ENHANCING THE QUALITY OF HIGH DIMENSIONAL NOISY DATA FOR CLASSIFICATION AND REGRESSION PROBLEMS

by

BOSEON BYEON

(Under the Direction of Khaled Rasheed)

ABSTRACT

In classification and regression problems, classifiers for high dimensional noisy data suffer from the concurrent negative effects of noise and high dimensionality. Noise disrupts data and high dimensionality prevents the classifier from focusing on relevant features; potentially reducing classification and regression accuracies. However, most noise detection techniques cannot be used for high dimensional data and many dimensionality reduction methods are not applicable to noisy data. The goal of this dissertation is to enhance the quality of high dimensional noisy data by simultaneously removing noise and providing relevant features. To achieve that we propose the NDFS algorithm which relies on two genetic algorithms, one for noise detection (GA-ND) and the other for feature selection (GA-FS), which exchange their results periodically at certain generation intervals. Also prototype selection (PS-ND) is used together with the genetic algorithm to improve the performance of the noise detection part. Our experimental results show that while the sequential application of noise detection and feature selection methods may not overcome the concurrent negative effects of noise and high dimensionality, the NDFS algorithm succeeds in this and achieves high performance by simultaneous noise removal and feature selection. We demonstrate that the NDFS algorithm substantially increases the classification

accuracies and reduces the error rates, and show that it significantly enhances the quality of high dimensional noisy data for both classification and regression problems.

INDEX WORDS: Noise, Outlier, Prototype, High Dimensionality, Noise Detection, Prototype Selection, Dimensionality Reduction

ENHANCING THE QUALITY OF HIGH DIMENSIONAL NOISY DATA FOR CLASSIFICATION AND REGRESSION PROBLEMS

by

BOSEON BYEON

B.A., Korea University, Seochang, Korea, 1995

M.A., Yonsei University, Seoul, Korea, 1997

M.S., Long Island University, Brooklyn, 2002

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

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2009

ENHANCING THE QUALITY OF HIGH DIMENSIONAL NOISY DATA FOR CLASSIFICATION AND REGRESSION PROBLEMS

by

BOSEON BYEON

Major Professor: Khaled Rasheed

Committee: Maria Hybinette
Walter D. Potter
William P. McCormick

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2009

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

In classification and regression problems, classifiers for high dimensional noisy data suffer from the concurrent negative effects of noise and high dimensionality. Noise disrupts data and high dimensionality prevents classifier from focusing on relevant features. Therefore they may reduce classification and regression accuracies. In the machine learning field, there are two primary topics related to this problem: *noise detection* and *dimensionality reduction*. Noise detection is the process to identify noise in data, and dimensionality reduction is the procedure to reduce the number of features and to identify relevant features.

Noise refers to incorrect or erroneous values in the data. It is closely related to *outlier*, another popular term. A common definition of an outlier is given by Barnett and Lewis [5] as an instance that appears to be inconsistent with the remaining instances in the data. Noise, outlier, error, and exception are terms frequently used to describe the same or similar concept. Noise may exist in regular attributes (features) or the target attribute. Witten and Frank [68] note that a classifier learns how to use *feature noise* to build a more accurate model, and removing feature noise may reduce the performance of a classifier. However, *target noise* rather than feature noise disrupts data and training a classifier on target-noise-free data may increase its performance [68].

We define noise as incorrect or erroneous target value. Hodge and Austin [27] indicate that many real world data contain noise because of human errors, instrument errors, system errors or malicious human behavior. Noise may disrupt data and lead a classifier to build an incorrect model. Therefore, detecting and removing noise from data will probably increase the

accuracy of a classifier by guiding it to build a more accurate model. A large amount of research effort has been devoted to noise detection. We briefly introduce noise detection methods in Chapter 2. Despite the large amount of work, there is no universally accepted noise detection method [27].

In high dimensional domains, noise detection is quite difficult due to the concurrent negative effect of high dimensionality. High dimensionality suggests that data includes many features irrelevant or redundant to the target. Most noise detection algorithms do not provide dimensionality reduction. Such noise detection algorithms without dimensionality reduction suffer from *the curse of dimensionality* that a classifier faces when managing high dimensional data. We describe the curse of dimensionality in Chapter 3. Reducing dimensionality and providing relevant features to a classifier will probably increase the performance of the classifier by curing the curse of dimensionality and guiding the classifier to focus on relevant features. In addition to the curse of dimensionality, Chapter 3 includes previous related work on dimensionality reduction.

Most noise detection techniques cannot be used for high dimensional data and many dimensionality reduction methods are not applicable to noisy data. The goal of this dissertation is to enhance the quality of high dimensional noisy data by simultaneously removing noise and providing relevant features for classification and regression problems.

We propose the NDFS algorithm in this dissertation. NDFS relies on two genetic algorithms, one for noise detection (GA-ND) and the other for feature selection (GA-FS), which exchange their results periodically at certain generation intervals. Also prototype selection (PS-ND) is used together with the genetic algorithm to improve the performance of the noise detection method.

In addition to previous work on noise detection in Chapter 2 and dimensionality reduction in Chapter 3, we introduce the proposed NDFS algorithm for simultaneous noise removal and feature selection in Chapter 4. The three main components of NDFS: GA-ND, PS-ND and GA-FS, are explained in detail and their pseudo-codes are provided.

