels of imbalance without producing stratified sampling folds devoid of positive (minority) instances. Larger sets, which would have accommodated even higher levels of imbalance, would have caused substantial increases in computational time. Since this increase would have occurred for all thirty-six configurations, the total increase in computational load would have been too great given the time and resources available.

It is important to use separate instances to construct and evaluate the classifier. For singular data sets, it is common practice to partition the examples into disjoint training and testing sets randomly. The training set is typically larger than the testing set and is used to construct a model of the data set as a whole. The model's predictions are compared against the known labels of each example in the testing set, and the learner's performance is evaluated based on its ability to predict those labels correctly. If there are not enough examples to use separate training and testing sets, it may be necessary to use a technique called cross-validation. In cross-validation, the data set is partitioned randomly into subsets called folds. Each fold is reserved as the testing set in turn while the union of the other folds is used for training. Once every fold has been used for evaluation, some form of aggregation produces a single measure that is representative of the learner's performance on the data set as a whole. If folds are selected in such a way that the relative class composition is

maintained, this strategy is called stratified sampling (e.g. if class "A" represents 30% of the total data set, it will represent 30% of each fold).

In some cases, it may be necessary to retain a third portion of the data set for intermediate evaluation that guides the learner; a set of examples for this purpose is called a validation set. For instance, one circumstance in which a validation set is often employed is detecting whether an artificial neural network has begun to overfit its training data: if the accuracy observed for the training data increases with successive epochs, but the accuracy for the validation set is decreasing, then it is likely that the neural network has begun overfitting. The validation set serves as a sample that is representative of the whole data set but not necessarily the training set. Since the validation provides information that is used to train the learner (if indirectly), it is important that it be separate from the testing set; at the same time, since it is being used to evaluate the model, it must not be used directly for training. In the context of this thesis, the GA requires a set of instances to evaluate the performance of each resampling, and this set must be distinct from the final testing set.

For each data set in this experiment, eight-fold cross-validation is used to ensure that performance on the artificial data sets is evaluated fairly. Two-fold cross-validation is used during fitness evaluation to split the training data further into training and validation sets, since there is no validation set, and it would have been unfair to allow the testing set to guide the genetic algorithm. In application, the method would likely benefit if a set of ideal training examples were selected by experts for inclusion in the validation set. This would provide an opportunity to ensure that any rare concepts are represented in the fitness measure. The genetic algorithm is not given any such information in this experiment, however, because such information will not necessarily be available in practice. To test the pre-processors' performance in total generality, cross-validation over the training set is used to guide the genetic algorithm. Stratified sampling is used at both levels of cross-validation so that the class ratios remain the same throughout, and all of the training sets produced contain

minority examples. Using a larger number of sub-folds for fitness evaluation would have reduced variability, but it also would have raised the computational cost and increased the absolute rarity of minority examples.

4.2 Real-World Data Sets

To evaluate the effectiveness of these methods, it is also important to examine their performance on real-world data sets. The University of California, Irvine maintains a collection of data sets from a wide array of research domains [Bache and Lichman, 2013]. Data sets from this repository are frequently 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 the table suggests, the data sets are altered slightly. As in a number of other studies, classes are merged to create binary classification problems (Table 4.3). After merging, examples are discarded at random from each class so that the number of examples in each class would be a factor of sixteen. The data is culled so that each sub-fold would contain an identical, non-zero number of minority-class and majority-class instances). Which examples are omitted from each UCI data set is given in Appendix C.

Table 4.2: UCI data sets

Data Set Name	Size (after cull)	Minority Class (after cull)	Majority Class (after cull)	Imbalance Ratio
E. Coli	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
E. Coli	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 its 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 Quinlan's 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 [Chang and Lin, 2011]. Once again, preliminary experiments indicated that it was necessary to set some parameters manually. For the higher-dimensionality data sets, a polynomial ker-

nel produced much better results than using a radial basis function (RBF) kernel. For the two-dimensional data sets, more individualized configuration was required to produce acceptable levels of performance from the learner. Appropriate settings for the low-dimensional classifiers were found by trial-and-error. Finally, the k-nearest neighbor (k-NN) classifier used in these experiments is implemented by the author. It is used both in the wrapper evaluation and classifier evaluation sections with a k value of 3. Using a 1-NN classifier would have made the classifier more vulnerable to noise, and using larger values of k would have increased the bias towards the majority class in contested spaces. This classifier uses the Mahalanobis distance instead of the Euclidean distance so that no attribute has disproportionate influence in calculating the distance between points.

Chapter 5

Results and Analysis

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 results in 33,620 performance evaluations, or 820 evaluations per data set. This does not include the evaluations used to guide the genetic algorithm or the aggregate F2-scores from combining the cross-validation folds. It is difficult to analyze the results of this experiment without condensing them. As described in Section 3.1, for each run, an overall 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 using only these combined F2-scores reduces the number of values per data set to 120: five for each pre-processor/learner combination. To condense the results further, the arithmetic mean of the aggregate F2-measures is used to assign a representative value for each combination on each data set. This is useful for making general comparisons between resampling methods and between classifier paradigms; these values are plotted for each artificial data set in Appendix A and each UCI data set in Appendix B.

The Wilcoxon signed-rank test is employed to determine the comparative performance between pre-processors over the separate data set classes (artificial and UCI). This is useful

because it determines whether, in general, the techniques proposed were able to make a difference in classifier performance. It also allows one to make comparisons between methods to determine whether the differences in improvement over the data sets were statistically significant. The Wilcoxon signed-rank test is a pairwise statistical 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 reviewed here to evaluate the performance of pre-processing techniques and learning algorithms. Examining the relative performances over the set of artificial data (where there are 36 samples), if the test produces a value greater than 1.645, 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.

The artificial data sets are presented first because they compose the greater part of the experiment. 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, comparing the performances of the evolutionary pre-processing techniques in the context of each paradigm. This is followed by a brief analysis of the pre-processors' efficacy with respect to imbalance and overlap. Finally, the classification results for the UCI data sets are presented.

5.1 3-NN Classifier Performance

Despite its simplicity, the 3-NN classifier is competitive with the more sophisticated learners for the two-dimensional artificial data sets. The 3-NN classifier's performance diminishes as the dimensionality increases, but this is a well-known phenomenon called the curse of

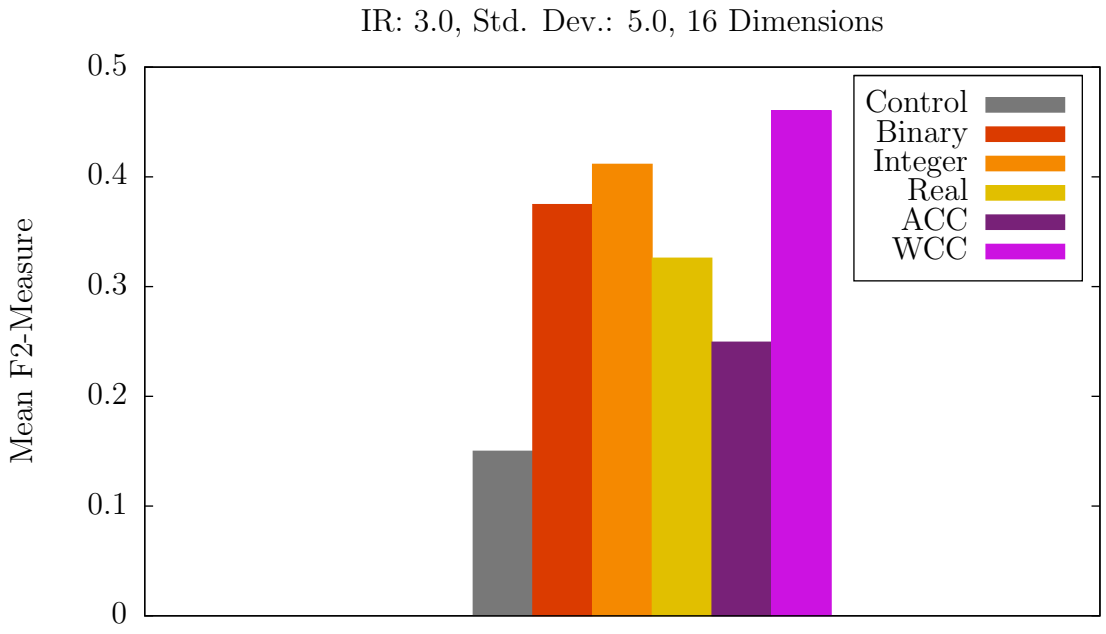


Figure 5.1: 3-NN classifier benefits from evolutionary pre-processing on a high-dimensionality data set

dimensionality. The higher dimensionality creates greater sparsity for a given number of data points. This difficulty, however, presents an opportunity for the evolutionary pre-processing methods, and resampling provides substantial gains for the 3-NN classifiers on the high dimensionality data sets. One example is shown in Figure 5.1. 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 compared to the other classifiers.

The mean F2-measures in Table 5.2 show the benefits of evolutionary resampling for the 3-NN classifier on 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

Table 5.1: 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	+	+	=	+	+	

lower levels of imbalance, the integer representation is better when the imbalance ratio is very high (15.0). The Wilcoxon signed-rank tests shown in Table 5.1 confirm that the superiority of these methods is statistically significant, though neither has an advantage over the other. Offering a lesser degree of improvement, 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 the other evolutionary methods, it is still significantly better than the unassisted 3-NN classifier. Thus evolutionary pre-processing is very beneficial for the 3-NN classifier as all of the resampling techniques promote the construction of a better classifier.

Table 5.2: 3-NN classifier performance on the artificial data sets

	Control	Binary	Integer	Real	ACC	WCC
IR: 3, σ:2.0, 2 Dim.	0.99372	0.97422	0.99404	**0.99686**	0.99468	0.98996
IR: 3, σ:2.0, 8 Dim.	0.53966	0.64166	0.72846	0.67882	0.62010	**0.79300**
IR: 3, σ:2.0, 16 Dim.	0.40598	0.54348	0.61642	0.56350	0.44732	**0.66330**
IR: 3, σ:3.0, 2 Dim.	0.89500	0.85906	0.90756	0.89810	0.90548	**0.91022**
IR: 3, σ:3.0, 8 Dim.	0.44220	0.53450	0.58512	0.56162	0.48726	**0.62094**
IR: 3, σ:3.0, 16 Dim.	0.31320	0.49042	0.50848	0.46776	0.37774	**0.60452**
IR: 3, σ:4.0, 2 Dim.	0.66896	0.69638	0.73510	0.73618	0.67052	**0.75598**
IR: 3, σ:4.0, 8 Dim.	0.37074	0.50728	0.57824	0.52120	0.40474	**0.62044**
IR: 3, σ:4.0, 16 Dim.	0.25522	0.45754	0.49014	0.40306	0.34730	**0.56382**
IR: 3, σ:5.0, 2 Dim.	0.50926	0.59252	0.60100	0.59602	0.49210	**0.65886**
IR: 3, σ:5.0, 8 Dim.	0.20998	0.41544	0.43622	0.37292	0.25204	**0.47446**
IR: 3, σ:5.0, 16 Dim.	0.15010	0.37488	0.41158	0.32596	0.21542	**0.46170**
IR: 7, σ:2.0, 2 Dim.	0.98626	0.94100	**0.98944**	0.98814	0.98500	0.98876
IR: 7, σ:2.0, 8 Dim.	0.30674	0.43200	0.56774	0.47900	0.33804	**0.59046**
IR: 7, σ:2.0, 16 Dim.	0.26380	0.36364	0.41962	0.39374	0.27668	**0.45494**
IR: 7, σ:3.0, 2 Dim.	0.78252	0.75386	0.83036	0.76662	0.74620	**0.83134**
IR: 7, σ:3.0, 8 Dim.	0.15068	0.34644	0.39536	0.31136	0.20482	**0.43684**
IR: 7, σ:3.0, 16 Dim.	0.08994	0.22844	0.23826	0.18654	0.13920	**0.24998**
IR: 7, σ:4.0, 2 Dim.	0.65010	0.68522	0.74794	0.70882	0.67630	**0.75700**
IR: 7, σ:4.0, 8 Dim.	0.09888	0.23922	0.22092	0.23024	0.11752	**0.28208**
IR: 7, σ:4.0, 16 Dim.	0.05014	0.24648	0.24142	0.16080	0.10390	**0.30536**
IR: 7, σ:5.0, 2 Dim.	0.24778	0.38684	0.42558	0.34818	0.26254	**0.44730**
IR: 7, σ:5.0, 8 Dim.	0.05884	0.23808	0.27546	0.17218	0.07432	**0.32210**
IR: 7, σ:5.0, 16 Dim.	0.04378	0.20596	0.22578	0.12592	0.07638	**0.24570**
IR:15, σ:2.0, 2 Dim.	**0.98382**	0.85866	0.98290	0.98026	0.97514	0.97246
IR:15, σ:2.0, 8 Dim.	0.09898	0.25420	**0.37616**	0.28416	0.19980	0.27074
IR:15, σ:2.0, 16 Dim.	0.00000	0.17318	**0.17698**	0.10186	0.09430	0.12846
IR:15, σ:3.0, 2 Dim.	0.76450	0.63216	0.75310	0.74076	**0.76716**	0.73246
IR:15, σ:3.0, 8 Dim.	0.19250	0.23178	**0.25960**	0.21660	0.19504	0.21276
IR:15, σ:3.0, 16 Dim.	0.00000	0.08710	**0.09910**	0.05538	0.00000	0.07518
IR:15, σ:4.0, 2 Dim.	0.37616	0.45086	**0.51608**	0.41814	0.32644	0.49726
IR:15, σ:4.0, 8 Dim.	0.03782	**0.09742**	0.09350	0.05524	0.04398	0.01974
IR:15, σ:4.0, 16 Dim.	0.02252	0.14376	**0.17516**	0.06250	0.00740	0.08812
IR:15, σ:5.0, 2 Dim.	0.35840	0.38856	0.41036	0.39916	0.37824	**0.43188**
IR:15, σ:5.0, 8 Dim.	0.01504	**0.15064**	0.13790	0.04022	0.01438	0.10580
IR:15, σ:5.0, 16 Dim.	0.05084	0.14002	**0.14506**	0.07674	0.06364	0.10318

5.2 J48 Classifier Performance

The unaided J48 classifier is very competitive with the other unassisted learners. One of its strengths is its resistance to degradation as the dimensionality increases. As noted above, the dimensionality of the artificial data sets is increased by adding attributes that 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 that contain the most information, it is able to ignore these extraneous features without any difficulty. In contrast, the 3-NN classifier 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.

Examining the performance increases offered by evolutionary pre-processing, it is clear in Table 5.4 that the inductive tree learner benefits less than the nearest-neighbor approach. One reason for this may be that the GA is guided by the performance of a nearest-neighbor algorithm. As the algorithm evolves a resampling to maximize the 3-NN classifier's performance, it may disregard a resampling that would produce a better F-score from J48. This is also true of the multilayer perceptron and support vector machines; it is a consequence of using a different classifier paradigm as the genetic algorithm's heuristic. Comparing the evolutionary resampling methods, the integer-based resampling approach and WCC emerge as the best methods again. The Wilcoxon signed-rank tests in Table 5.3 show that WCC is better than the control and all of the other methods except integer-based resampling by a statistically significant margin. For J48, however, integer-based resampling is not as good as WCC. While binary undersampling and evolutionary real-valued weighting are significantly worse than WCC, they are not significantly worse than integer-based resampling. Thus, among these four pre-processing methods, WCC stands apart, but all of these encodings are

Table 5.3: 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	+	+	=	+	+	

significantly better than ACC. The sets modified by ACC offer comparable performance to the unmodified data sets at considerable computational expense.

Table 5.4: J48 classifier performance on the artificial data sets

	Control	Binary	Integer	Real	ACC	WCC
IR: 3, σ:2.0, 2 Dim.	0.94062	0.94594	0.96328	**0.96528**	0.96450	0.95514
IR: 3, σ:2.0, 8 Dim.	0.95664	0.94790	0.96748	0.95036	0.95666	**0.98284**
IR: 3, σ:2.0, 16 Dim.	0.93364	0.93342	0.91972	0.93568	0.92628	**0.94738**
IR: 3, σ:3.0, 2 Dim.	0.86078	0.85564	0.86164	**0.88150**	0.85920	0.86002
IR: 3, σ:3.0, 8 Dim.	0.84836	0.84394	0.84160	**0.84856**	0.82836	0.81258
IR: 3, σ:3.0, 16 Dim.	0.87508	0.86614	0.86468	0.84832	0.87104	**0.89542**
IR: 3, σ:4.0, 2 Dim.	0.63936	0.70524	0.69568	**0.72844**	0.65088	0.71136
IR: 3, σ:4.0, 8 Dim.	0.73318	**0.76118**	0.73600	0.73618	0.72896	0.73332
IR: 3, σ:4.0, 16 Dim.	0.71176	0.74742	0.67158	0.71994	0.63632	**0.75890**
IR: 3, σ:5.0, 2 Dim.	0.54424	0.61574	0.54742	0.59946	0.53464	**0.67670**
IR: 3, σ:5.0, 8 Dim.	0.60662	0.63490	0.55228	0.60518	0.55504	**0.66356**
IR: 3, σ:5.0, 16 Dim.	0.51524	0.58144	0.54762	0.57338	0.55238	**0.62268**
IR: 7, σ:2.0, 2 Dim.	0.93748	0.94516	**0.95514**	0.94888	0.94954	0.94836
IR: 7, σ:2.0, 8 Dim.	0.91708	0.92476	**0.93752**	0.92062	0.92636	0.93514
IR: 7, σ:2.0, 16 Dim.	0.95948	0.96212	**0.97012**	0.95692	0.94552	0.94286
IR: 7, σ:3.0, 2 Dim.	0.73034	0.75348	0.78182	0.74014	0.73192	**0.80192**
IR: 7, σ:3.0, 8 Dim.	0.76434	0.72190	0.71920	0.73936	0.73602	**0.79102**
IR: 7, σ:3.0, 16 Dim.	**0.69664**	0.68270	0.67978	0.65574	0.61252	0.67460
IR: 7, σ:4.0, 2 Dim.	0.57724	**0.66422**	0.62474	0.62572	0.55914	0.62892
IR: 7, σ:4.0, 8 Dim.	0.56740	0.58428	**0.62654**	0.55956	0.55322	0.61140
IR: 7, σ:4.0, 16 Dim.	0.53842	**0.58884**	0.48810	0.53618	0.47250	0.53622
IR: 7, σ:5.0, 2 Dim.	0.28186	0.39710	0.35952	0.39936	0.34768	**0.44426**
IR: 7, σ:5.0, 8 Dim.	0.48506	0.47828	0.49014	0.44542	0.44808	**0.51938**
IR: 7, σ:5.0, 16 Dim.	0.31504	**0.38500**	0.35492	0.35894	0.30948	0.36346
IR:15, σ:2.0, 2 Dim.	0.92726	0.90190	0.92822	0.92892	0.85520	**0.92988**
IR:15, σ:2.0, 8 Dim.	0.84882	0.87258	0.87352	**0.87368**	0.86808	0.78314
IR:15, σ:2.0, 16 Dim.	**0.94022**	0.90374	0.91920	0.90208	0.91984	0.92660
IR:15, σ:3.0, 2 Dim.	0.68062	**0.73920**	0.71780	0.70188	0.71134	0.72046
IR:15, σ:3.0, 8 Dim.	**0.82920**	0.73704	0.80458	0.76694	0.74784	0.74216
IR:15, σ:3.0, 16 Dim.	0.54916	0.56772	**0.59472**	0.58082	0.57630	0.58394
IR:15, σ:4.0, 2 Dim.	0.29768	0.45646	**0.47460**	0.43826	0.42188	0.44750
IR:15, σ:4.0, 8 Dim.	0.15408	0.30052	**0.35150**	0.26704	0.24850	0.25716
IR:15, σ:4.0, 16 Dim.	0.47464	0.42614	0.41336	**0.48688**	0.40128	0.45882
IR:15, σ:5.0, 2 Dim.	0.17500	0.36850	0.40888	**0.44318**	0.30898	0.41516
IR:15, σ:5.0, 8 Dim.	0.18774	0.19374	**0.30734**	0.21866	0.21794	0.28684
IR:15, σ:5.0, 16 Dim.	0.18180	0.18264	**0.25572**	0.12680	0.13404	0.19786

5.3 Multilayer Perceptron Classifier Performance

The unassisted multilayer 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 more imbalanced data sets (i.e. two thirds of the artificial data) means that it is not very strong as an unaided method in this experiment.

Although the MLP classifier's performance on the artificial data sets without resampling is lackluster compared to that of support vector machines and J48, evolutionary preprocessing extends 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 eight dimensions (Figure 5.2 and Figure 5.3). These data sets are relatively challenging: they present the extremes of both overlap and imbalance. Among the unaided classifiers, MLP achieves the either the lowest or second-lowest mean F2-measure. Pre-processing the data using integer-encoded resampling, however, yields the best results out of all classifiers and all preprocessing methods. While these examples are exceptional, Table 5.6 shows that the multilayer perceptron does generally benefit from the application of evolutionary pre-processing. Among 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 3-NN and J48, WCC provides relatively large gains as well. For the MLP classifier, Table 5.5 shows that the integer-encoded method is significantly better than WCC and that WCC is significantly better than the other methods. Real-valued evolutionary weighting and binary undersampling offer commensurate, modest benefits for this classifier,

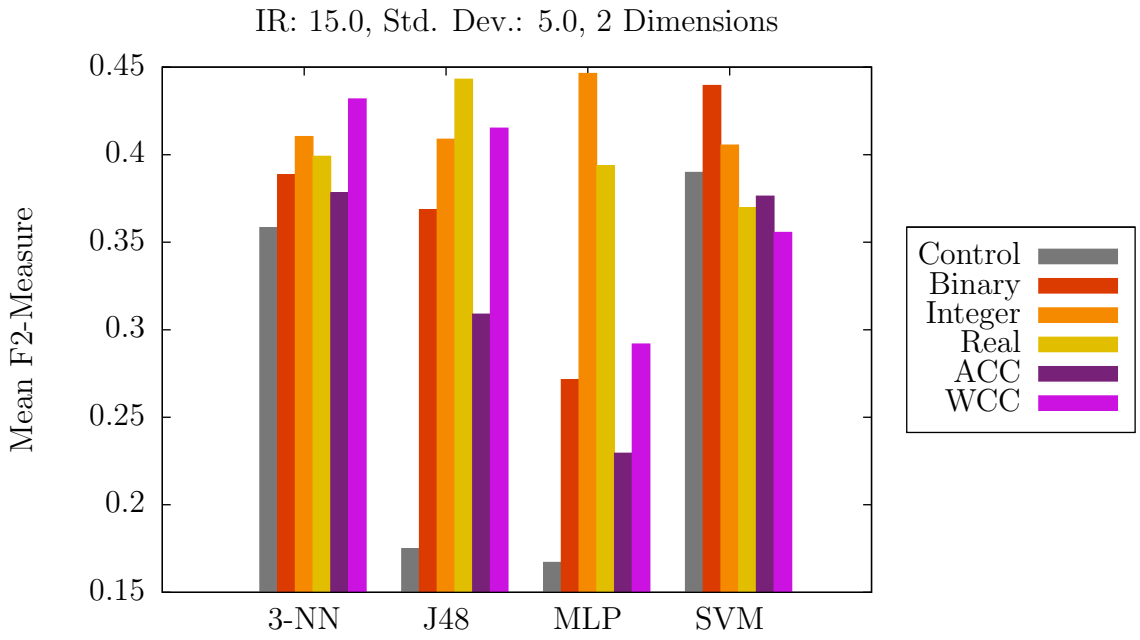


Figure 5.2: MLP classifier benefits from evolutionary pre-processing (two dimensions)

and as in the case of J48, ACC does not produce a significant benefit for the classifier over the artificial sets as a whole.

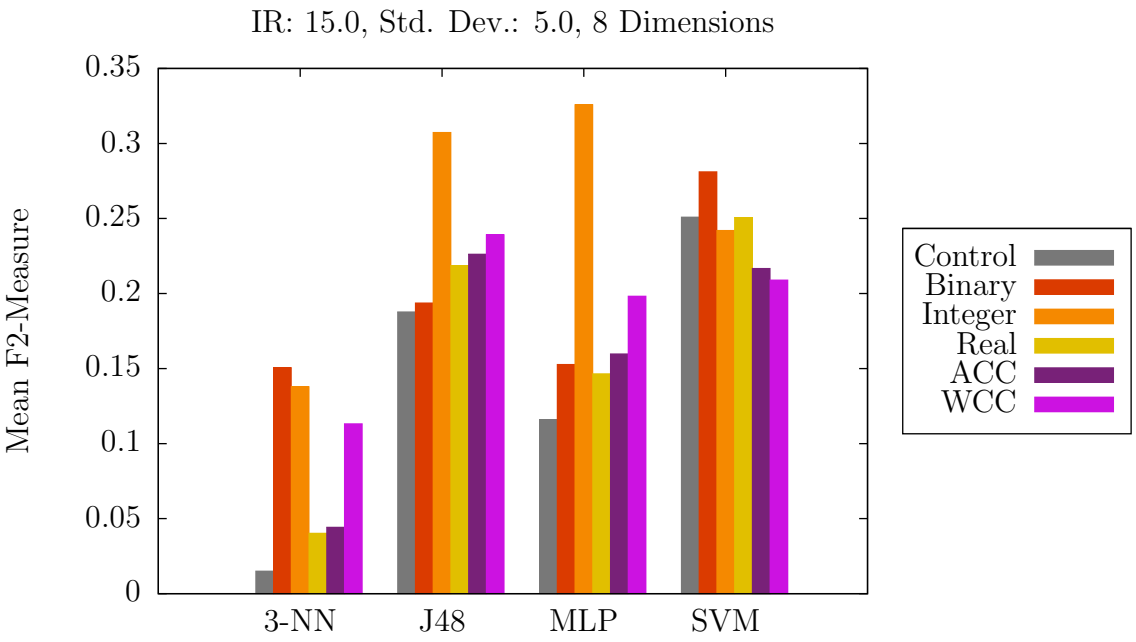


Figure 5.3: MLP classifier benefits from evolutionary pre-processing (eight dimensions)

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	+	+	-	+	+	

Table 5.6: MLP classifier performance on the artificial data sets

	Control	Binary	Integer	Real	ACC	WCC
IR: 3, σ:2.0, 2 Dim.	**0.98092**	0.96118	0.96622	0.98074	0.84624	0.88158
IR: 3, σ:2.0, 8 Dim.	**0.97808**	0.97512	0.97276	0.96810	0.75264	0.84294
IR: 3, σ:2.0, 16 Dim.	0.90508	0.89040	**0.92762**	0.88450	0.71944	0.86180
IR: 3, σ:3.0, 2 Dim.	0.86202	0.87874	**0.88222**	0.88192	0.82390	0.85376
IR: 3, σ:3.0, 8 Dim.	0.76786	0.74980	**0.78074**	0.76212	0.63414	0.77948
IR: 3, σ:3.0, 16 Dim.	0.78152	0.76562	**0.81322**	0.73944	0.64084	0.80558
IR: 3, σ:4.0, 2 Dim.	0.70510	0.72140	**0.75826**	0.73108	0.61854	0.72914
IR: 3, σ:4.0, 8 Dim.	0.58270	0.62528	0.72210	0.64070	0.59080	**0.72798**
IR: 3, σ:4.0, 16 Dim.	0.53912	0.60138	0.62032	0.58452	0.48988	**0.69494**
IR: 3, σ:5.0, 2 Dim.	0.53318	0.62538	0.67102	0.62178	0.53000	**0.68348**
IR: 3, σ:5.0, 8 Dim.	0.49100	0.50602	0.60238	0.52116	0.43770	**0.62762**
IR: 3, σ:5.0, 16 Dim.	0.42974	0.52450	0.56520	0.49302	0.40968	**0.56992**
IR: 7, σ:2.0, 2 Dim.	0.51372	0.69076	**0.93334**	0.89290	0.65710	0.64764
IR: 7, σ:2.0, 8 Dim.	0.41950	0.84834	**0.85808**	0.83278	0.57970	0.77086
IR: 7, σ:2.0, 16 Dim.	0.39252	0.68864	**0.85846**	0.75900	0.57480	0.84194
IR: 7, σ:3.0, 2 Dim.	0.56680	0.67904	**0.78454**	0.69158	0.45570	0.63298
IR: 7, σ:3.0, 8 Dim.	0.48884	0.59474	0.64374	0.62560	0.45540	**0.66912**
IR: 7, σ:3.0, 16 Dim.	0.33662	0.39626	**0.51030**	0.48062	0.36842	0.50496
IR: 7, σ:4.0, 2 Dim.	0.61032	0.62658	**0.71234**	0.61648	0.47890	0.62360
IR: 7, σ:4.0, 8 Dim.	0.35468	0.38590	0.35748	0.30406	0.35324	**0.42788**
IR: 7, σ:4.0, 16 Dim.	0.35276	0.34778	**0.39536**	0.33338	0.29186	0.38754
IR: 7, σ:5.0, 2 Dim.	0.25026	0.34612	**0.44316**	0.35466	0.24568	0.42472
IR: 7, σ:5.0, 8 Dim.	0.33720	0.39572	0.40922	0.34610	0.28584	**0.42692**
IR: 7, σ:5.0, 16 Dim.	0.25368	0.26326	0.27980	0.19378	0.20572	**0.29956**
IR:15, σ:2.0, 2 Dim.	0.42588	0.52126	**0.70452**	0.60404	0.52748	0.55214
IR:15, σ:2.0, 8 Dim.	0.26850	0.32606	**0.40480**	0.37854	0.34690	0.39270
IR:15, σ:2.0, 16 Dim.	0.23942	0.36802	**0.55680**	0.36758	0.49082	0.47208
IR:15, σ:3.0, 2 Dim.	0.25558	0.36174	**0.66926**	0.41524	0.33950	0.49084
IR:15, σ:3.0, 8 Dim.	0.20402	0.29774	**0.34872**	0.32096	0.26548	0.32234
IR:15, σ:3.0, 16 Dim.	0.28504	0.25510	0.31408	0.30136	0.26532	**0.35424**
IR:15, σ:4.0, 2 Dim.	0.13974	0.35638	**0.47186**	0.30164	0.20006	0.38650
IR:15, σ:4.0, 8 Dim.	0.20874	0.18000	**0.37060**	0.18446	0.13812	0.18182
IR:15, σ:4.0, 16 Dim.	0.04986	0.11190	**0.25108**	0.14638	0.12666	0.17202
IR:15, σ:5.0, 2 Dim.	0.16716	0.27156	**0.44644**	0.39374	0.22950	0.29186
IR:15, σ:5.0, 8 Dim.	0.11598	0.15278	**0.32580**	0.14650	0.21064	0.19422
IR:15, σ:5.0, 16 Dim.	0.11626	0.13076	**0.18266**	0.10728	0.13544	0.09494

5.4 Support Vector Machines Classifier Performance

The unaided support vector machines classifier performs relatively well on all of the artificial data sets. The SVM classifier is subject to more parameter tuning than the other methods because it performs exceptionally poorly using the libSVM default parameters. The benefit of this tuning deserves consideration when comparing it against the other classifiers, which underwent minimal configuration. As predicted by previous research, this learner is relatively unaffected by imbalance, though overlap degrades its performance substantially [Japkowicz and Stephen, 2002, Denil and Trappenberg, 2010]. The dimensionality of the artificial data sets makes a substantial difference with respect to which kernel functions and parameter values are effective, but with an appropriate configuration selected, support vector machines are relatively unaffected by dimensionality.

The SVMs do not benefit very much from evolutionary pre-processing compared to the multilayer perceptron or the nearest-neighbor classifier. As shown in Table 5.8, small gains in performance are the norm, but substantial gains are visible in some data sets (e.g. the data set shown in Figure 5.4). Nonetheless, comparing the resampling methods' relative performance is still valuable. Here, as for the other classifiers, integer-based resampling is a good pre-processing method. Although within-class cluster weighting is able to produce significant improvements for the other machine learning paradigms, the SVM built from data altered by WCC is not significantly better than unassisted SVMs when examined over the entire 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, but neither is significantly better than the other. On the other hand, none of the resampling methods significantly degraded the performance of the SVM classifier.

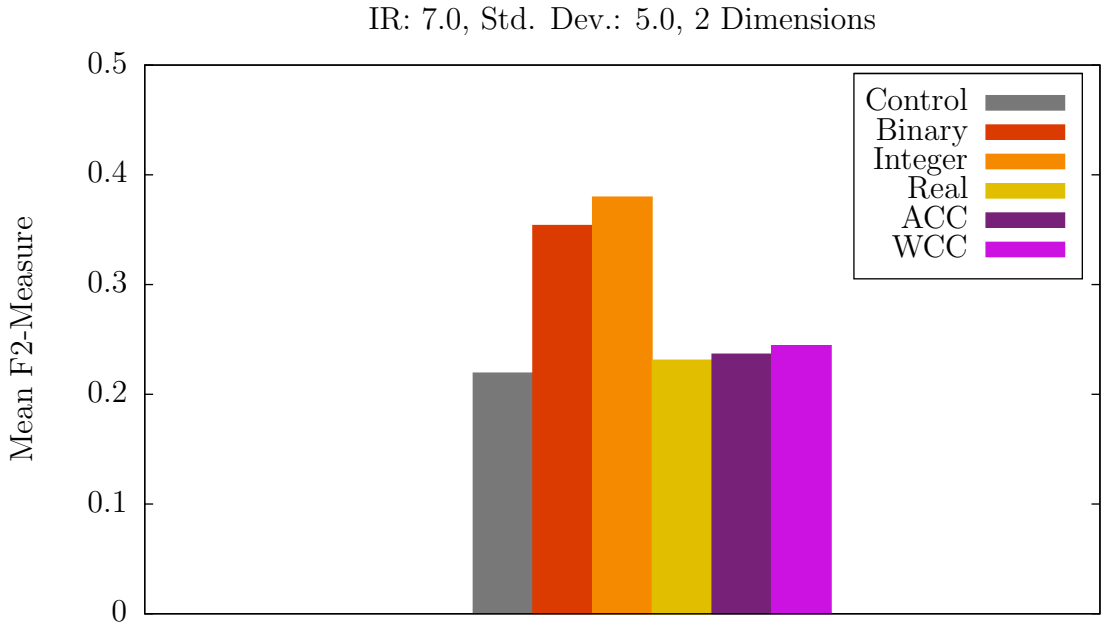


Figure 5.4: SVM classifier benefits from evolutionary pre-processing

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	=	-	-	=	=	

Table 5.8: SVM classifier performance on the artificial data sets

	Control	Binary	Integer	Real	ACC	WCC
IR: 3, σ:2.0, 2 Dim.	**1.00000**	0.99874	**1.00000**	**1.00000**	**1.00000**	**1.00000**
IR: 3, σ:2.0, 8 Dim.	0.98684	0.98258	0.98436	0.98464	0.98278	**0.98716**
IR: 3, σ:2.0, 16 Dim.	0.94616	0.92564	0.94926	0.94902	**0.95036**	0.94714
IR: 3, σ:3.0, 2 Dim.	0.90580	**0.91342**	0.89950	0.89950	0.89554	0.89624
IR: 3, σ:3.0, 8 Dim.	0.70956	**0.76896**	0.72724	0.70386	0.71452	0.71512
IR: 3, σ:3.0, 16 Dim.	**0.86730**	0.84778	0.86508	0.86116	0.86228	0.86220
IR: 3, σ:4.0, 2 Dim.	0.69910	0.76088	**0.78024**	0.70708	0.70804	0.69632
IR: 3, σ:4.0, 8 Dim.	0.62378	**0.68744**	0.62916	0.59144	0.63618	0.62312
IR: 3, σ:4.0, 16 Dim.	0.64280	**0.70082**	0.65870	0.63744	0.66828	0.66128
IR: 3, σ:5.0, 2 Dim.	0.53074	0.63532	**0.64014**	0.53056	0.52296	0.52872
IR: 3, σ:5.0, 8 Dim.	0.46930	**0.53768**	0.47134	0.46392	0.48032	0.47176
IR: 3, σ:5.0, 16 Dim.	0.47834	**0.57650**	0.48302	0.47650	0.46520	0.49204
IR: 7, σ:2.0, 2 Dim.	0.96986	0.96224	0.96926	**0.97300**	0.96926	0.97180
IR: 7, σ:2.0, 8 Dim.	0.94234	0.93824	0.93810	0.94838	0.94174	**0.94858**
IR: 7, σ:2.0, 16 Dim.	0.91440	0.92800	0.92886	0.92144	0.91692	**0.92894**
IR: 7, σ:3.0, 2 Dim.	0.71144	**0.73682**	0.69652	0.70492	0.70876	0.70926
IR: 7, σ:3.0, 8 Dim.	0.60978	**0.67810**	0.62958	0.60402	0.61450	0.59592
IR: 7, σ:3.0, 16 Dim.	**0.67866**	0.62764	0.65478	0.65630	0.66206	0.65262
IR: 7, σ:4.0, 2 Dim.	0.58072	0.57960	**0.59624**	0.56956	0.57506	0.56894
IR: 7, σ:4.0, 8 Dim.	0.37526	**0.49266**	0.42028	0.38152	0.38294	0.37378
IR: 7, σ:4.0, 16 Dim.	0.47592	**0.53176**	0.50858	0.48230	0.46100	0.48756
IR: 7, σ:5.0, 2 Dim.	0.21948	0.35394	**0.37978**	0.23120	0.23674	0.24452
IR: 7, σ:5.0, 8 Dim.	0.32720	**0.39806**	0.36266	0.37294	0.36716	0.38036
IR: 7, σ:5.0, 16 Dim.	0.25218	0.22548	**0.26842**	0.25124	0.25234	0.23588
IR:15, σ:2.0, 2 Dim.	0.96392	0.93616	0.95648	0.96010	**0.96880**	0.95374
IR:15, σ:2.0, 8 Dim.	0.90736	0.87114	**0.92600**	0.90564	0.90962	0.89530
IR:15, σ:2.0, 16 Dim.	0.83410	0.74266	0.83646	**0.84014**	0.83638	0.80620
IR:15, σ:3.0, 2 Dim.	0.79866	0.76528	0.74758	**0.80074**	0.79520	0.79656
IR:15, σ:3.0, 8 Dim.	0.67918	0.66652	**0.68698**	0.64836	0.67262	0.68504
IR:15, σ:3.0, 16 Dim.	**0.42652**	0.42042	0.41238	0.40710	0.40112	0.40228
IR:15, σ:4.0, 2 Dim.	0.46840	0.53424	0.46790	0.46162	0.45964	**0.53468**
IR:15, σ:4.0, 8 Dim.	0.24688	**0.28746**	0.24194	0.23714	0.21718	0.25640
IR:15, σ:4.0, 16 Dim.	0.22602	0.14830	0.19752	**0.23400**	0.21340	0.19254
IR:15, σ:5.0, 2 Dim.	0.38994	**0.43960**	0.40552	0.36964	0.37634	0.35562
IR:15, σ:5.0, 8 Dim.	0.25094	**0.28110**	0.24208	0.25062	0.24366	0.22410
IR:15, σ:5.0, 16 Dim.	0.14078	**0.15236**	0.15040	0.11670	0.14782	0.13472

5.5 Effects of Imbalance and Overlap

A battery of artificial data sets is employed here to evaluate each evolutionary pre-processing method and each classifier paradigm while controlling the presence of data set features which have been shown to make constructing effective classifiers more difficult. This allows 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. WCC does seem to be more effective at lower levels of imbalance, but the difference is not stark. Predictably, these experiments show that imbalance and overlap hinder classifier performance, but which pre-processing method will offer the greatest gains does not seem to be predictable from the relative quantities of these two features. It may be that trends would emerge by examining more data sets with different distributions. It is also 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 not visible here.