In Chapter 5, we describe how our synthetic data sets are generated and what properties they have. Also we present the parameter settings used for NDFS.

In Chapter 6, experimental results of the NDFS algorithm are discussed. While the sequential application of noise detection and feature selection methods may not overcome the concurrent negative effects of noise and high dimensionality, the NDFS algorithm succeeds in this and achieves high performances by simultaneous noise removal and feature selection.

Finally Chapter 7 concludes the dissertation with a brief summary and a discussion of the drawbacks of the NDFS algorithm.

CHAPTER 2

NOISE DETECTION AND NOISE HANDLING

Categories of Noise Detection Approaches

There are two methodologies for categorizing noise detection methods. The first methodology is to classify the detection methods by the availability of data. According to the availability of data, detection methods are divided into three approaches: supervised, semi-supervised, and unsupervised detection approaches ([15] and [27]). The *supervised* detection approach uses data pre-labeled as *normal* or *noise*. A classifier builds a model that classifies instances as normal or noisy, and decides which class the testing data fall into on the basis of this model. While this type of noise detection method may have high accuracy, fully labeled data are not available in many cases. The *semi-supervised* detection approach is known as *novelty detection*. In these detection methods, only normal data are available and a classifier defines a *boundary of normality*. Any testing data within this boundary of normality is considered normal; otherwise, it is identified as noise. However it is not easy to collect enough data for all the possible boundaries of normality. In the *unsupervised* detection approach, a classifier does not have any knowledge of normal and noisy data. The basic idea of unsupervised detection methods is that noisy data can be separated from normal data by some criteria. These methods suffer from low accuracies.

The second methodology for categorizing noise detection methods is by the underlying *detection algorithm*. We use this methodology to provide related previous work on noise detection. In the next section, we divide noise detection methods into seven approaches:

distribution-based, distance-based, cluster-based, classifier-based, misclassification-based, prototype-based, and other approaches.

Noise Detection Approaches

1. Distribution-based Approaches

Assuming the data were generated from some distribution such as Gaussian or Poisson, these approaches propose an appropriate distribution. If instances significantly deviate from the proposed distribution, they are detected as noise. The drawback of the approaches is that finding an appropriate distribution is expensive and difficult [14].

Grubb's test is used to detect noise for univariate data on the basis of the *normality assumption* ([14] and [15]). The test statistic is the absolute difference between the value for the feature and the average of the data for the feature divided by the standard deviation of the data for the feature: $Grubs = |x - mean(x)| / std(x)$. The statistic is tested with significance levels to detect noisy instances.

Ye and Chen [71] calculate a chi-square statistic as follows: $\Sigma_i^n[(x_i - E_i)^2/E_i]$ where x_i is the value of the i^{th} feature, E_i is the expected value of i^{th} feature, and n is the number of features. If the chi-square statistic for an instance is greater than [E_i+3×(standard deviation)], it is detected as noise.

Aggarwal [2] calculates statistical deviations from the values predicted by a polynomial regression; a high statistical deviation indicates the possibility of a noisy instance.

Robert [53] proposes *extreme value theory* (EVT) for noise detection. EVT uses a *gaussian mixture model* (GMM), a popular method of data density estimation. The maximum

likelihood parameters of the GMM are estimated by an *expectation maximization* (EM) algorithm. Noise is defined as the extremal instances in the tail of the data distribution. Therefore instances that lie outside of the range of expected extreme values are identified as noise by EVT.

Filzmoser et al. [21] introduce a *sign noise detection method* that uses *Principal Component Analysis* (PCA) (refer to Chapter 3 for PCA) and the *Mahalanobis* distance (described below). The intuition is that noisy instances will be more outstanding in principal component space than original data space since they increase variance. The Sign method calculates robust Mahalanobis distances from sphere data normalized in principal component space. Then instances with robust distance values greater than a threshold calculated from a Chi-square distribution are identified as noise.

2. Distance-based Approaches

Distance-based approaches are among the most popular noise detection approaches. These approaches identify an instance as noise if the instance has no more neighbors than a fraction of the data within its neighborhood, using metrics such as Euclidean or Mahalanobis distance. The Mahalanobis distance is a distance function that considers the correlations between features when measuring the distance between an instance and the center of all instances: $Mahalnaobis = sqrt\ [(x-\mu)^t \Sigma^{-1} (x-\mu)]$ where μ is the center and Σ is the covariance matrix ([14] and [27]). These approaches are expensive computationally since the distances between all instances are measured.

Ramaswamy et al. [49] calculate the distance of an instance from its k^{th} nearest neighbor and rank each instance on the basis of the distance. While higher ranked instances are sparse, dense instances have lower ranks. Therefore, the top n instances are identified as noise.

Knorr and Ng [34] count the neighbors of each instance within some distance *d* and define as noise any instance that has less than or equal to *m* neighbors.

Breuning et al. [9] propose a method to detect *density-based local outliers*. They assign to each instance a degree of being an outlier. The local outlier factor (LOF) represents the degree of isolation of an instance from its neighbors and is calculated from the *k*-distance neighborhood (refer to their paper for the formal definition of LOF).