5.6 UCI Data Set Performance

In the tables below and the plots in Appendix B, the UCI results resemble the artificial data results. These similarities are encouraging, as they suggest that the conclusions drawn from the analysis above may be applicable outside the scope of data sets resembling Gaussian clusters in an XOR pattern. For the 3-NN (Table 5.9) and MLP (Table 5.10) classifiers, WCC and genetic algorithm using the integer encoding produced relatively large improvements. J48 (Table 5.11) appears to benefit from WCC the most, and support vector machines (Table 5.12) derive the most benefit from the genetic algorithms using integer and binary

Table 5.9: 3-NN classifier performance on the UCI data sets

	Control	Binary	Integer	Real	ACC	WCC
E. Coli	0.93982	0.93984	**0.95560**	0.94778	0.93982	0.93750
Iris	0.93932	0.91586	0.94444	**0.94630**	0.93676	0.94188
SPECTF	0.34264	0.42626	0.50310	0.43180	0.31666	**0.55432**
WDBC	0.94512	0.92516	**0.95298**	0.94750	0.94972	0.95112
Yeast	0.52054	0.55106	0.59212	0.55746	0.52004	**0.60252**

Table 5.10: MLP classifier performance on the UCI data sets

	Control	Binary	Integer	Real	ACC	WCC
E. Coli	0.94422	0.94218	**0.95206**	0.94672	0.94710	0.93878
Iris	0.93254	0.94700	0.94706	**0.95296**	0.93358	0.94182
SPECTF	0.35682	**0.53638**	0.48008	0.36804	0.43244	0.51274
WDBC	0.94920	0.95788	**0.96500**	0.95804	0.95408	0.95724
Yeast	0.52022	0.58198	**0.61930**	0.59024	0.48316	0.60990

encodings. When the unassisted classifier is able to achieve high performance, there is little, if any, gain from pre-processing. The E. Coli, Iris, and WDBC data sets demonstrate this. On the more difficult SPECTF and Yeast sets, the F2-measures produced by all four classifier methods are improved substantially by pre-processing, especially pre-processing with WCC, binary undersampling, and integer resampling.

Table 5.11: J48 classifier performance on the UCI data sets

	Control	Binary	Integer	Real	ACC	WCC
E. Coli	0.94440	**0.94672**	0.92504	0.93796	0.93936	0.93574
Iris	0.92240	0.91900	0.91850	0.92522	0.90932	**0.93412**
SPECTF	0.37190	0.45304	0.40096	0.34822	0.34414	**0.49750**
WDBC	0.92740	0.92150	**0.94106**	0.91914	0.92366	0.93482
Yeast	0.51186	0.52984	0.50906	0.53306	0.49394	**0.57874**

Table 5.12: SVM classifier performance on the UCI data sets

	Control	Binary	Integer	Real	ACC	WCC
E. Coli	0.92172	0.92726	**0.95486**	0.92384	0.92238	0.92172
Iris	**0.96262**	0.95454	0.94036	0.96260	0.94166	0.95686
SPECTF	0.38450	**0.54196**	0.44112	0.41506	0.42134	0.39616
WDBC	0.93682	0.94002	**0.95024**	0.93116	0.93852	0.93826
Yeast	0.38098	0.56272	**0.61660**	0.37654	0.38228	0.37380

The Wilcoxon signed-rank test reveals relatively few significant performance differences. The 3-NN classifier benefits significantly from two methods, as shown in Table 5.13. As in the artificial data sets, integer resampling provides relatively large improvements: it is significantly better than the unaided classifier, binary undersampling, and ACC. The GA using the real-valued representation is also able to produce statistically significant benefits compared to binary undersampling and the control. The Wilcoxon signed-rank test produces a very similar table for multilayer perceptrons (Table 5.14). Once again, the integer-based GA and the real-valued, one-to-one GA provide the greatest benefit over the UCI group as a whole, and once again, the integer representation performs slightly better than the real-valued encoding. The former outperforms the control and ACC by a significant margin while the latter outperforms only the control.

For both the 3-NN and MLP classifiers, WCC improves learning on the artificial data sets, so it is surprising that neither inductive method benefits significantly from WCC on the UCI group. Despite its large improvements (especially for 3-NN) on the SPECTF and Yeast data, WCC does not perform significantly better than any other processor, much less the control. This is due in large part to its minute negative effects on the mean F-measure in the E. Coli data set. WCC offers a better mean F-measure than the unassisted 3-NN classifier, ACC, and binary undersampling for every other data set. The multilayer perceptron's results are similar, but WCC does not outperform binary undersampling as consistently.

The results of the Wilcoxon signed-rank test for J48 show only that ACC is significantly worse than the control (Table 5.15). This is due in large part to the small number of UCI data sets and J48's performance on the E. Coli data set. The Wilcoxon signed-rank test requires a minimum of five samples. If only five are available, then for one source to be significantly better than another, all of its samples must be larger than the other's. For the E. Coli data set, ACC produces a higher mean F-measure than integer-based resampling and WCC, if only by a small margin. This is enough to preclude the integer-encoded GA and WCC from being significantly better than ACC despite the fact that they produce stronger performance for the other four data sets. The WDBC data set has a similar effect for evolutionary binary undersampling, which produces better performances than ACC for the other four sets. The SVM classifier demonstrates a similar pattern: the only statistically significant difference is that ACC is worse than the binary-encoded GA. Integer-based resampling and binary undersampling would have been significantly better than the control except that their mean F-measures were slightly worse for a single data set: Iris.

A recurring pattern is that small differences in a single, relatively "easy" data sets prevent differences in performance from being statistically significant. The data sets evaluated here were chosen because they are frequently used in the literature and not for their complicating properties. Like Stefanowski's SPIDER or Chawla et al.'s wrapper variant of SMOTE, however, the evolutionary methods proposed in this thesis are most suitable for data sets on which the unassisted classifier's performance is very poor [Stefanowski, 2013, Chawla et al., 2008]. For data sets that do not fit this description, pre-processing may fail to produce any improvement or even produce slightly worse F-measures; in these circumstances, the default behavior of the learner is effective, and changing it is not beneficial. Furthermore, although measures have been taken to reduce variation across trials, it is possible that the small F-score differences are due in part to the stochastic elements of sampling and genetic search. To gain greater insight into how the evolutionary pre-processors affect classifier performance

Table 5.13: 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 5.14: Wilcoxon signed-rank test for the MLP classifier's performance on the UCI data sets

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

on real-world data sets, it would be helpful to use data sets more representative of ideal applications and to use more data sets. This does not undermine the findings above, but it is important to understand why the pre-processors' improvements are not significant.

Table 5.15: 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	=	=	=	=	=	

Table 5.16: Wilcoxon signed-rank test for the SVM 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 study of data pre-processing by proposing four novel evolutionary techniques and evaluating their performance across data sets with varying dimensionality, imbalance, and overlap. These techniques take the form of genetic representations and operators to modify them: integer resampling, real-valued weighting, and cluster-based weighting for clusters generated within and across class boundaries. They are compared against classifiers built without pre-processing and a GA that uses a binary encoding similar to pre-existing methods. For many data sets, especially data sets with a high degree of imbalance and overlap, training the classifier on pre-processed data sets yields dramatic improvements in the F2-score. Evolutionary pre-processing is most applicable when classifiers trained from the raw training set exhibit poor recall. The methods presented here are most relevant in the presence of imbalance because they are designed specifically for sets where one class is underrepresented. The real-valued one-to-one representation and ACC both include operators for correcting between-class imbalance, and for all of the genetic algorithms that use a one-to-one representation, the population seeded to emphasize the minority class. In this experiment, evolutionary pre-processing yields relatively large increases in the F2-measure for 3-NN classifiers and multilayer perceptrons, so applications that employ these

inductive techniques might also be ideal applications for genetic resampling. Although evolutionary pre-processing can improve classification substantially, it is likely not worth its computational cost for problems where an unassisted learner is able to achieve nearly perfect recall. If support vector machines are the classifier of choice, it is also important to be aware that the relative performance gains from pre-processing may be small. Additionally, the one-to-one representations do not scale well with data set size: they require substantially more time as the size of the training set grows. While evolutionary pre-processing is not ideal for all applications, it can be very helpful when the machine learning task is difficult. The experiment presented in this thesis compares the proposed pre-processors, and it suggests some avenues for future research.

The genetic algorithm using the integer representation provides large, consistent improvements for classifiers built from its resamplings. The benefits of this technique are particularly visible for 3-NN classifiers and multilayer perceptrons, and although support vector machines experience little benefit from evolutionary pre-processing in general, they derive the most benefit from integer-based resampling. Although one might anticipate that this method would cause overfitting to the detriment of the F2-score, its improvements generally outweigh its disadvantages. 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 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 mitigates the problem of absolute rarity identified by Japkowicz and Stephen as the most detrimental aspect of between-class imbalance [Japkowicz and Stephen, 2002]. Integer-based resampling's success is very encouraging because it is simple and lacks domain-specific operators; there are many possible extensions to investigate. To avoid overfitting, instances could be oversampled not

by direct duplication but by small, random perturbations. To reduce dimensionality, one could consider only elements in borderline regions for resampling. Addressing only elements at risk for misclassification is a common tactic for pre-processing methods. To counter data set growth, it is possible to incorporate a term in the fitness function which penalizes exceedingly large data sets, introduce a local search component to reduce the sum of allele values without degrading the fitness, or apply genetic operators which cause large allele values to decay if there is not strong selection pressure to maintain their presence.

Weighting the instances using within-class cluster weighting also produces outstanding gains for all of the classifiers except support vector machines. The principal advantage of this method is the size of the genetic representation, which allows for fast convergence and better efficiency. Unfortunately, its effectiveness 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. 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 is beneficial for the SVM classifier where most other methods fail. The success of the binary representation is not entirely surprising since, despite its simplicity, similar techniques have proven successful in other work [Ishibuchi and Nakashima, 2000, 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 un-

dersampling 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, real-valued one-to-one weighting is not particularly effective on the artificial data sets. It does offer consistent performance on the UCI group for the 3-NN classifier and the multilayer perceptron, but little else can be said in its favor. This may be due in part to the finding that in some cases, resampling can be more effective than equivalent instance weighting [Seiffert et al., 2008]. 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). The poor performance of one GA scheme with one set of parameters is not a sufficient basis to conclude that real-valued one-to-one encodings are fruitless in general, but at this point, it does not appear that this method is an especially promising avenue of research. 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 sampling of the data.

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 effectiveness 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 could, however, be extended by applying some form of parameter control. In short, any of the evolutionary techniques and the machine learning methods 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.

[Bache and Lichman, 2013] Bache, K. and Lichman, M. (2013). UCI machine learning repository.

[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.

[Ishibuchi and Nakashima, 2000] Ishibuchi, H. and Nakashima, T. (2000). Multi-objective pattern and feature selection by a genetic algorithm. In *GECCO*, page 1069.

[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 k-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.

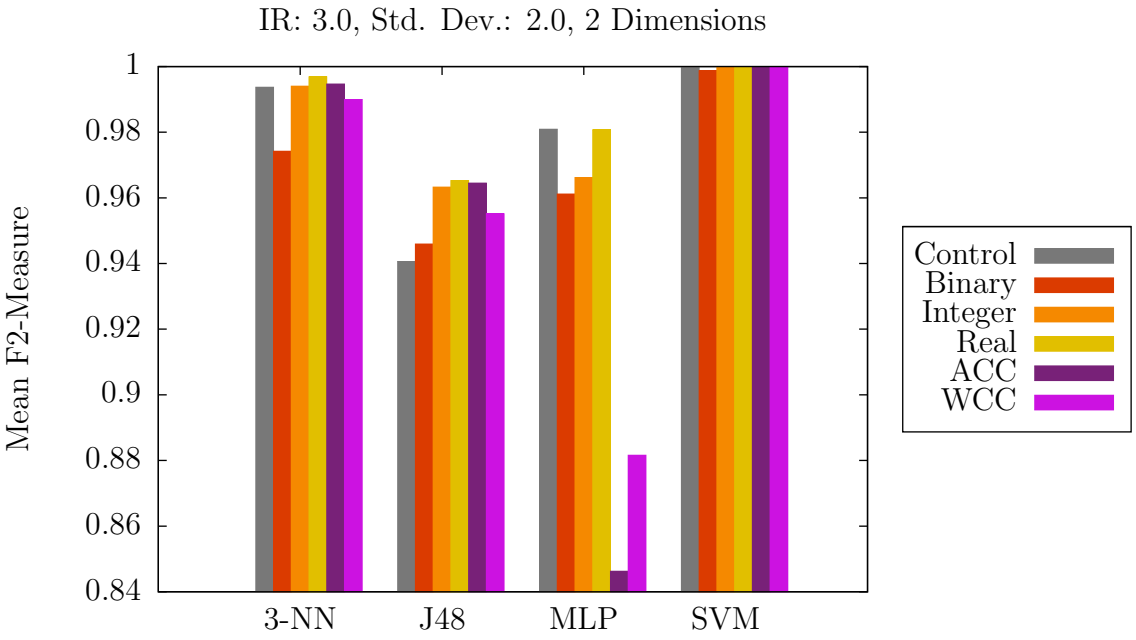
[Weiss, 2004] Weiss, G. M. (2004). Mining with rarity: a unifying framework. *ACM SIGKDD Explorations Newsletter*, 6(1):7–19.

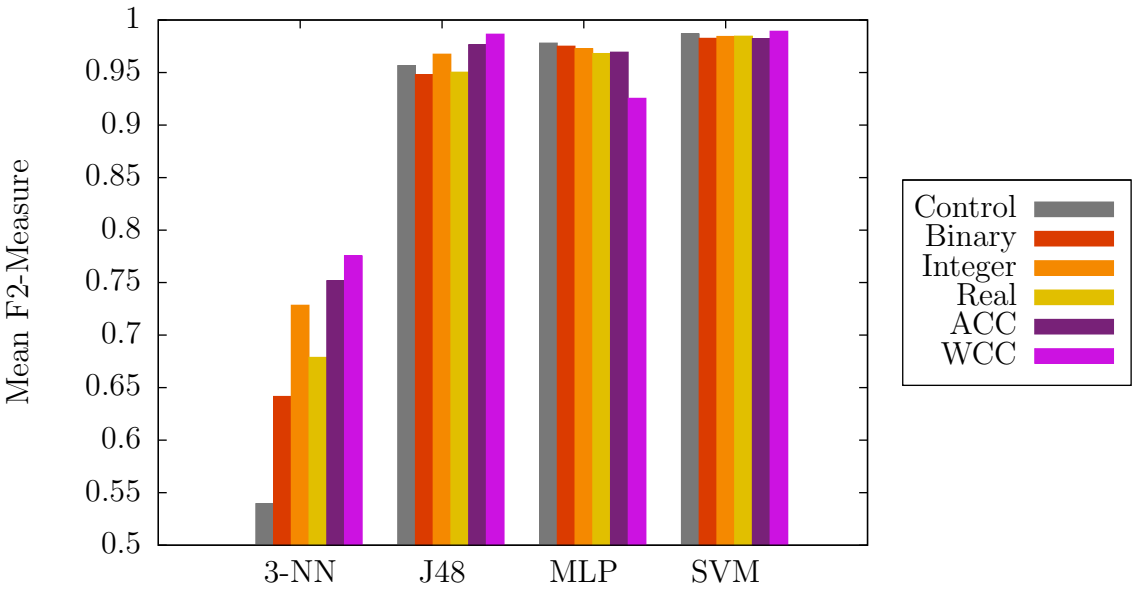
Appendices

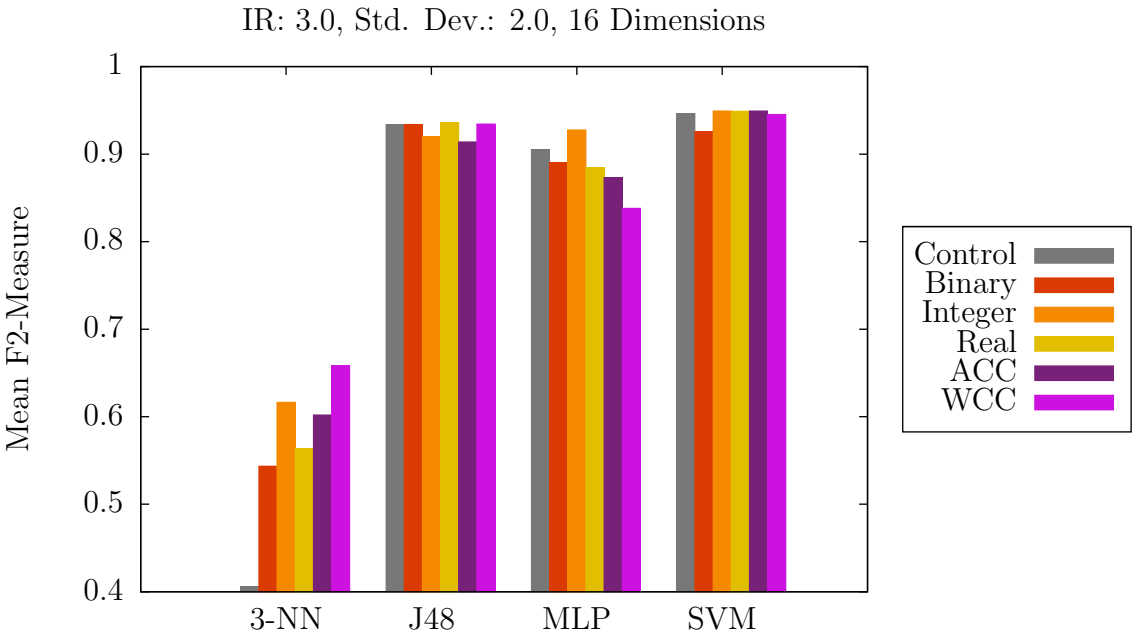
Appendix A

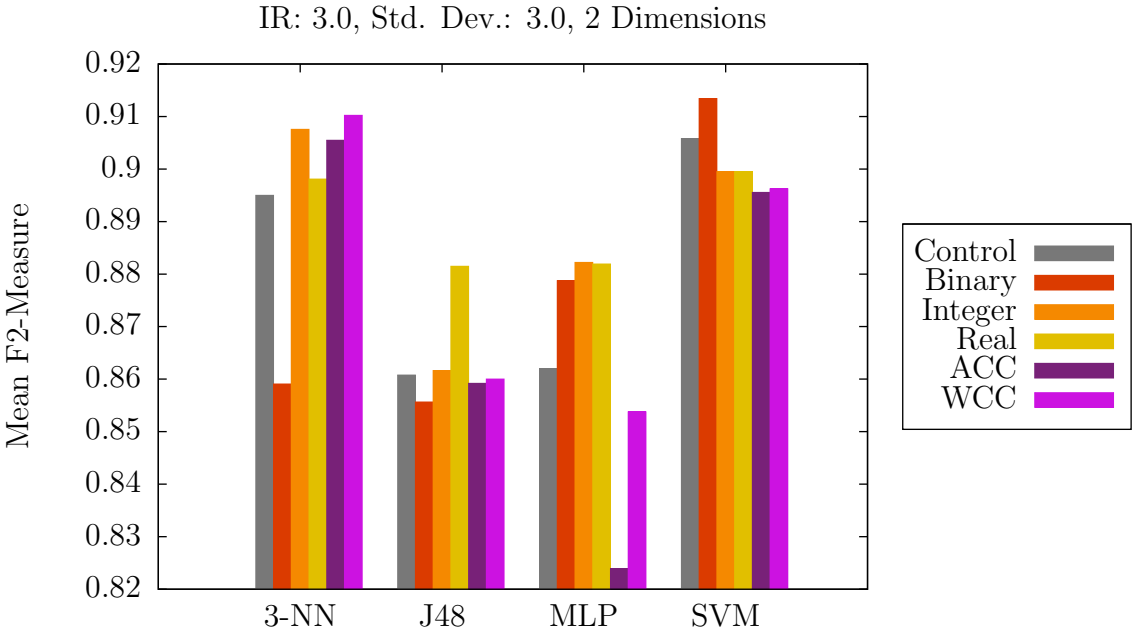
Artificial Data Performance Plots

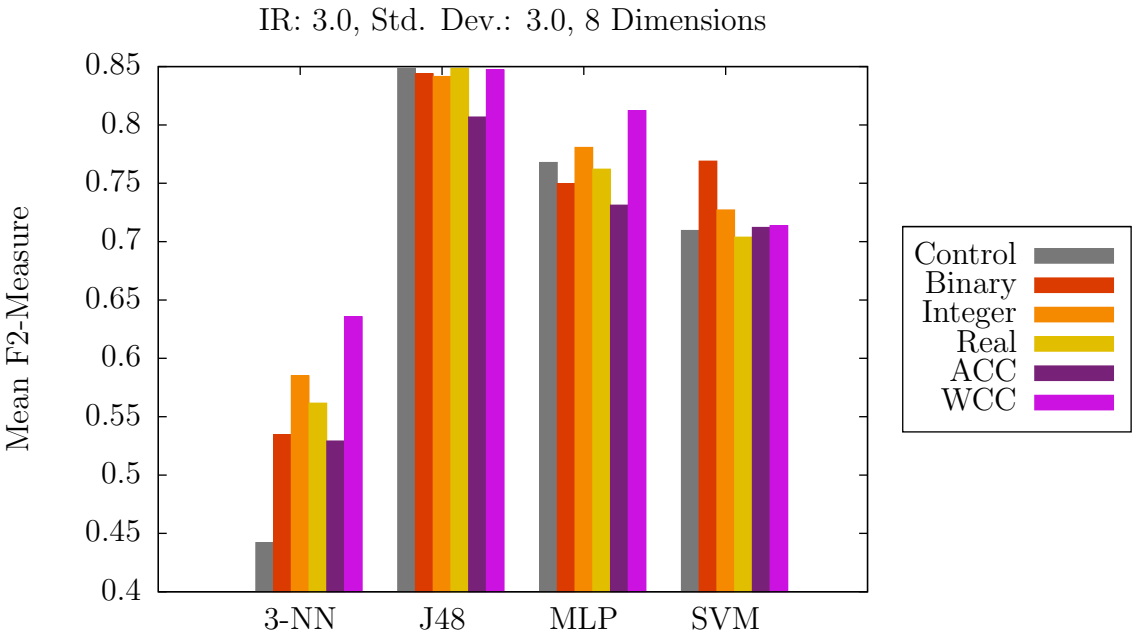
This appendix contains graphical representations of each classifier/pre-processor combination's performance on each artificial data set. The mean F2-measures plotted here are the same as those in Table 5.2, Table 5.4, Table 5.6, and Table 5.8. They are provided in this form for easier comparison. The graphs are arranged in order of ascending imbalance, then ascending overlap, and finally, ascending dimensionality.

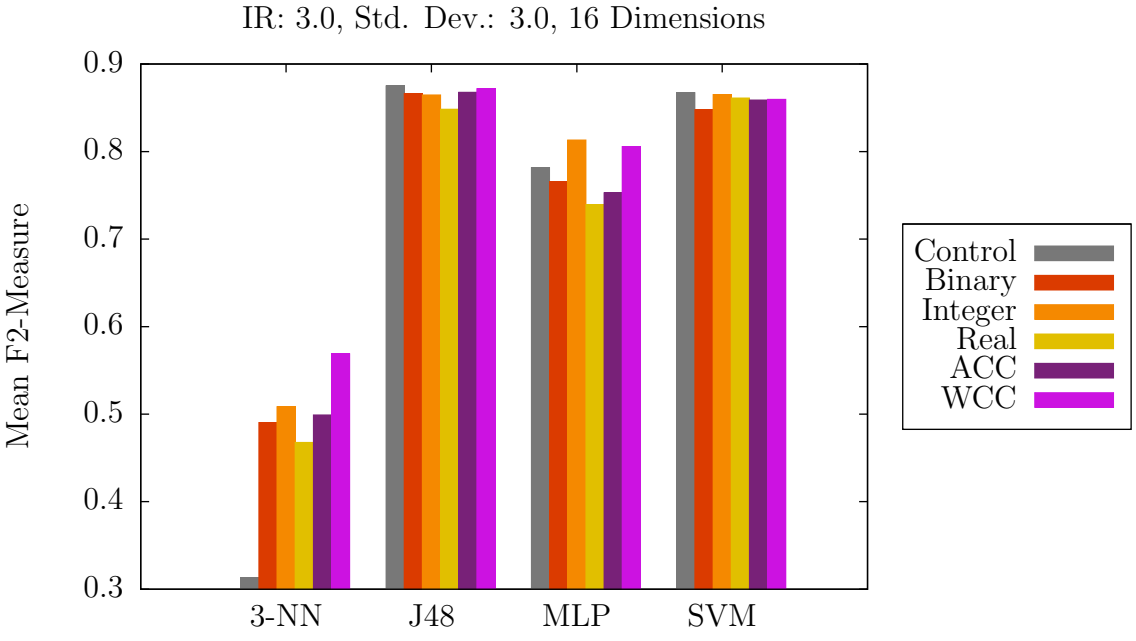

IR: 3.0, Std. Dev.: 2.0, 2 Dimensions
Mean F2-Measure
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

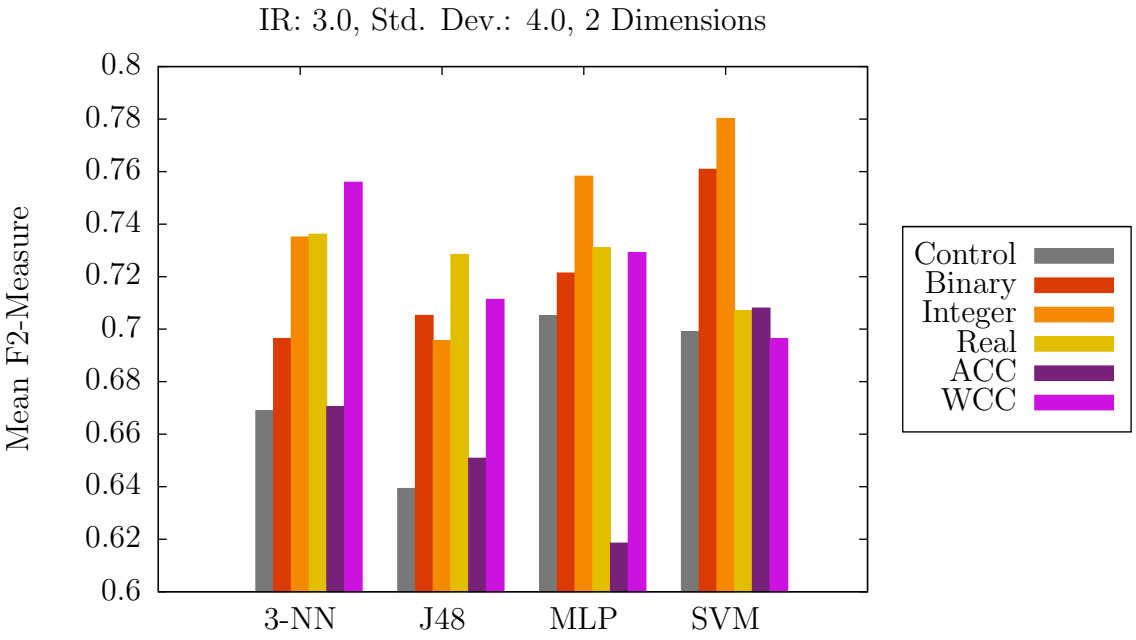

IR: 3.0, Std. Dev.: 2.0, 8 Dimensions
Mean F2-Measure
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

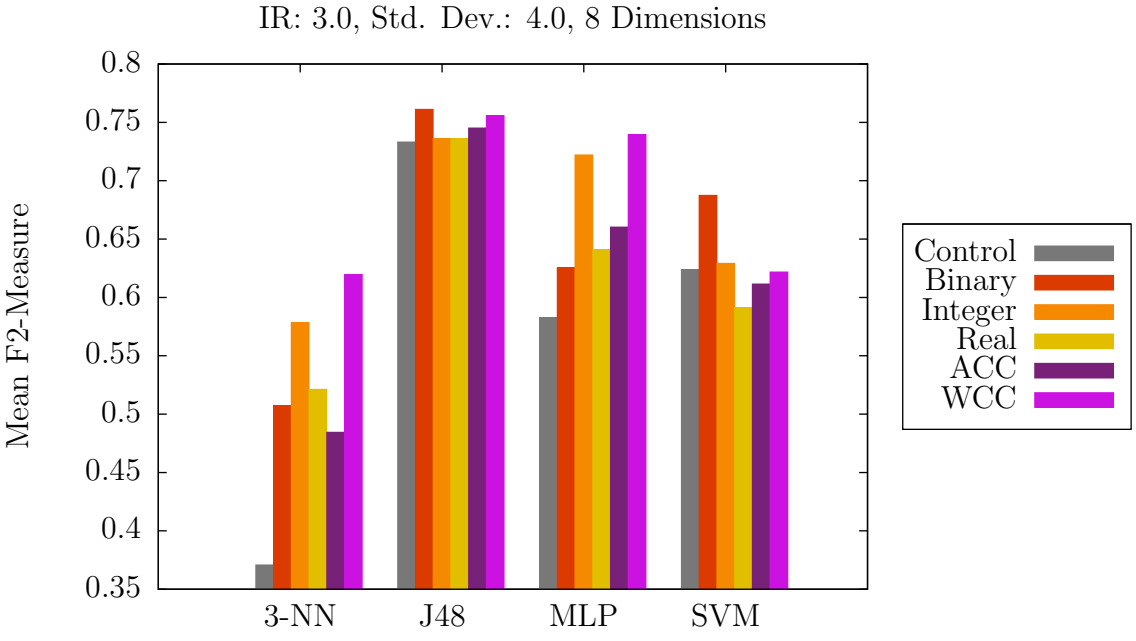

IR: 3.0, Std. Dev.: 2.0, 16 Dimensions
Mean F2-Measure
1
0.9
0.8
0.7
0.6
0.5
0.4
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

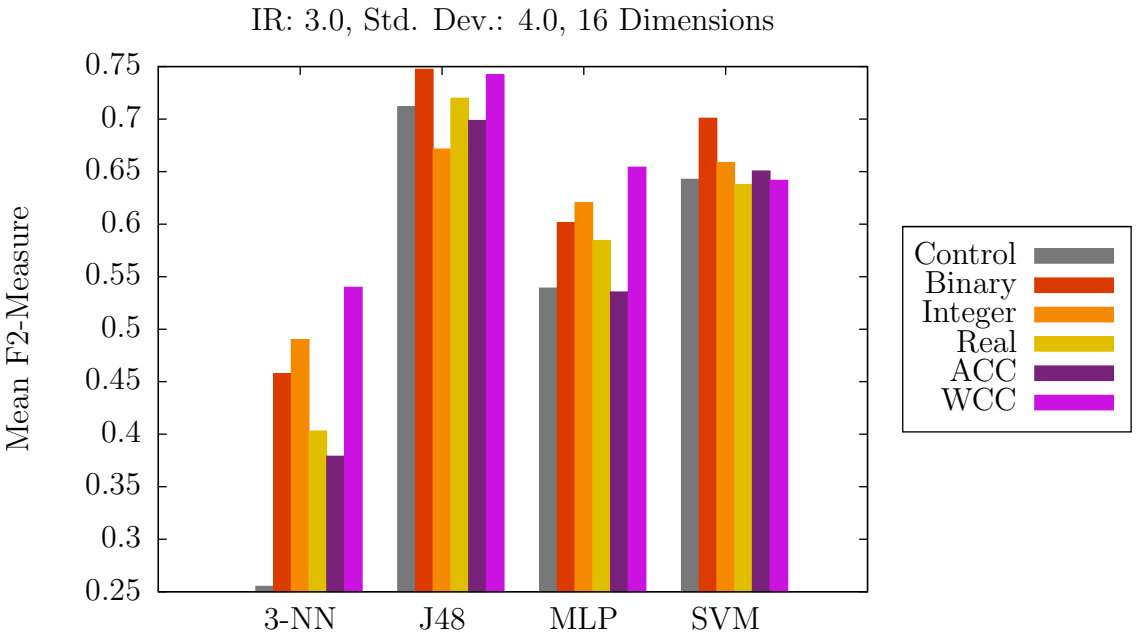

IR: 3.0, Std. Dev.: 3.0, 2 Dimensions
Mean F2-Measure
0.92
0.91
0.9
0.89
0.88
0.87
0.86
0.85
0.84
0.83
0.82
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

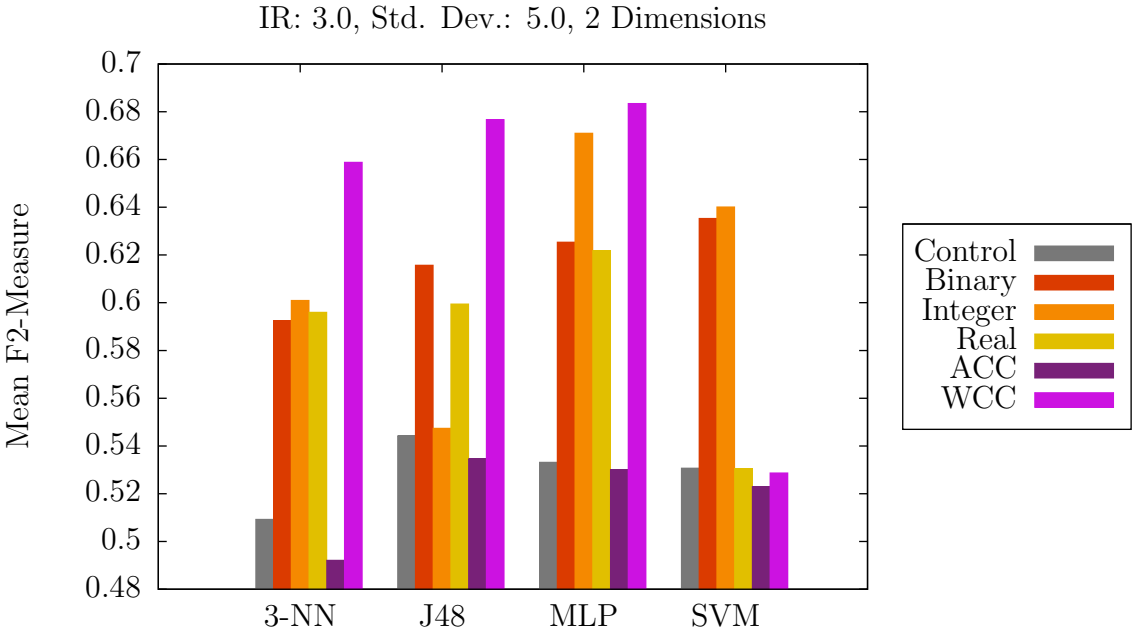

IR: 3.0, Std. Dev.: 3.0, 8 Dimensions
Mean F2-Measure
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

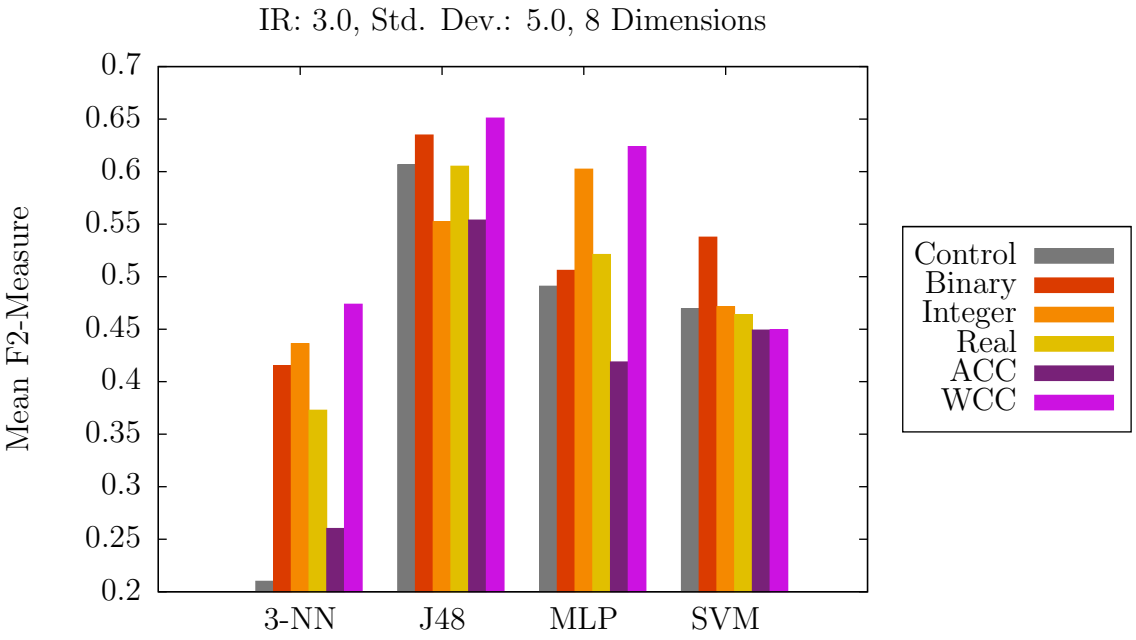

IR: 3.0, Std. Dev.: 3.0, 16 Dimensions
Mean F2-Measure
0.9
0.8
0.7
0.6
0.5
0.4
0.3
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