Aggarwal and Yu [1] provide an intuition for noise detection in high dimensional data. The method measures the sparsity of data points in the lower dimensions searched by genetic algorithms. It divides the data into *f* ranges and generates individuals which can contain the values 1 to *f*, or * (don't care). The evaluation function computes the sparsity coefficient of the data in the lower dimension with the corresponding grid ranges.

Li and Kitagawa [43] propose a distance-based noise detection method combined with an example-based algorithm for high dimensional data. User noisy instances are implanted in the high dimensional data, and feature spaces are searched by a genetic algorithm. The aim of the genetic algorithm is to find the most suitable feature space in which the user noise is significantly outstanding. The evaluation function of the genetic algorithm using a binary representation is related to the number of neighbors around normal and user noisy instances. After the most suitable features are found, other parameters are determined to separate user noise from normal instances. With the determined parameters, a regular distance-based noise detection method is then applied.

Brieman ([7] and [8]) introduces the *Random forests technique* that is a proximity-based noise detection method. The technique constructs many trees by randomly sampling instances with replacement from the training data set, as well as randomly selecting features. After each

tree is constructed, all instances in the data follow down the tree and proximities are calculated for all of the instances. The proximities are stored in an $N \times N$ matrix, and if two instances are in the same leaf of a tree, their proximity increases by one. At the end, proximities are normalized, dividing by the number of trees. In Random forests, noise is defined as instances with lower proximities to all other instances in the same class. Breiman defines a noise measure and then recommends that if the measure is greater than 10, the instance should be considered as noise. This method takes time to construct its many decision trees.

3. Cluster-based Approaches

In cluster-based approaches, noise is detected as a byproduct of the clustering process. These approaches assume that noisy instances do not belong to any cluster, belong to small clusters, or belong to clusters significantly different from others, as they are significantly different from other normal instances. Similar to distance-based approaches, cluster-based approaches are computationally expensive.

Ng and Han [46] develop a clustering method, CLARANS (Clustering Large Applications based on RANdomized Search) that is based on randomized search. After applying CLARANS, the instances in a cluster with silhouette widths below 0.5 are removed as noise. They note that the silhouette of an instance indicates how much the instance truly belongs to the cluster and an instance with a value close to 1 belongs to the cluster with high certainty. The silhouette width is the average of the silhouettes of all instances in the cluster.

Zhang et al. [74] propose BIRCH (Balance Iterative Reducing and Clustering using Hierarchies) which builds a clustering tree on the basis of their CF (clustering feature). A node in the CF tree represents a cluster made up of all its entries. While dense instances on the CF tree

are treated as a cluster, sparse instances are detected and removed as noise. BIRCH works incrementally, dynamically adding new incoming instances to CF tree.

Ester et al. [20] introduce the DBSCAN (Density Based Spatial Clustering of Applications with Noise) clustering algorithm. DBSCAN uses the density-based notions: *directly density-reachable*, *density-reachable*, and *density-connected* (refer to their paper for the formal definitions of these notions). A cluster is defined as a set of density-connected instances, and noise is identified as instances not belonging to any clusters.

He et al. [26] define a noise factor, CBLOF (Cluster-Based Local Outlier Factor) that represents the degree of deviation of an instance from a cluster. The CBLOF of an instance is determined by the size of its cluster and the distance between the instance and its *closest cluster*. Their method clusters instances with its *squeezer clustering algorithm* and calculates a CBLOF value for each instance.

4. Classifier-based Approaches

These approaches look into the characteristics of a machine learning classifier and focus on how it responds to noise. Therefore the performance and complexity highly depend on the accuracy of the adopted classifier on the data.

Torr and Murray [64] use a regression function for noise detection. This method analyzes the effect of an instance on regression. They propose to create the regression function, remove the instance that has the greatest effect on the regression function, and recalculate the regression function. This procedure is repeated until the effect is less than a threshold. However they note that using a robust regression method is critical when there are many noisy instances in the data.

Another classifier used frequently for noise detection is a neural network. Hawkins et al. [25] propose a *replicator neural networks* (RNN) method. RNN is a feed-forward multi-layer perceptron with an input, an output, and three hidden layers. The numbers of units in the hidden layers are decided empirically to minimize the average reconstruction error across all the training data. The trained RNN calculates the outlier factor (OF) of all instances, and higher valued instances are identified as noise. OF is defined by the average reconstruction error over all features. Williams [67] reports that RNN performs successfully for small and large data sets.

Vesanto et al. [66] use a self-organizing map (SOM) which is an unsupervised neural network to detect noise. SOM is trained iteratively on each instance of the data. It assigns an instance to its best matching units and updates the weight vector of the node, which is similar to the mean vector in the *k*-mean clustering algorithm. If the vector distance that is the distance between the instance and its best matching unit is large relative to the accuracy of the map unit, the instance is identified as noise. Also Saunders and Gero [55], and Ypma et al. [72] propose methods to use SOMs for noise detection.

Tax et al. [63] introduce support vector data description (SVDD), a modified SVM. An SVDD is the minimal volume sphere which contains all data within the normal class. SVDD is a more flexible and tighter description using Gaussian kernels than the normal spherical description of SVM. If an instance is not included in SVDD, it is identified as noise.

5. Misclassification-based Approaches

Misclassification-based approaches identify instances that are misclassified by a classifier as noise. Similarly to classifier-based approaches, the performance of these approaches highly depends on the accuracy of the classifier.