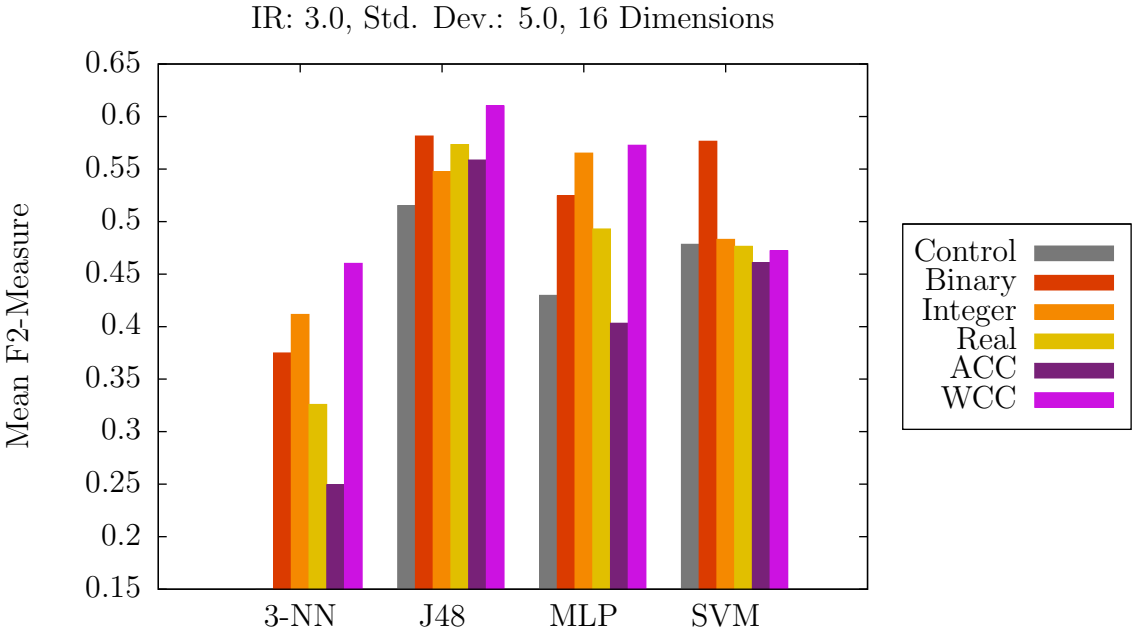

IR: 3.0, Std. Dev.: 4.0, 2 Dimensions
Mean F2-Measure
0.8
0.78
0.76
0.74
0.72
0.7
0.68
0.66
0.64
0.62
0.6
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

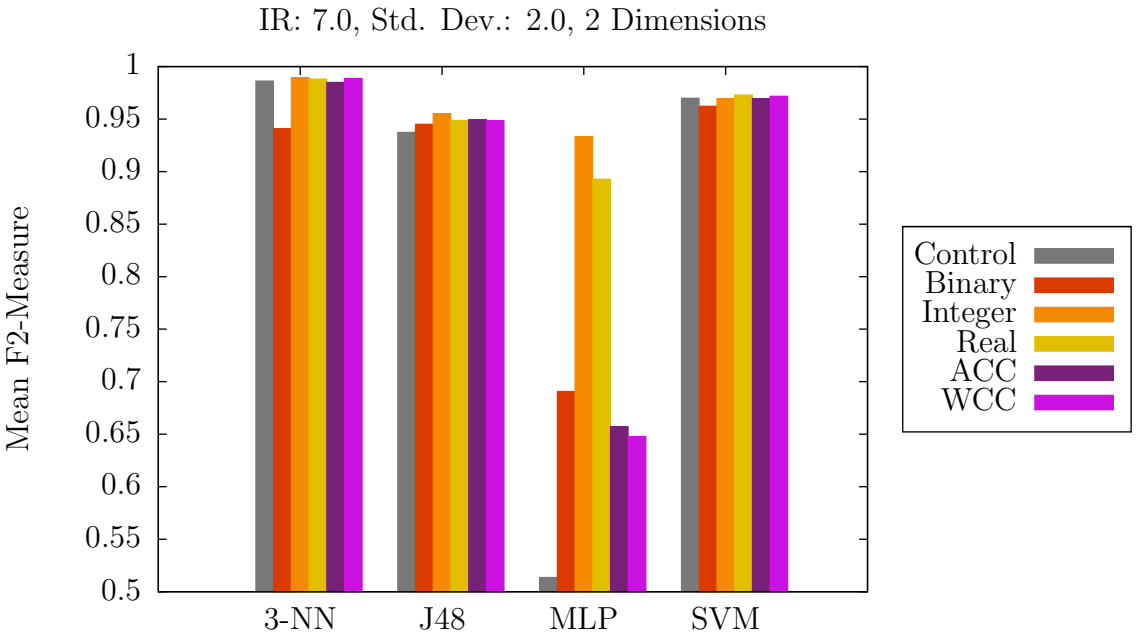

IR: 3.0, Std. Dev.: 4.0, 8 Dimensions
Mean F2-Measure
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

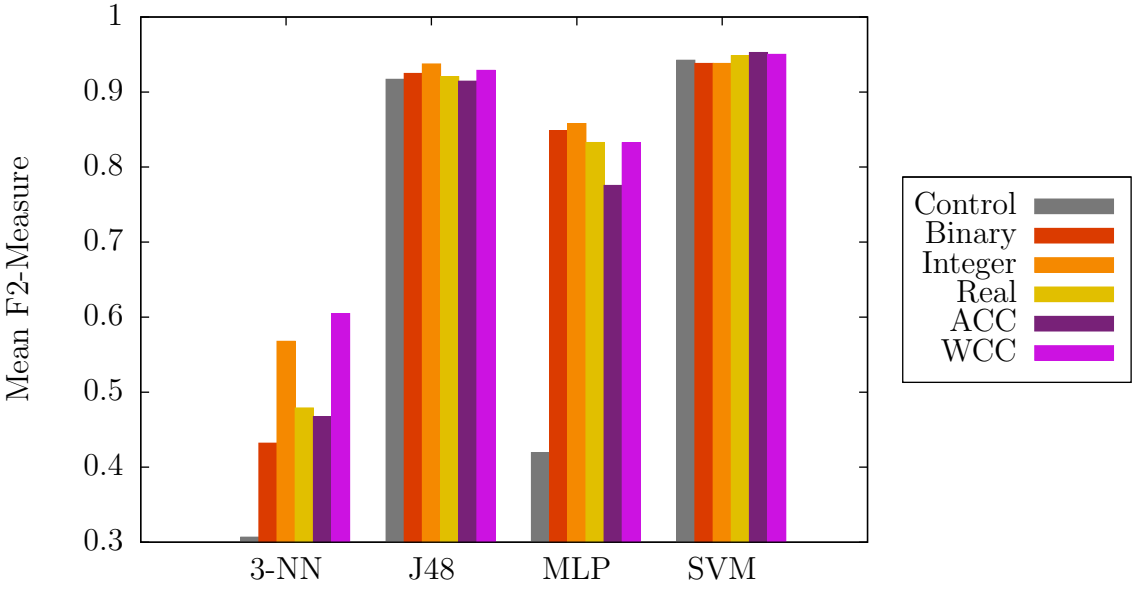

IR: 3.0, Std. Dev.: 4.0, 16 Dimensions
Mean F2-Measure
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

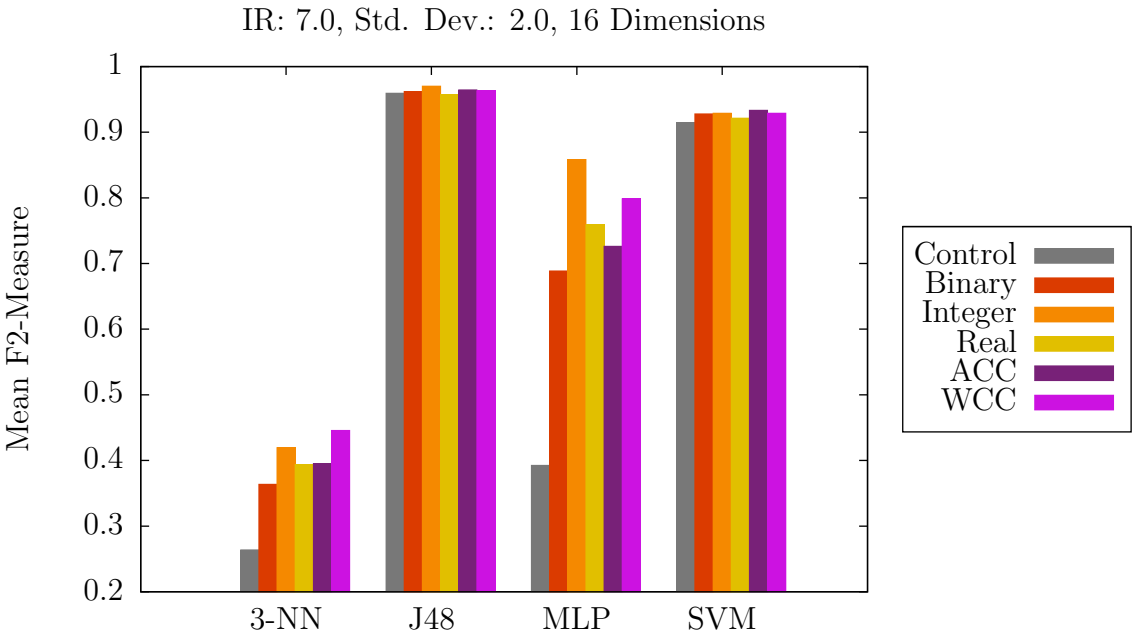

IR: 3.0, Std. Dev.: 5.0, 2 Dimensions
Mean F2-Measure
0.7
0.68
0.66
0.64
0.62
0.6
0.58
0.56
0.54
0.52
0.5
0.48
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

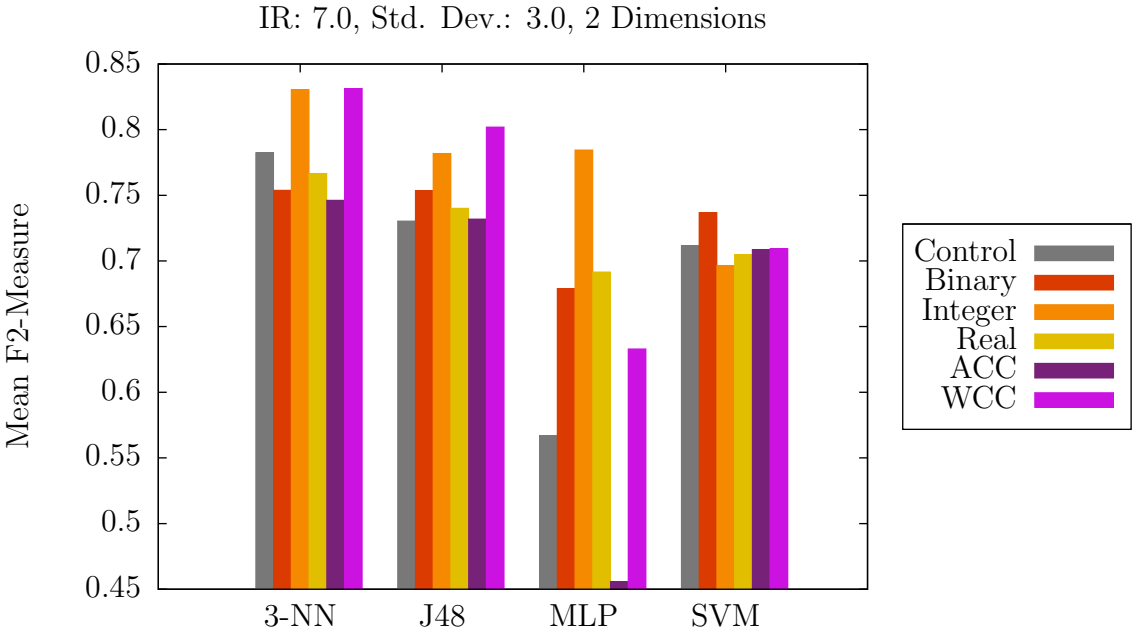

IR: 3.0, Std. Dev.: 5.0, 8 Dimensions
Mean F2-Measure
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

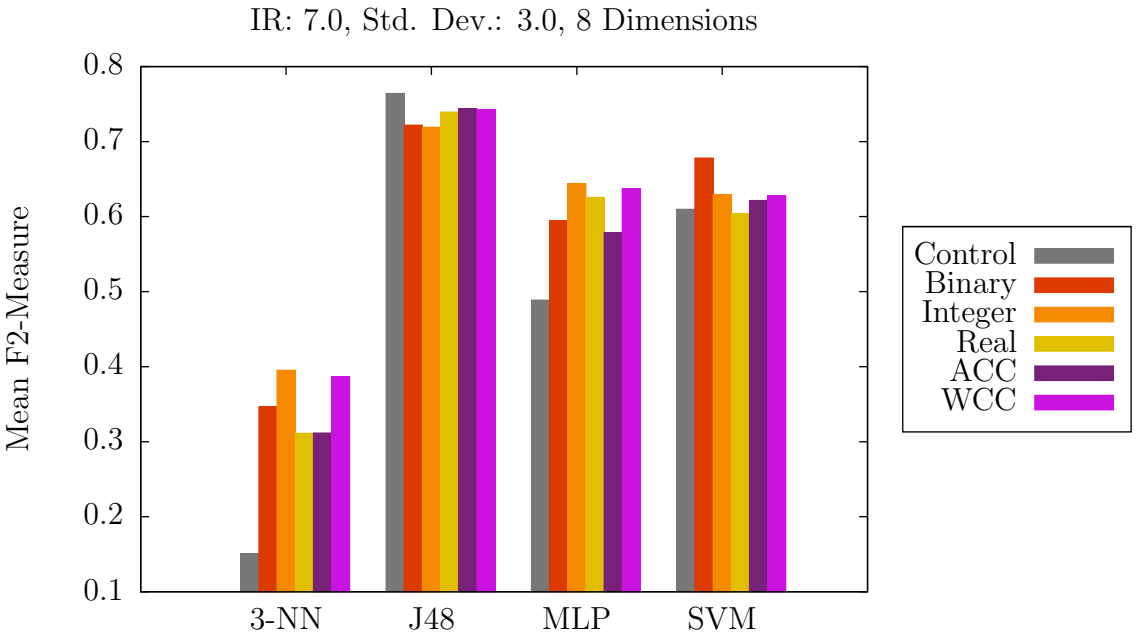

IR: 3.0, Std. Dev.: 5.0, 16 Dimensions
Mean F2-Measure
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

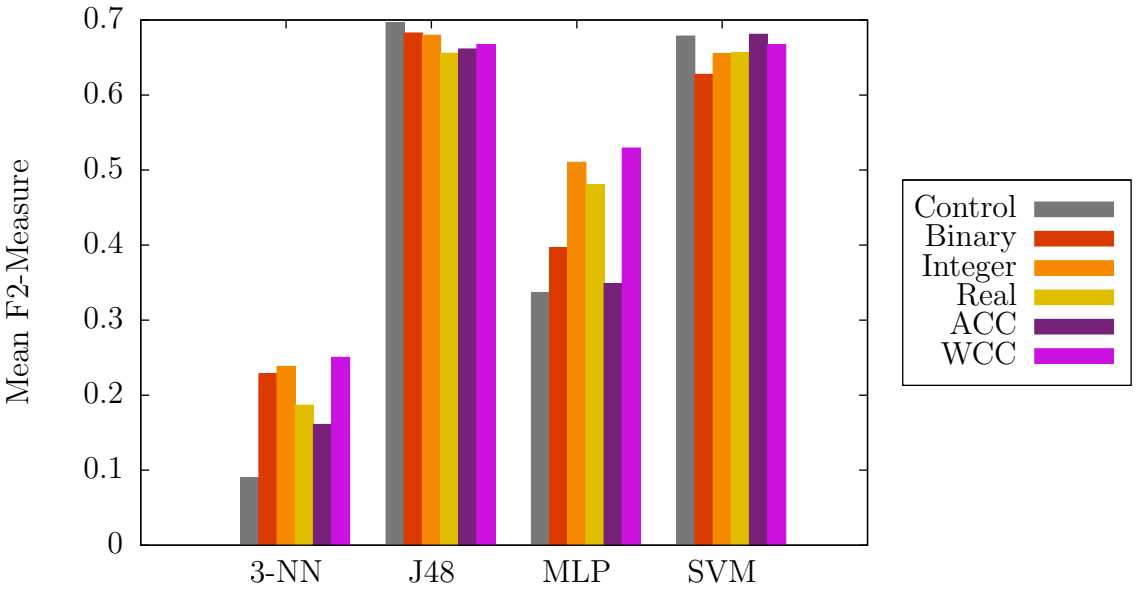

IR: 7.0, Std. Dev.: 2.0, 2 Dimensions
Mean F2-Measure
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

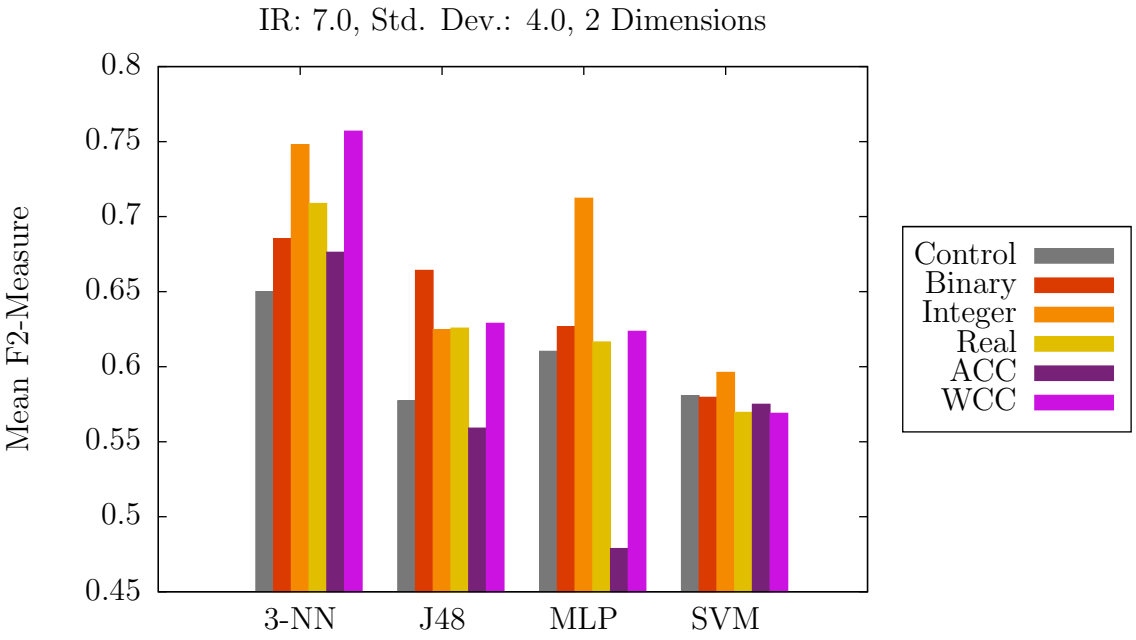

IR: 7.0, Std. Dev.: 2.0, 8 Dimensions
Mean F2-Measure
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