Brighton and Mellish [10] introduce *Wilson's editing method* and *Tomek's extension*. Wilson's editing method removes noisy instances that are incorrectly classified by their nearest neighbors. Tomek enhances Wilson's method by using a *k-NN* algorithm in which the value *k* is increased after each iteration, and repeating the editing rule until it is not applicable to any more instances.

John [32] proposes the *Robust-C4.5 algorithm*, which uses a pruning tree to remove noisy instances. The basic assumption of the algorithm is that the instances classified incorrectly by the pruning tree are not useful locally. Furthermore it assumes that locally useless instances are not useful globally as well. The method builds a pruning tree based on the data and classifies each instance in the data. The instances misclassified by the pruned tree are removed from the data. These processes are repeated until the pruned tree correctly classifies all instances in the data.

Brodley and Friedl [11] use *n*-fold cross-validation to identify misclassified instances. The data are partitioned into *n* subsets. For each of the *n* subsets, *m* classifiers are trained on the instances in the other *n-1* subsets and then used to classify the instances in the excluded subset. Each classifier tags the instance as misclassified if the instance is classified incorrectly. Majority voting or consensus can be used in the filtering process.

Muhlenbach et al. [45] provide a preliminary procedure in which misclassified instances are filtered. The algorithm creates a geometrical neighborhood graph of the data set and optionally removes or re-labels an instance if the proportion of instances with the same class is not significantly large in its neighborhood.

6. Prototype-based Approaches

Prototypes refer to small subsets of instances that represent a large original data set. Normally prototypes are used to reduce the memory space for large databases. Skalak [58] notes that prototypes tend not to include many noisy instances and the limited number of prototypes may avoid overfitting the training data. Therefore prototype-based approaches achieve the effect of noise removal on training data.

Skalak [58] uses Monte Carlo sampling and random mutation Hill Climbing to generate prototypes that exclude many noisy instances. The procedure by which Monte Carlo sampling creates prototypes is as follows. (1) Select k random samples of n instances with replacement from the training data. (2) Measure the classification accuracy for each sample. (3) Choose the set of instances with the highest classification accuracy as prototypes. The Hill Climbing method uses the following steps. (1) Choose a binary string representation of n prototypes at random. The length of the binary string is $\lceil \log_2{}^m \rceil \times n$ where m is the number of instances in data. (2) Mutate a bit at random. (3) Measure the fitness of the mutated string using a classifier. If the fitness is better, the binary string is replaced by the mutated string. (4) Repeat steps 2 and 3 for a maximum number of iterations. In both methods, the 1-nearest neighbor classifier is used.

In Skalak's algorithm described above, the number of prototypes (n) is fixed. Sebban [56] proposes a method to determine the number of prototypes by constructing homogenous subsets. After constructing the neighborhood graph of the minimum spanning tree, the homogenous subsets are constructed by deleting the edges connecting points that belong to different classes. Therefore the homogenous subset is a connected sub-graph in the minimum spanning tree. The number of prototypes is then set proportional to the number of homogenous subsets. Afterwards, Skalak's Monte Carlo sampling is applied to identify the prototypes.

Skalak [59] applies a genetic algorithm for prototype selection as well. Each individual in the population represents a set of n prototypes. As with the previous Hill Climbing method, the size of an individual is $\lceil \log_2{}^m \rceil \times n$ where m is the number of instances in the data. The fitness is the accuracy on the training data by a 1-nearest neighbor classifier.

Sierra et al. [57] use an *Estimation of Distribution Algorithm* (EDA). An EDA is similar to a genetic algorithm but does not have crossover and mutation operators. Instead it estimates the joint probability distribution of an individual among the individuals selected by a selection operator, and samples new individuals from this distribution to create the next population. Binary representation is used with a 1 indicating the selection of the corresponding instance.

7. Other Approaches

Crawford and Wainwright [17] apply genetic algorithms for noise detection. Each GA individual includes k noisy instances, given n data instances. This method sets the fixed size k of the individual by experimental testing from 2 up to $\lfloor n/2 \rfloor$ separately, and compares three different evaluation functions: Least Squares, Cook's squared distance formula, and the determinantal ratio developed by Andrews and Pregibon.

Xiong et al. [69] propose the Hcleaner technique, a hyper-clique based data cleaner. Every pair of instances in a hyper-clique pattern has a high level similarity related to the strength of the relationship between two instances. Hcleaner filters out instances that are not included in any hyper-clique pattern as noise.

Arning et al. [4] provide a set-based approach. The subset of data whose removal causes the greatest contribution in the dissimilarity of the remaining data with the least number of removed instances is considered as a noise set. The dissimilarity function can be any function

that returns a lower value between similar data and a higher value between dissimilar data such as variance. However, they note that finding a universal dissimilarity function is difficult.

Lane and Brodley [41] use similarity sequence matching to detect noise in string sequence data. The similarity of an instance to the training data is calculated. If the similarity measure is between minimum and maximum bounds, the instance is normal. Otherwise the instance is noisy. The bounds are determined empirically, and the similarity measure depends on the similarity function and an adjacency counter. Their similarity function returns a high value for pairs of closely resembling sequences and a low value for pairs of largely different sequences.

Noise Handling

There are two general treatments of noisy instances: noise removal and noise accommodation. The first solution, noise removal, detects and removes noisy instances to enhance the quality of data. Noisy instances detected by the methods in the previously described approaches can be removed in data preprocessing.

The alternative solution, accommodation accepts noisy instances and builds a robust model that withstands noise and minimizes its effects. *Bagging* and *stacking* achieve robustness by combining multiple models. While bagging combines models of the same type by voting, stacking uses a meta-learner to combine the classifications of models of different types. The Random forests method is an example of bagging in which many classification trees are constructed, each tree votes to classify a tested instance, and the majority class is assigned to the instance. Breiman [7] tries to prove the robustness of the Random forests method, comparing with Adaboost. Least median of squares regression is a robust version of least square regression.

Unlike least square regression that minimizes the sum of squared errors, it minimizes the median of the squared errors. Barnett [5] notes that the least median square regression is noise robust up to a degree of noise contamination as high as 50 percent.

CHAPTER 3

DIMENSIONALITY REDUCTION

The Curse of Dimensionality

A classifier that uses a limited number of instances in high dimensional space suffers from *the curse of dimensionality*. The curse of dimensionality is that the volume of a data space grows exponentially as the number of dimensions increases. Therefore many more instances are required in order to keep the same density of instances throughout the entire space. The problems of high dimensionality are described by the following two geometric properties of the curse of dimensionality ([6], [16] and [24]).

1. To grab a small fraction of the instances in a high dimensional data space, a large amount of space must be covered. Hastie notes that in ten dimensions, 63% or 80% of the space must be enclosed respectively in order to capture 1% or 10% of the instances on average. Therefore a classifier that uses lazy learning such as k-nearest neighbor may not be used in high dimensional space.
2. Most instances are concentrated in the boundary of high dimensional space. It was found that when 100 instances are uniformly distributed in a 10-dimensional unit ball centered at the origin, the median distance of the nearest instance to the center of the space is approximately 0.61 which means that the ratio of distances from center and to boundary is 61:39. Thus the instance is closer to the boundary than the center of the space. This indicates that new instances may be extrapolated as each instance is far from the other instances. Therefore prediction in high dimension becomes much more difficult.

To cure the curse of dimensionality and build an accurate model from high dimensional data, dimensionality reduction is necessary. There are two fundamental approaches to dimensionality reduction: *feature extraction* that generates new features of lower dimension from the original higher dimensional feature space and *feature subset selection* that chooses a relevant subset of features from the large number of original features.

Feature Extraction

For dimensionality reduction, feature extraction generates new features by mapping the original higher-dimensional feature space to a lower-dimensional feature space. While a linear method calculated by a linear combination of original features is simpler and easier, a non-linear method is more difficult but more general. The disadvantage of these approaches is that interpreting the meaning of the new features is usually difficult.

One of the most popular feature extraction methods is *Principal Component Analysis* (PCA) ([22], [37] and [61]). Principal components (PCs) are defined as orthogonal linear combinations of original features. PCA generates the linear combination with the largest variance as the first PC. The second PC has the second largest variance and is orthogonal to the first PC. The third PC has the third largest variance is orthogonal to both the first and the second PCs, and so on. In PCA, there are as many PCs as the number of original features and the dimensionality can be reduced by excluding the later PCs with smaller variance.

Similar to PCA, *Independent Component Analysis* (ICA) generates linear combinations ([28], [29], [37] and [62]). However, ICA is different from PCA in that it seeks independent components (ICs) using higher order statistics while PCA uses second order statistics to generate uncorrelated features. Independence always implies uncorrelatedness but the opposite is not

always true. Only if the distribution is multivariate normal, they are equivalent. Therefore for Gaussian distribution, PCs are also ICs. ICA does not necessarily reduce dimensionality. Some other methods such as PCA are needed first to reduce dimensionality. Forder [22] provides a survey of more feature extraction methods such as factor analysis, projection pursuit, non-linear principal analysis, non-linear independent analysis, and random projection.

Another feature extraction approach is feature weighting using genetic algorithms. Multiplying the original features with the associated weights generates new features. Jarmulak and Craw [30], Kelly and Davis [35], and Punch et al. [48] present genetic algorithms to determine feature weights for k-NN classifiers. The individuals are real valued and each position in the individual holds the weight associated with each feature. The evaluation function uses the accuracy of the weighted k-NN classifier. The weights close to zero result in dimensionality reduction by eliminating the corresponding features.

Feature Subset Selection

Feature subset selection is an approach to select a relevant subset of features from the original features of high dimensions. There are two broad categories of feature subset selection approaches: filter and wrapper approaches. The filter approach is a simple and fast method that optimizes feature selection using evaluation measures such as distance, information, dependence and consistency ([18], [38] and [44]). The filter approach is fast because it does not use any induction algorithms to evaluate a selected subset of features. Meanwhile, the wrapper approach uses induction algorithms to evaluate the selected subset of features. It is slower but more accurate.

1. Filter Approaches

Distance-based filter approaches are known as separability-based, divergence-based and discrimination-based [18]. They assume that instances belonging to the same class are close to each other and instances belonging to different classes are relatively far apart. A feature that induces a greater distance between instances from different classes is preferred for selection.