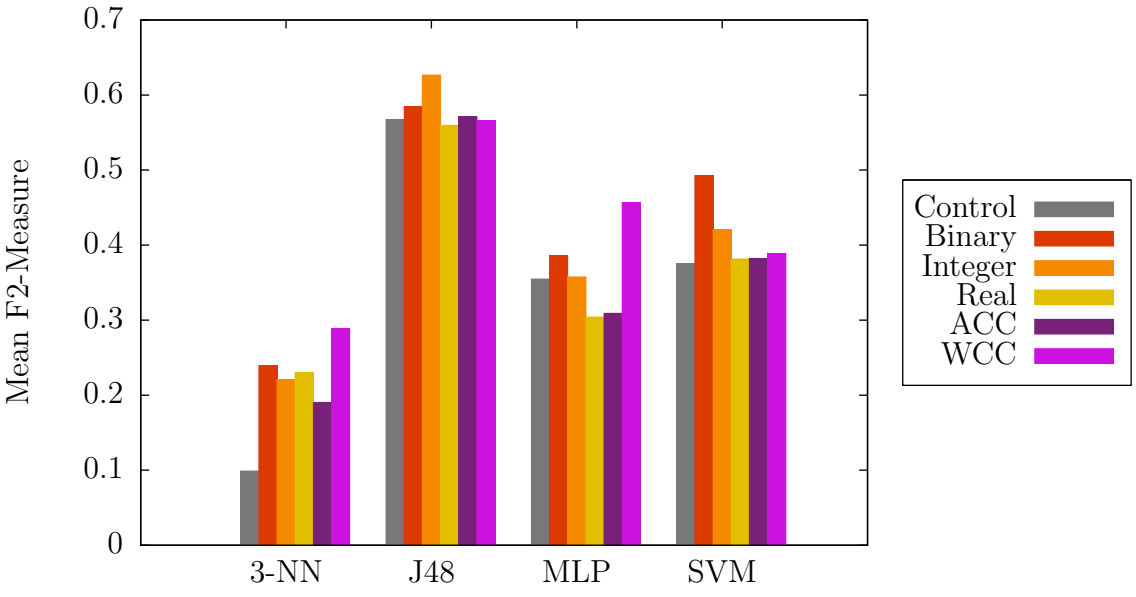

IR: 7.0, Std. Dev.: 2.0, 16 Dimensions
Mean F2-Measure
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

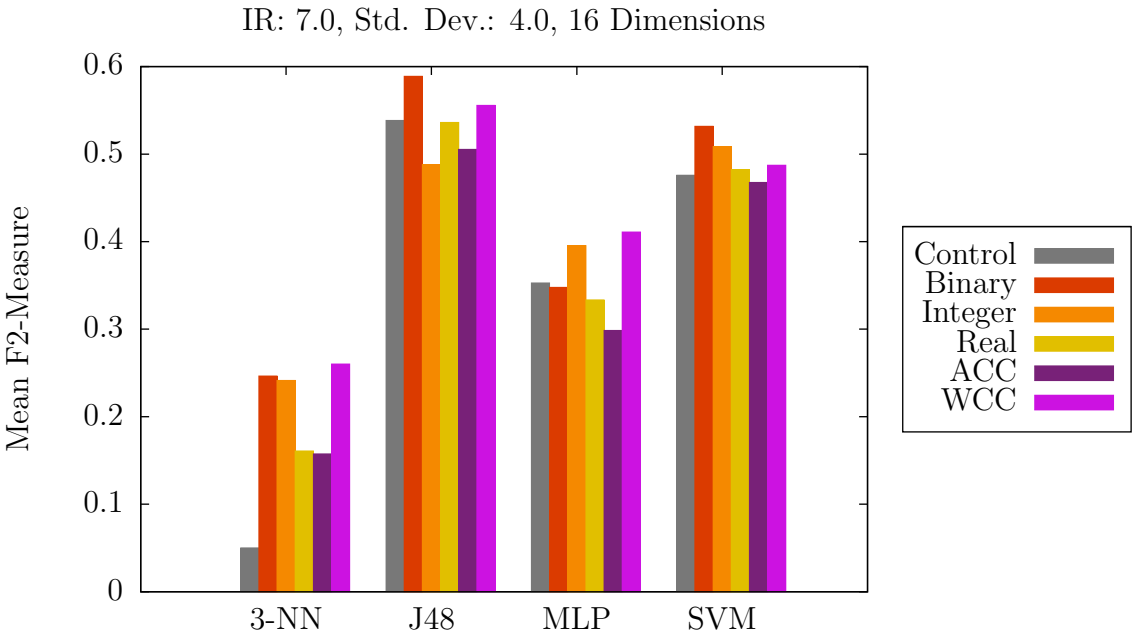

IR: 7.0, Std. Dev.: 3.0, 2 Dimensions
Mean F2-Measure
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

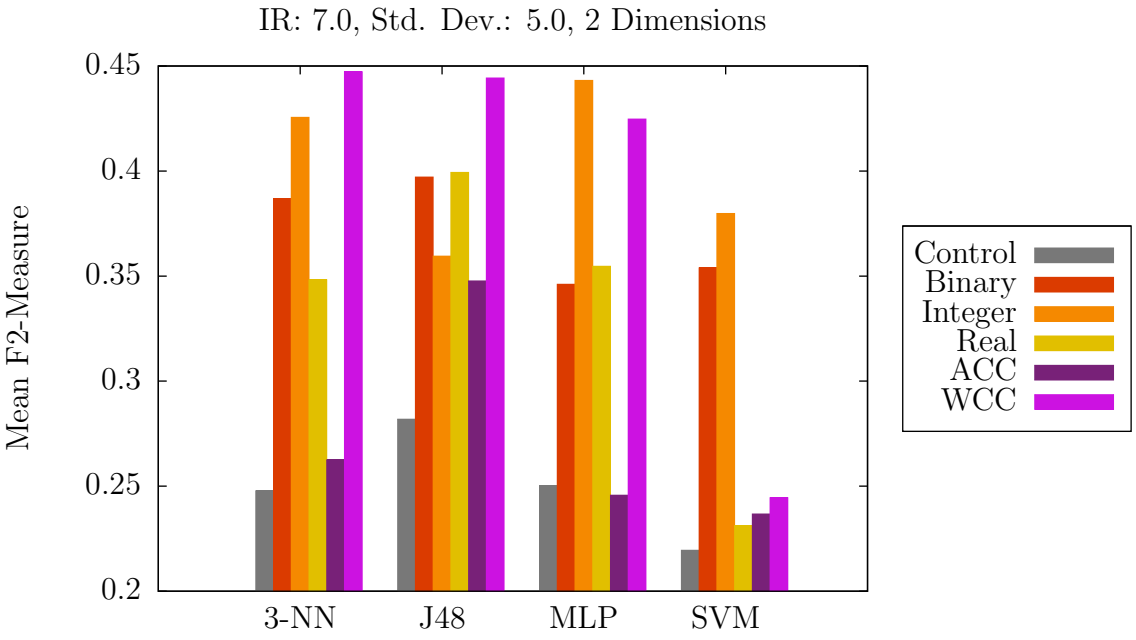

IR: 7.0, Std. Dev.: 3.0, 8 Dimensions
Mean F2-Measure
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

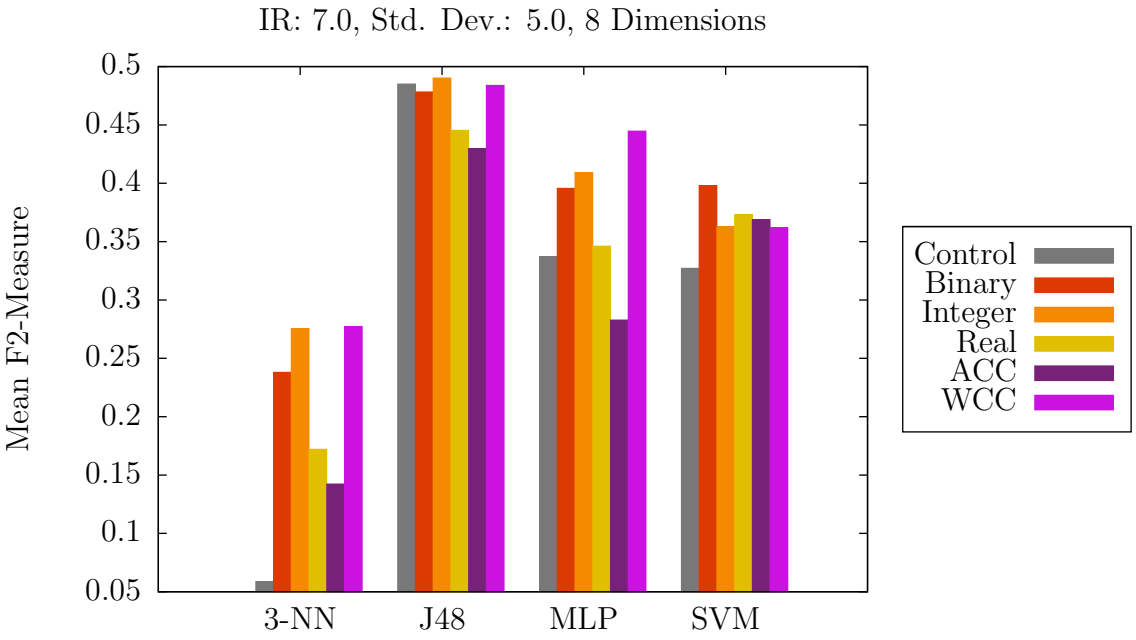

IR: 7.0, Std. Dev.: 3.0, 16 Dimensions
Mean F2-Measure
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

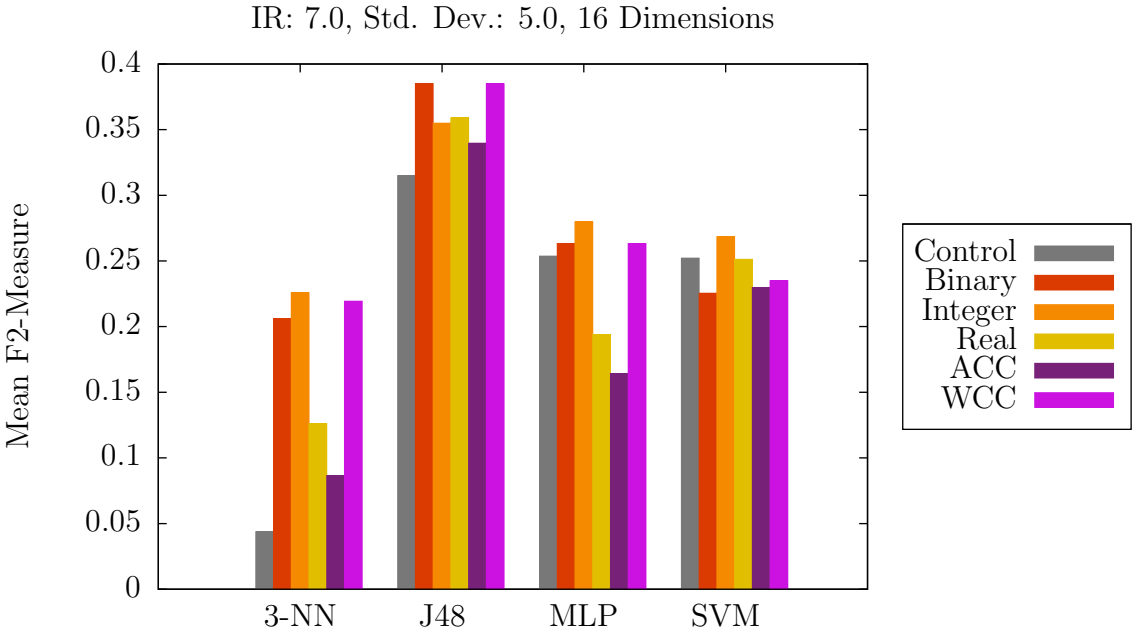

IR: 7.0, Std. Dev.: 4.0, 2 Dimensions
Mean F2-Measure
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

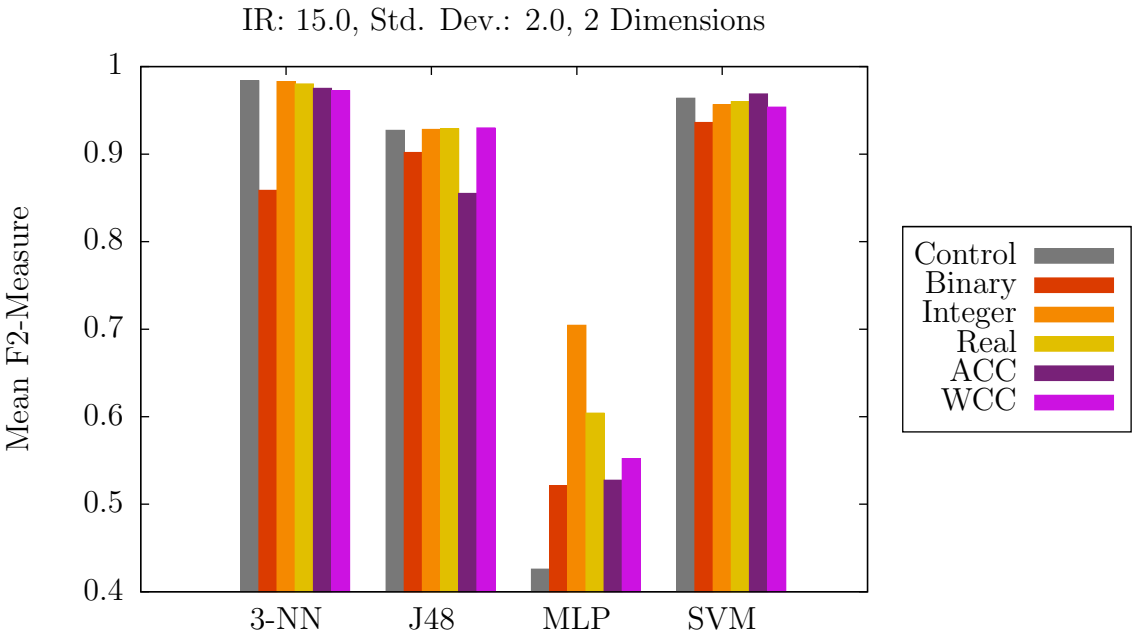

IR: 7.0, Std. Dev.: 4.0, 8 Dimensions
Mean F2-Measure
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

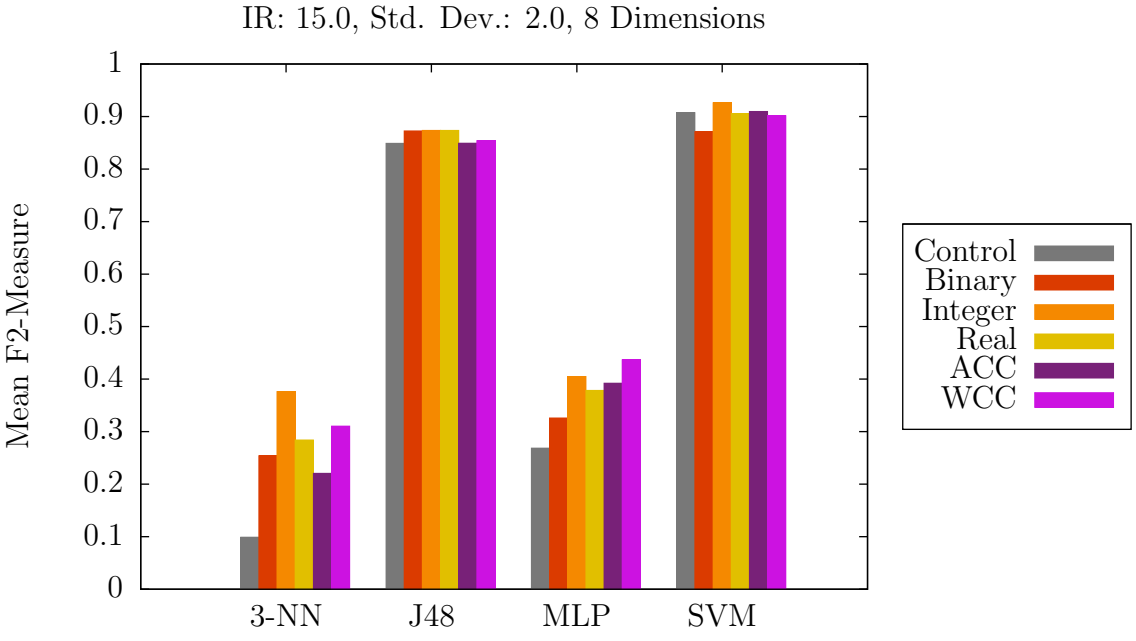

IR: 7.0, Std. Dev.: 4.0, 16 Dimensions
Mean F2-Measure
0.6
0.5
0.4
0.3
0.2
0.1
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

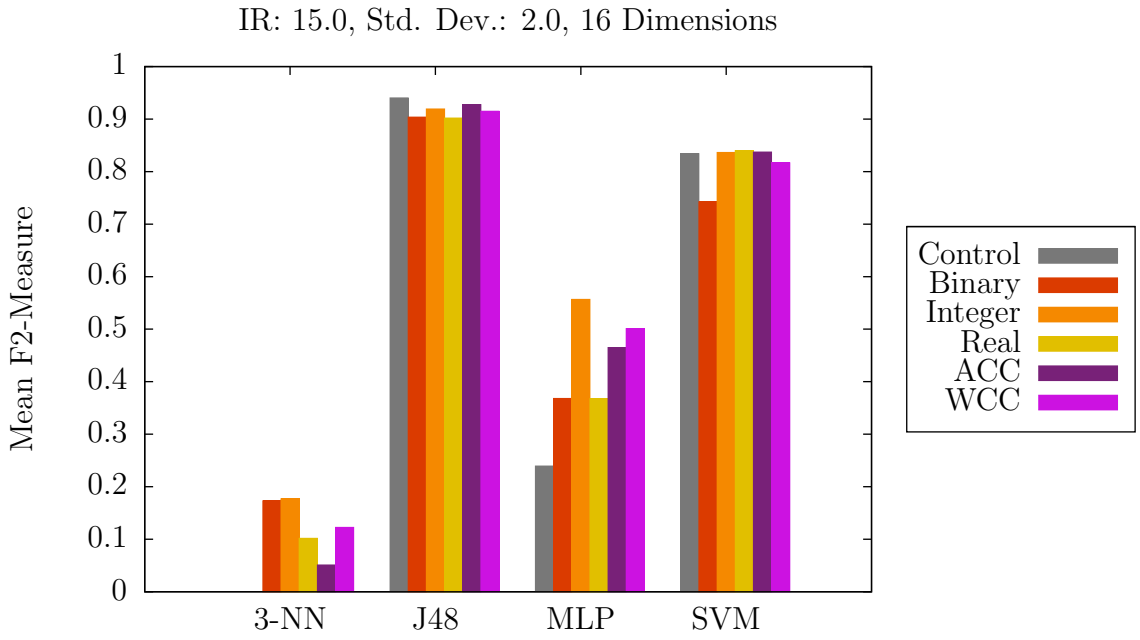

IR: 7.0, Std. Dev.: 5.0, 2 Dimensions
Mean F2-Measure
0.45
0.4
0.35
0.3
0.25
0.2
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