Relief for binary classification is a distance-based method ([36], [37] and [54]). Relief samples a set of instances randomly. For each selected instance, *Nearest Hit*, a nearest instance belonging to the same class and *Nearest Miss*, a nearest instance from the other class are found. For each feature, the feature weight is updated based on the differences between the instance and the Nearest Hit or the Nearest Miss. If the instance and the Nearest Hit are close, then the weight increases. Otherwise, it decreases. Also if the instance and the Nearest Miss are far, then the weight increases. Otherwise, it decreases. Features with weight greater than a threshold or a predefined number of features with higher weight are selected. Relief can be extended to multiple class problems as well as to regression problems.

Information-based filter approaches measure information gain of a feature. A feature that has high information gain is preferred for selection. Information gain is the expected reduction of impurity of instances separated by a feature. Information gain of a feature x is formally defined as follows:

$$Entropy\ (S) \equiv \Sigma_i -P_i \log_2 P_i,$$

$$Gain\ (S, x) \equiv Entropy\ (S) - \Sigma_{v\ \epsilon\ values(x)}\ (|S_v| / |S|) \times Entropy\ (S_v),$$

where S is a set of instances, P_i is the proportion of instances that belong to class i, $values(x)$ is the set of possible values for feature x, $|S_v|$ is the number of instances whose feature x has the value v, and $|S|$ is the number of instances.

Decision tree learning uses information gain to select the best features. Cardie [13] creates C4.5 decision tree and selects as relevant features the features remaining in the pruned tree. Also Singh and Provan [58] propose a feature selection method to find features that maximize the following information metrics using a forward greedy search (refer to their paper for the formal definitions of the metrics): conditional information gain (CIG), conditional gain ratio (CGR), and 1-conditional distance (CDC).

Dependence-based filter approaches measure the dependence between a feature and the class. A feature with a high dependence on the class is preferred for selection. While a feature with a low dependence on the class is irrelevant, a high dependence of a feature on other features is a measure of redundancy of the feature. Therefore lower inter-dependence between features is desired ([18]).

Correlation-based feature selection is a well-known dependence-based filter approach. Hall [23] applies a correlation metric to classification and regression problems, using best first search under the assumption that good features are highly correlated with the class but uncorrelated with each other. For regression problems, they apply linear correlation. For classification problems, symmetrical uncertainty (SU) is used to estimate the degree of association between features. Yu and Liu [73] also use SU to identify irrelevant and redundant features. The definition of symmetrical uncertainty is as follows:

$H(x) = -\Sigma_i P(x_i) log_2(P(x_i))$,

$H(x|y) = -\Sigma_j P(y_j) \Sigma_i P(x_i|y_i) log_2(P(x_i|y_i))$,

$SU(x,y) = 2 \times [(H(x)-H(x|y)) / (H(x)+H(y)]$, where $P(x_i)$ are the prior probabilities for all values of x, and $P(x_i|y_i)$ are the posterior probabilities of x given y.

*Consistency-based feature selection approach*es select a minimal set of features that separate classes most consistently. If instances have the same values for the selected features but different class label, the instances are inconsistent for the feature subset. For feature *x* in Table 3.1, the first and the second instances have the same value 0, but they are assigned the different target *t* values 0 and 1 respectively. Therefore the instances are inconsistent for feature *x*.

Table 3.1 Consistency of features

x	*y*	*T*
0	0	0
0	1	1
1	0	1
1	1	0

Almuallim and Dietterich [3] introduce the FOCUS algorithm that uses an exhaustive search to finds a minimal subset of features in which there are no two instances that are consistent in all the features but are not consistent in the class. Liu and Setiono [40] propose a probabilistic consistency-based filter method, LVF, which creates a random feature subset in each trial. If the number of features is less than the size of the best feature subset found so far and the inconsistency criterion is less than a predefined rate, the best feature subset is replaced with the current feature subset. The LVF repeats for a pre-determined number of trials. Dash and Liu [19] compare different inconsistency measures and different search strategies.

2. Wrapper Approaches

Wrapper approaches use induction algorithms to evaluate the selected subset of features. John et al. [31] search the feature space using greedy algorithms of backward elimination and forward selection. The selected subset of features is evaluated by decision trees using n-fold cross validation. Skalak [60] uses a random mutation hill climbing search algorithm, and the accuracy of a 1-nearst neighbor classifier as the evaluation function. Liu and Setiono [39] introduce a probabilistic wrapper approach LVW, a modified version of LVF. LVW generates a random subset of features and calculates the error rate using decision trees (C4.5 and ID3). If the error rate for current features is less than that for the best, the best feature subset is replaced with the current feature subset. LVW repeats this process until the error rate is not updated for a predefined number of times.

Another typical wrapper approach is to use genetic algorithms. Vafaie and De Jong [65], Jarmulak and Craw [30], and Yang [70] propose genetic algorithms for feature selection. They implement binary representation in which the value 1 corresponds to a relevant feature, and use evaluation functions related to classification accuracy. Pernkopf and O'Leary [47] present a genetic algorithm that uses an integer representation. In it, an individual has a predetermined size and is initialized with values between 1 and the number of original features. The evaluation function is based on the classification rate as well. Furthermore, there are many variations of genetic algorithms for feature selection. Cantú-Paz [12] presents a hybrid method which uses the output of a filter method to form the initial population, and uses classification accuracy as the evaluation function. Lanzi [42] applies a genetic algorithm to a filtering method using the inconsistency rate as the evaluation function. Cantú-Paz and Lanzi both use binary representation. Jourdan et al. [33] also use a genetic algorithm with binary representation for

feature selection. Their evaluation function is formularized with the total number of features and the number of selected significant features that are not too close in terms of the chromosomal distance.