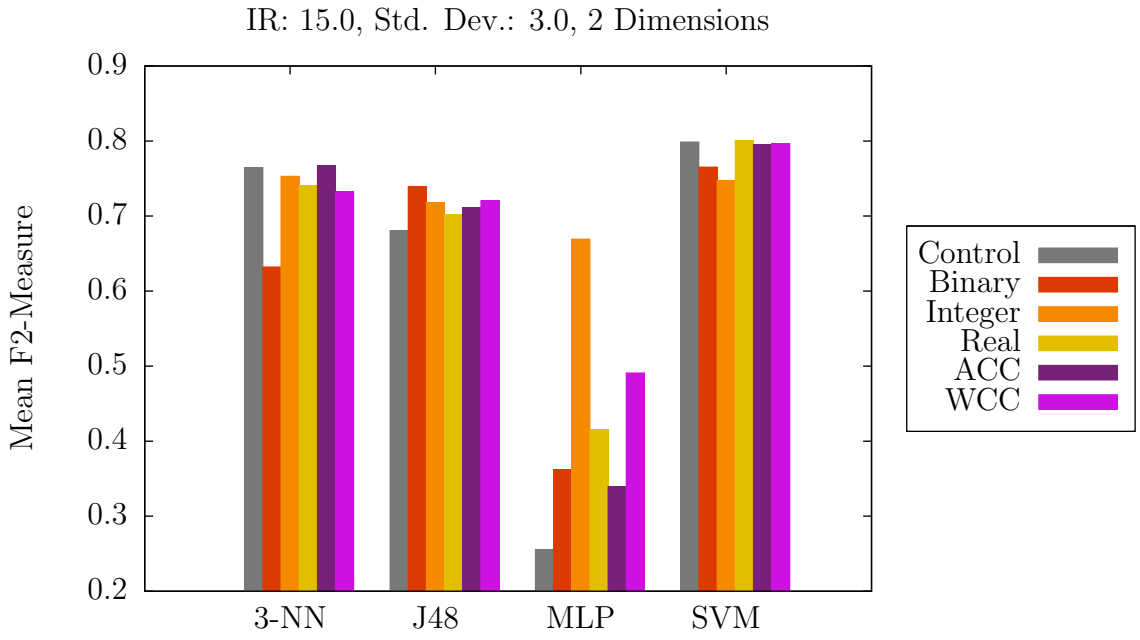

IR: 7.0, Std. Dev.: 5.0, 8 Dimensions
Mean F2-Measure
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

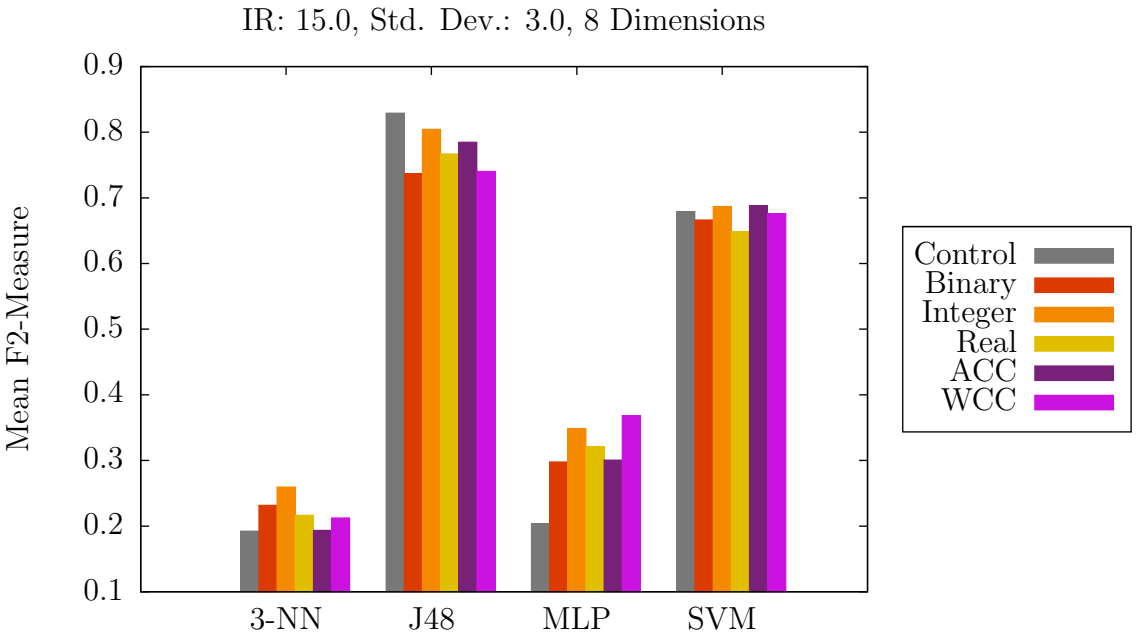

IR: 7.0, Std. Dev.: 5.0, 16 Dimensions
Mean F2-Measure
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

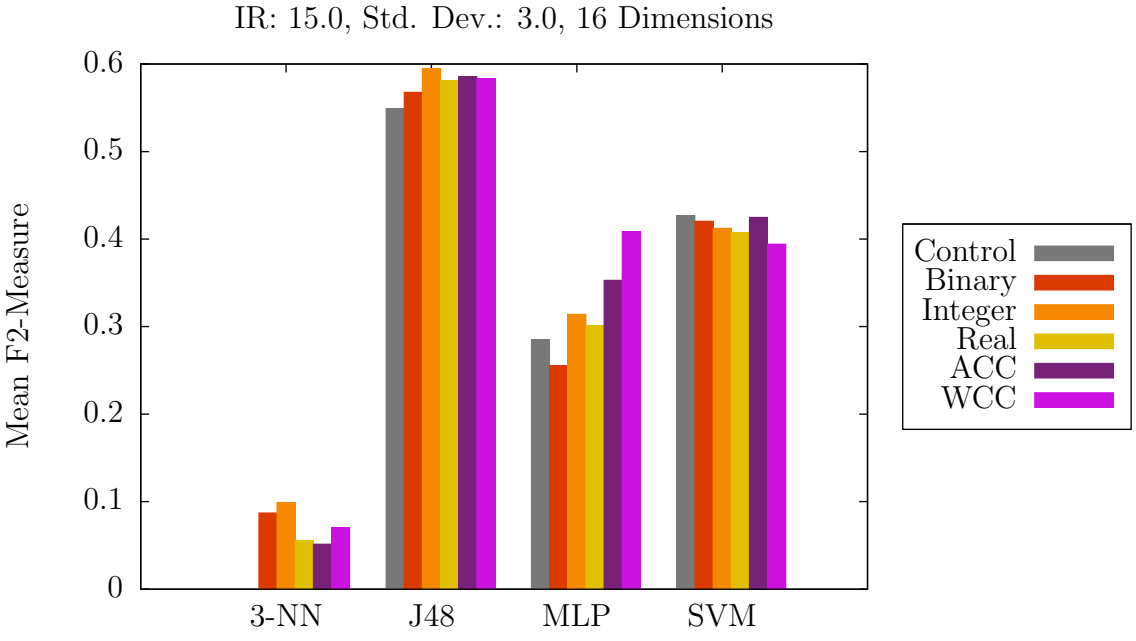

IR: 15.0, Std. Dev.: 2.0, 2 Dimensions
Mean F2-Measure
1
0.9
0.8
0.7
0.6
0.5
0.4
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

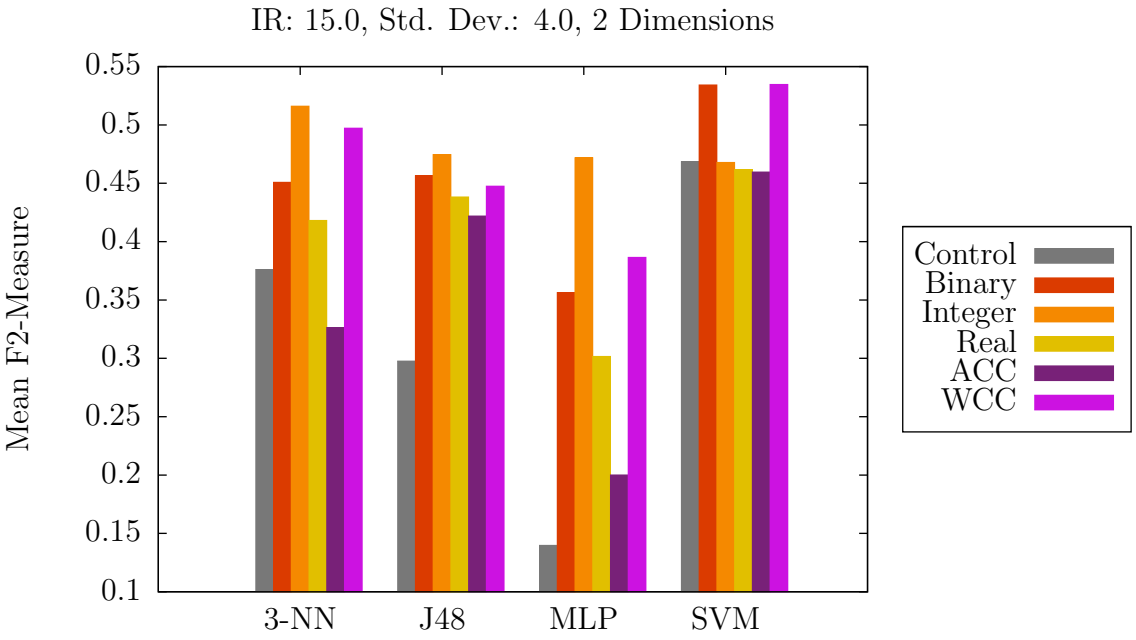

IR: 15.0, Std. Dev.: 2.0, 8 Dimensions
Mean F2-Measure
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

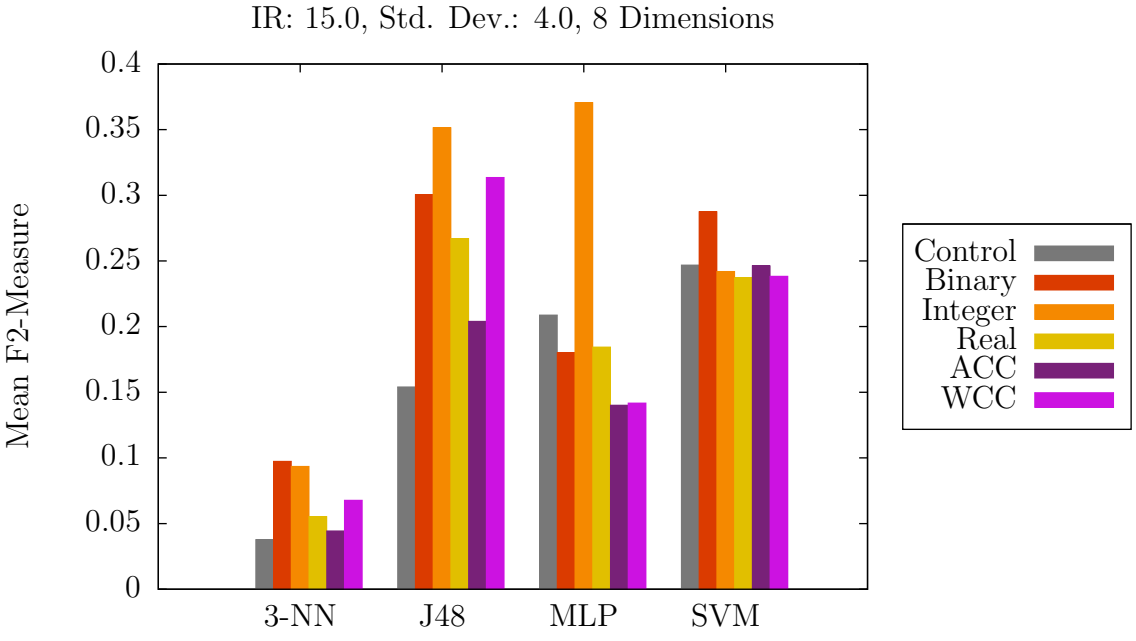

IR: 15.0, Std. Dev.: 2.0, 16 Dimensions
Mean F2-Measure
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

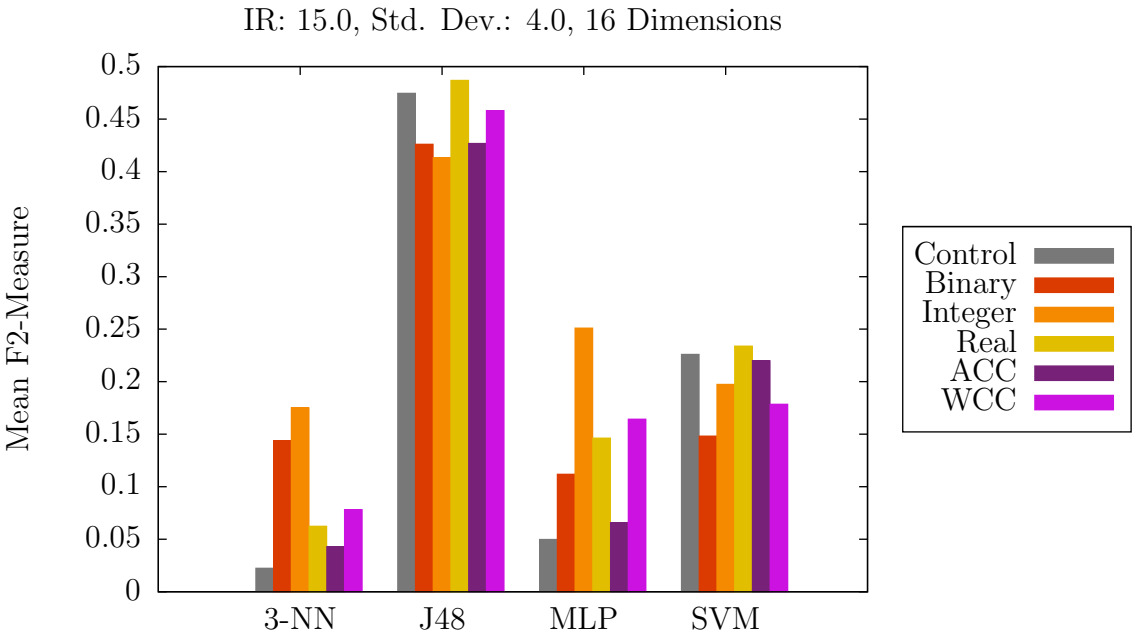

IR: 15.0, Std. Dev.: 3.0, 2 Dimensions
Mean F2-Measure
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

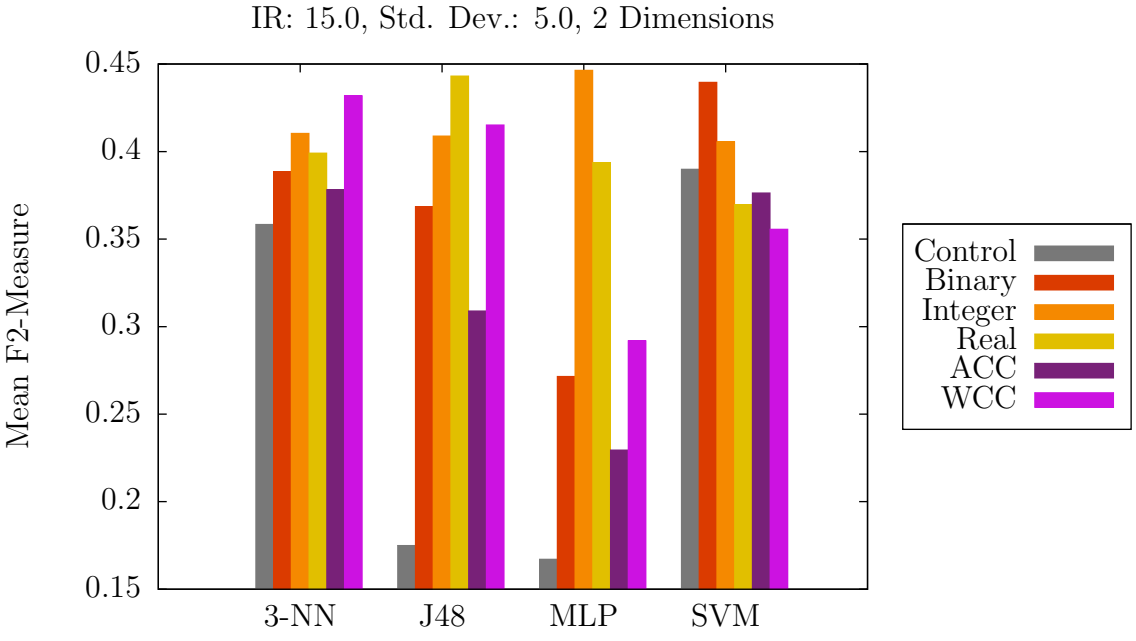

IR: 15.0, Std. Dev.: 3.0, 8 Dimensions
Mean F2-Measure
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

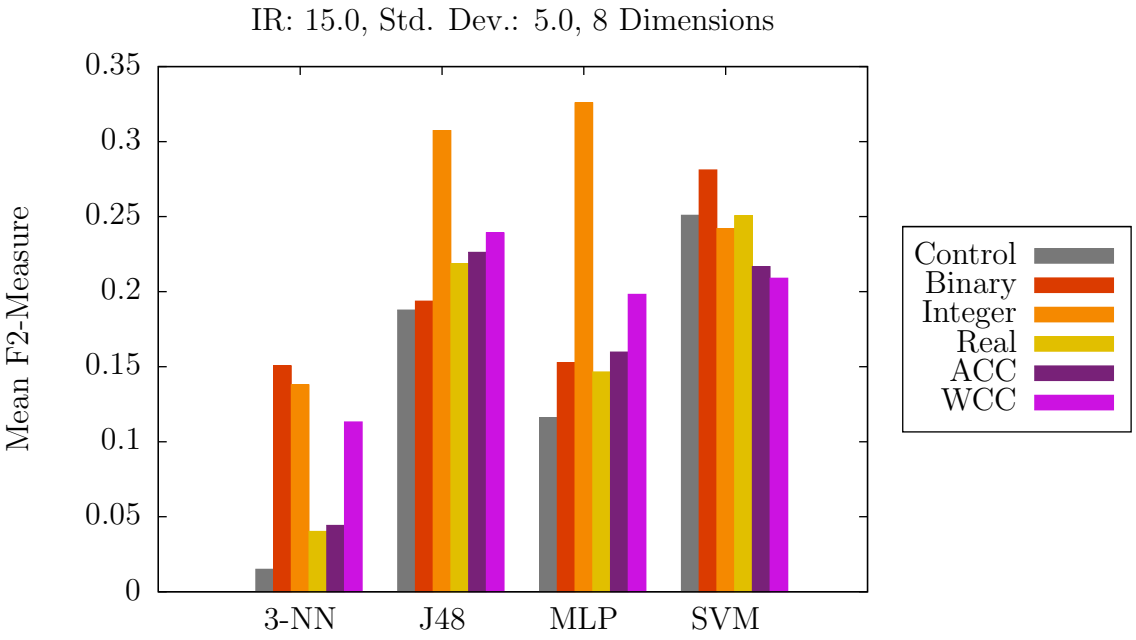

IR: 15.0, Std. Dev.: 3.0, 16 Dimensions
Mean F2-Measure
0.6
0.5
0.4
0.3
0.2
0.1
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