Combination of Feature Extraction and Feature Subset Selection

Genetic algorithms can be used as a hybrid method for feature generation and feature selection. Raymer et al. ([50] and [51]) introduce a genetic algorithm to select the relevant features and to determine the weights of the selected features simultaneously. The individuals are composed of two parts: the feature weighting part and the feature selection part. The positions for feature weighting contain real valued numbers between 0 and 100 and while those for feature selection contain binary masking digits – one for each feature. If the weight is close to 0 or the masking digit is 0, then the feature is not considered by the classifier. The evaluation function uses a weighted k-NN accuracy based on the selected features. Ritthoff et al. [52] use variable-length individuals. These individuals hold the selected features (e.g. x or y from Table 3.1 above) and the additional features (e.g. $x \dotplus y$ or $x \times z$) generated by feature generators. The features are evaluated by a support vector machine using both the selected features and the generated features.

CHAPTER 4

THE PROPOSED ALGORITHM - NDFS

General Description of NDFS

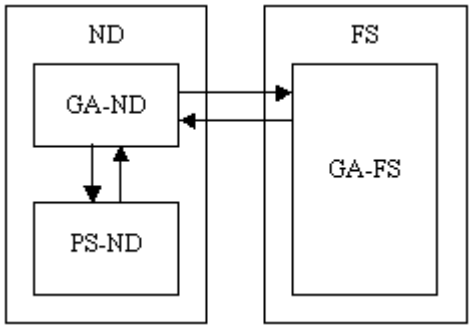

Figure 4.1 Framework of NDFS

As shown in Figure 4.1, the proposed algorithm (NDFS) is composed of two parts: a noise detection method (ND) and a feature selection (FS) which communicate to exchange their results. ND implements two sub-parts, a genetic algorithm (GA-ND) and prototype selection (PS-ND), to detect noise. FS also uses a genetic algorithm (GA-FS) to select the relevant features. GA-ND and GA-FS exchange their results at certain generation intervals. In the first step, GA-ND identifies a set of candidates suspected of being noisy instances. These suspicious instances are removed from the training data set as potentially noisy instances. Before GA-ND passes its result to GA-FS, PS-ND is triggered to detect actual noisy instances among the candidates. GA-FS then selects the relevant features based on the training data provided by ND, and returns these identified features to ND which in turn uses them for another round of noise detection. These steps are repeated until a stopping condition is satisfied.

Pseudocode for NDFS is provided in Figures 4.2 through 4.6, which describe common frameworks for classification and regression problems. Problem specific implementations are separately explained in the sections below. NDFS (Figure 4.2) includes two algorithms: ND (Figure 4.3) and FS (not given). Note that while ND has two subroutines, FS contains only one, GA-FS, and thus is simply a trigger for this subroutine.

Algorithm NDFS

```
INPUT : data
OUTPUT : noisy_instances, relevant_features

noisy_instances = null
relevant_features = original features of data

noisy_instances = ND (data, relevant_features)

for i = 0 to n
   relevant_features = FS (data, noisy_instances)
   noisy_instances = ND (data, relevant_features)
end for

return noisy_instances, relevant_features
```

Figure 4.2 Algorithm NDFS

ND first identifies potentially noisy instances using relevant features selected by FS, and then measures the actual noise of these candidates. These processes are performed by GA-ND in Figure 4.4 and PS-ND in Figure 4.5 respectively. Then ND updates the best individual in its GA-ND with the actual noise detected by PS-ND. FS identifies relevant features using instances identified as noise-free by ND. This process is performed by GA-FS (Figure 4.6). The next generations in GA-ND and GA-FS are probabilistically re-initialized with their current best individuals (the reason for this is discussed later). Note that GA-ND returns its noise candidates

after *n1* generation and GA-FS exchanges its relevant features every *n2* generation. Also NDFS repeats these steps *n* times (refer to Table 5.9 in the next chapter for all parameter settings).

Algorithm ND

INPUT : data, relevant_features
OUTPUT : noisy_instances

noise_candidates = GA-ND (data, relevant_features)
noisy_instances = PS-ND (data, noise_candidates, relevant_features)

update the best individual in GA-ND with noisy_instances

return noisy_instances

Figure 4.3 Algorithm ND

GA-ND: Identifying Candidates for Noise

GA-ND applies a genetic algorithm to identify noisy candidate instances using relevant features selected by FS.

1. Representations

GA-ND can be implemented using a binary or integer representation. For either, the size of individuals in the population is *I* which corresponds to the number of instances in the training data set. Each position in an individual corresponds to an instance in the training data and is used to indicate if an instance is noise-free or noisy. In the binary representation, a value of 0 at the i^{th} position of an individual indicates that the i^{th} instance in the training data set is noise-free while a 1 means the i^{th} instance is noisy.

In the integer representation, the size of each individual is *(I+1)*. Each individual has an additional position that contains a threshold. If the value at the i^{th} position of an individual is greater than or equal to this threshold, the i^{th} instance in the training data set is considered noise-free; otherwise, the i^{th} instance is noisy.