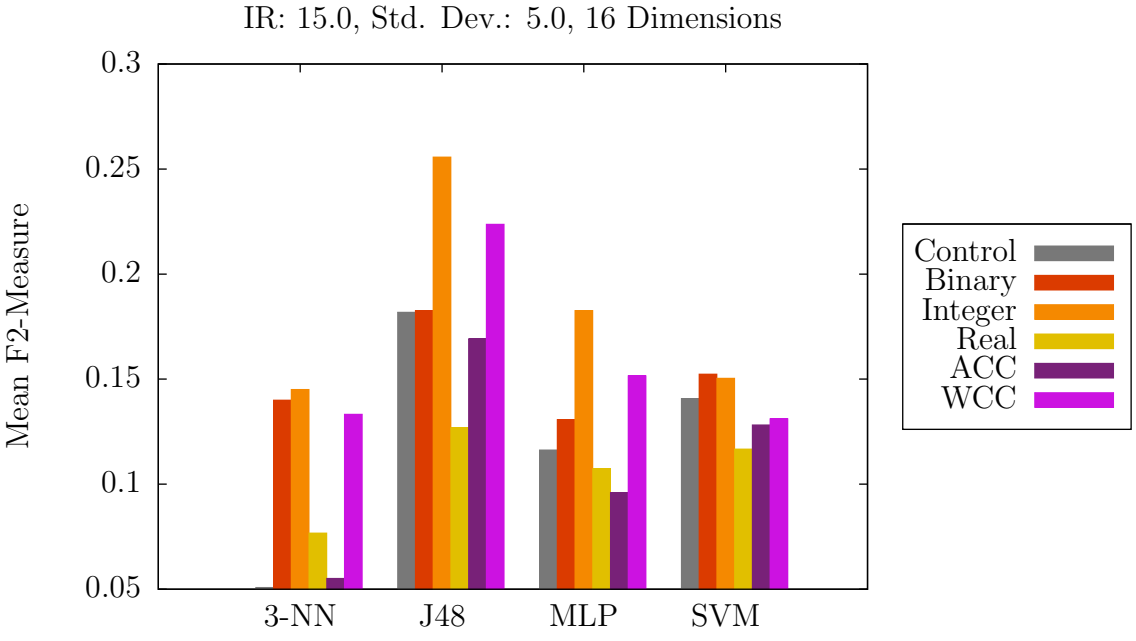

IR: 15.0, Std. Dev.: 4.0, 2 Dimensions
Mean F2-Measure
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC


IR: 15.0, Std. Dev.: 4.0, 8 Dimensions
Mean F2-Measure
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC


IR: 15.0, Std. Dev.: 4.0, 16 Dimensions
Mean F2-Measure
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC


IR: 15.0, Std. Dev.: 5.0, 2 Dimensions
Mean F2-Measure
0.45
0.4
0.35
0.3
0.25
0.2
0.15
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

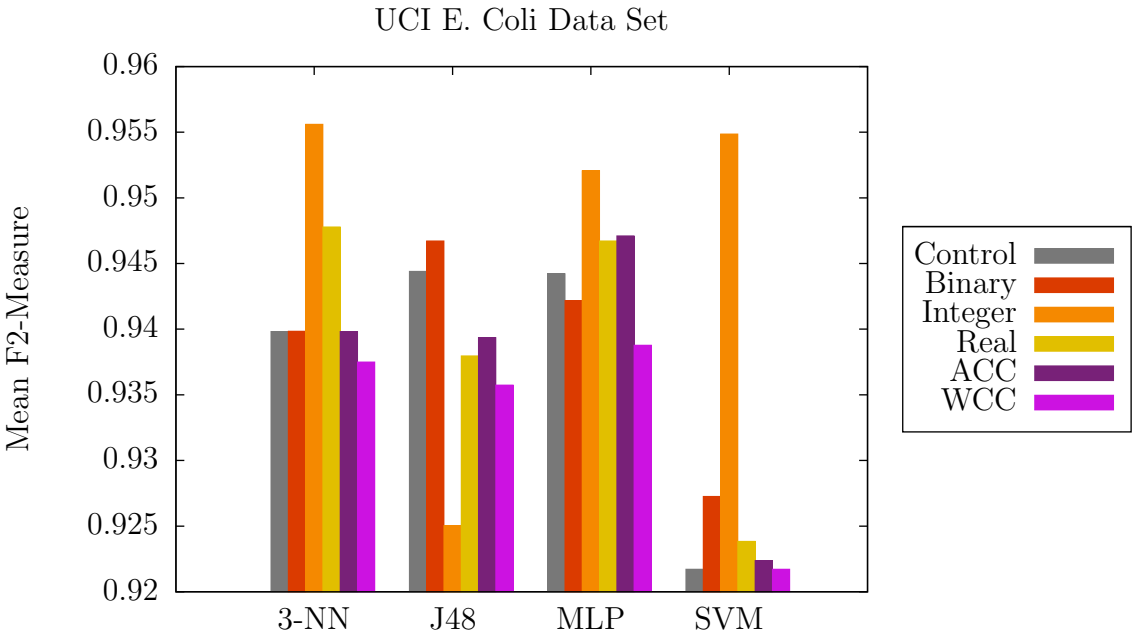

IR: 15.0, Std. Dev.: 5.0, 8 Dimensions
Mean F2-Measure
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

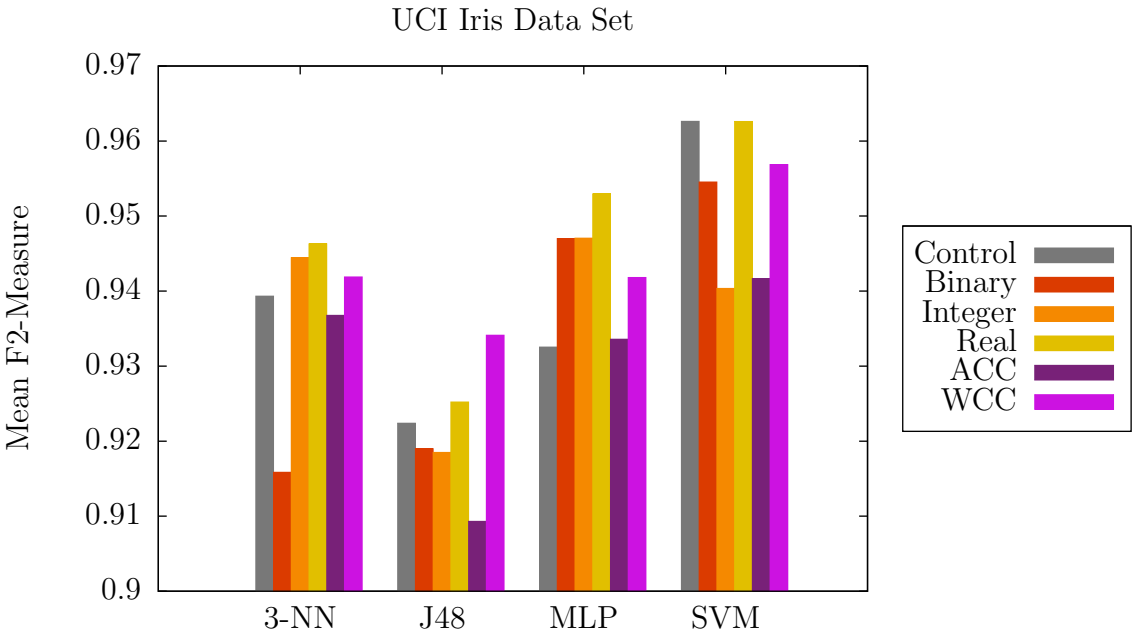

IR: 15.0, Std. Dev.: 5.0, 16 Dimensions
Mean F2-Measure
0.3
0.25
0.2
0.15
0.1
0.05
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

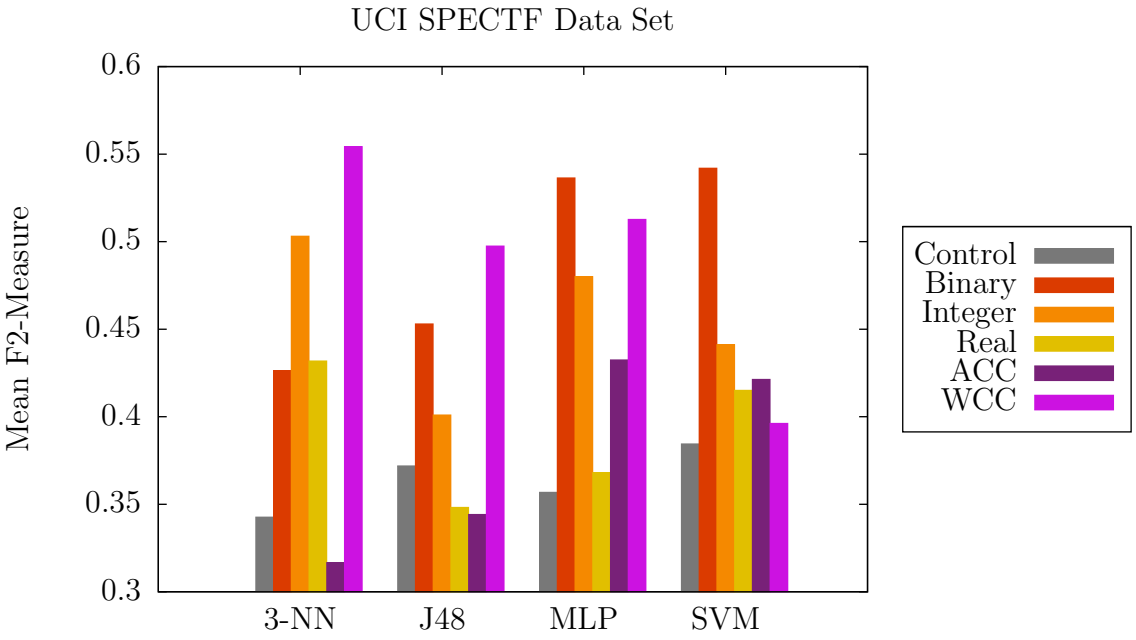
Appendix B

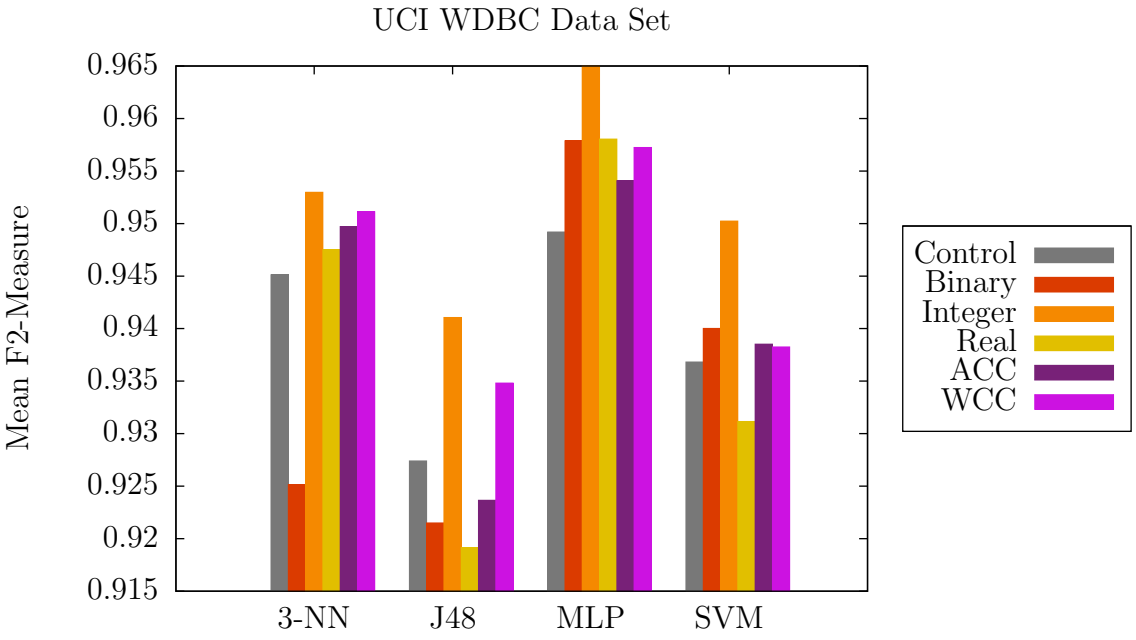
UCI Data Performance Plots

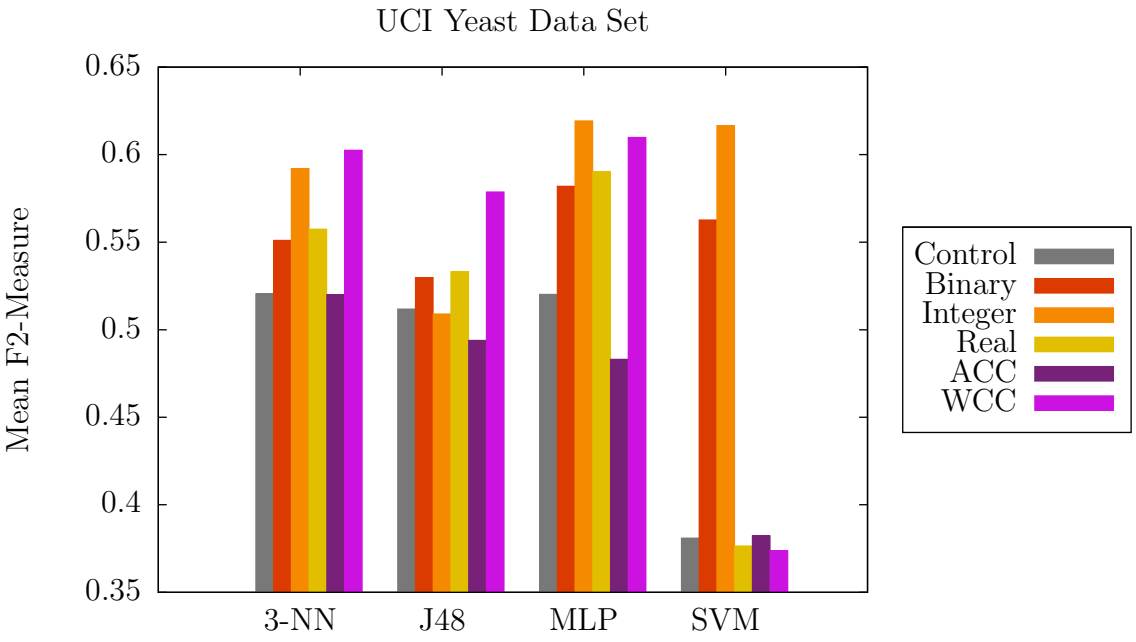
This appendix contains graphical representations of each classifier/pre-processor combination's performance on each UCI data set. The mean F2-measures plotted here are the same as those in Table 5.9, Table 5.10, Table 5.11, and Table 5.12. They are provided in this form for easier comparison. The graphs are arranged alphabetically data set name.


UCI E. Coli Data Set
Mean F2-Measure
0.96
0.955
0.95
0.945
0.94
0.935
0.93
0.925
0.92
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC


UCI Iris Data Set
Mean F2-Measure
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.9
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC


UCI SPECTF Data Set
Mean F2-Measure
0.6
0.55
0.5
0.45
0.4
0.35
0.3
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC


UCI WDBC Data Set
Mean F2-Measure
0.965
0.96
0.955
0.95
0.945
0.94
0.935
0.93
0.925
0.92
0.915
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC


UCI Yeast Data Set
Mean F2-Measure
0.65
0.6
0.55
0.5
0.45
0.4
0.35
3-NN
J48
MLP
SVM
Control
Binary
Integer
Real
ACC
WCC

Appendix C

Omitted UCI Instances

Table C.1: Omitted Instances from the Iris data set

Sepal Width	Sepal Length	Petal Length	Petal Width	Label
6.3	3.3	6.0	2.5	1
6.1	2.6	5.6	1.4	1
5.1	3.5	1.4	0.2	0
5.7	3.8	1.7	0.3	0
5.1	3.8	1.5	0.3	0
4.4	3.0	1.3	0.2	0

Table C.2: Omitted instances from the Yeast data set

mcg	gvh	alm	mit	erl	pox	vac	nuc	Label
0.58	0.49	0.50	0.41	0.50	0.00	0.57	0.22	1
0.46	0.44	0.53	0.31	0.50	0.00	0.48	0.22	1
0.50	0.57	0.62	0.25	0.50	0.00	0.51	0.22	1
0.46	0.46	0.56	0.12	0.50	0.00	0.55	0.22	1
0.33	0.37	0.54	0.28	0.50	0.00	0.53	0.27	0
0.53	0.52	0.49	0.27	0.50	0.00	0.51	0.22	0
0.77	0.60	0.42	0.33	0.50	0.00	0.57	0.22	0
0.49	0.43	0.51	0.13	0.50	0.00	0.47	0.22	0
0.53	0.68	0.34	0.42	0.50	0.00	0.54	0.27	0
0.48	0.46	0.38	0.29	0.50	0.00	0.53	0.22	0
0.44	0.49	0.49	0.27	0.50	0.00	0.52	0.22	0
0.34	0.58	0.50	0.26	0.50	0.00	0.51	0.22	0

Table C.3: Omitted instances from the WDBC data set

ID Number	Label
855167	1
8712729	1
8669691	1
892189	1
8910499	0
862485	0
905520	0
89869	0
8939988	0

Table C.4: Omitted instances from the SPECTF data set

F1R	F1S	F2R	F2S	F3R	F3S	F4R	F4S	F5R	F5S	F6R	F6S	F7R	F7S	F8R	F8S	F9R	F9S	F10R	F10S	F11R	F11S	F12R	F12S	F13R	F13S	F14R	F14S	F15R	F15S	F16R	F16S	F17R	F17S	F18R	F18S	F19R	F19S	F20R	F20S	F21R	F21S	F22R	F22S	Label
62	67	68	70	65	70	73	77	69	70	69	73	71	74	71	71	76	75	66	67	73	73	70	74	63	67	58	68	66	69	78	79	69	70	71	73	72	71	73	77	72	76	64	66	1
72	74	67	69	69	70	74	81	66	70	73	78	69	80	65	65	70	77	69	70	73	79	64	68	56	58	67	64	68	68	69	74	62	67	66	70	73	77	74	77	71	72	63	65	1
61	76	71	68	77	69	77	69	64	75	71	81	75	72	71	69	70	73	61	71	69	79	64	65	62	66	61	65	71	68	67	71	59	64	66	65	60	68	74	71	69	68	63	59	1
75	75	70	77	67	75	75	75	67	66	74	73	68	72	64	70	76	70	67	63	74	75	72	68	69	68	75	69	71	74	75	76	63	70	71	69	66	63	70	73	66	68	58	59	1
66	67	63	70	69	70	73	72	61	62	68	68	70	71	71	67	71	69	65	65	76	78	71	70	65	64	68	71	70	71	73	74	58	63	68	71	70	72	77	79	79	79	66	66	1
72	70	75	80	73	70	76	73	66	56	72	70	73	75	67	65	69	69	73	72	81	77	80	79	67	64	64	66	69	68	70	75	63	59	66	63	74	77	81	78	79	75	65	66	1
71	71	69	71	65	65	76	73	67	66	69	79	76	76	63	62	74	71	58	66	76	78	78	74	68	66	69	68	68	68	71	74	59	57	65	66	73	71	78	74	68	70	57	55	1
69	68	73	74	62	67	74	73	67	65	70	75	64	68	58	57	66	70	57	63	68	71	68	69	40	38	55	59	67	67	76	78	65	67	61	65	76	79	74	73	64	56	53	45	0
73	73	75	79	64	66	68	61	71	68	77	75	63	73	62	55	70	64	77	75	75	79	70	75	65	59	68	56	63	67	74	73	60	60	62	60	73	77	70	72	73	67	64	61	0
68	63	67	67	65	72	74	72	70	71	79	71	72	67	68	69	75	79	67	65	78	69	72	67	64	59	67	65	73	70	80	69	63	61	70	70	70	67	77	71	77	72	68	59	0
65	62	67	68	65	67	71	71	64	56	73	72	68	69	56	57	67	62	74	66	80	76	80	78	53	47	48	36	68	65	74	73	60	60	67	63	74	63	77	79	68	70	59	56	0