Algorithm GA-ND

INPUT : data, relevant_features
OUTPUT : noise_candidates

```
create training_data by choosing only relevant_features from data

re-initialize population with the best individual in the previous population
calcuate the fitness values using training_data

for i = 0 to nI
   do selection operation
   do crossover operation
   do mutation operation
   calcuate the fitness values using training_data
   update population
end for

create noise_candidates with instances associated to the best individual
return noise_candidates
```

Figure 4.4 Algorithm GA-ND

2. Populations

For binary representation, each position in an individual of GA-ND is initialized with the value 0 with probability *p1* or the value 1 with probability *p2*. For integer representation we initialize each position of an individual with a uniform random integer value between 0 and *(I-1)*. The size of populations for both representations is set to *N1*.

3. Evaluation (fitness) Functions

GA-ND uses only the features selected by the FS algorithm to calculate the fitness values. The fitness value of each individual is calculated on the basis of the classification accuracy of the primary classifier (we call the classifier in GA-ND and GA-FS the primary classifier). Individuals in GA-ND split the training data set into two subsets: the remaining data set and the removed data set. The remaining data set is composed of the instances considered as noise-free by the individual, and the removed data set is created by grouping the instances identified as potential noise. Note that the removed data set is denoted by *noise_candidates* in GA-ND of Figure 4.4.

GA-ND maximizes the following evaluation function:

ND-Fitness = %RemainingCorrect + %RemovedIncorrect – TreeSize,

where *%RemainingCorrect* is the percentage of instances classified correctly by the primary classifier in *f1*-fold cross-validation on the remaining data set, *%RemovedIncorrect* is the percentage of the instances in the removed data set classified incorrectly by the primary classifier trained on the remaining data set, and *TreeSize* is the size of the tree built by the primary classifier on the remaining data set. In the case where all instances in a small removed data set are classified incorrectly and *ND-Fitness* is very high, without removing more instances from the training data set, the genetic algorithm may become stuck in a local optimum. In this event *TreeSize* provides the momentum the genetic algorithm needs to move forward and reach the global optimum. If more instances are removed from the training data set, the size of the tree built on the remaining data set becomes smaller and *ND-Fitness* gets higher. In addition, a smaller tree may prevent overfitting noisy instances.

4. Selection and Variation Operators

In both representations, we use tournament selection with replacement and tournament size *k*. We use *r*-point crossover with probability *p5*. For binary representation we apply bitwise mutation to each individual with probability *p6*. Mutation is done by flipping the bit at one randomly selected position. For integer representation, we apply creep mutation to each individual with probability *p7*. Mutation is done by adding a creep constant *a1* with probability *p8* or a creep constant *a2* with probability *p9* to the value at each position in the individual.

5. Consideration of Multiple Individuals in Mature Population

It is observed that under the evaluation function of GA-ND, several individuals may get the same fitness value. This phenomenon can cause some of the best individuals to fail to converge to the global optimum. Thus, we consider the *b* individuals with the highest fitness values instead of picking only the best individual in the population in the final generation. If a position in *b* individuals has been selected more than B×*b* times, the corresponding instance is added to the set of candidates for noisy instances. Therefore the set of candidates is likely to include more elements than the set of actual noisy instances. Finally GA-ND outputs its set of candidates for noisy instances based on *b* individuals in the population in the last generation. Note that this process is performed only once in the final generation of GA-ND where the population is completely mature.

PS-ND: Detecting Actual Noise

We observe that GA-ND tends to remove many false noisy instances from the training data set. Therefore PS-ND is used to attempt to recover the incorrectly removed noisy instances.

As mentioned above, the word “prototypes” refers to small subsets of instances that represent a large original dataset. Normally prototypes are used to reduce the memory space for large databases. Skalak [60] notes that prototypes tend not to include many noisy instances and the limited number of prototypes may avoid overfitting the training data.

Algorithm PS-ND

```
INPUT : data, noise_candidates, relevant_features
OUTPUT : noisy_instances

create training_data removing noise_candidates and selecting relevant_features
    from data

for i = 0 to d
   for j = 0 to t
      select m percent of instances from training_data
      train a classifier i on the selected instances
      classify training_data
      measure accuracy
   end for
end for

prototypes = selected instances with highest accuracy
prototype_classifier = classifier that generated prototypes

for i = 0 to number of noise_candidates
  train prototype_classifier
  classify instance i in noise_candidates
   if (instance i is classified correctly)
      remove it from noise_candidates
end for

noisy_instances = noise_candidates
return noisy_instances
```

Figure 4.5 Algorithm PS-ND

Although potentially noisy instances are removed from the training data set by GA-ND, the remaining data set still has a chance to contain noisy instances. We therefore apply the prototype selection method to create a noise-free representation of the remaining training data. Note that the remaining data are denoted as *training_data* and the removed data as *noise_candidates* in PS-ND of Figure 4.5.

We follow similar steps to [60] to create prototypes that represent the remaining data set without noise:

1. Select *m* percent of the instances at random from the remaining data set.
2. Build a model using the instance set randomly selected by the secondary classifier (we call the classifier in PS-ND the secondary classifier).
3. Classify the remaining data set using the model.
4. Measure the classification accuracy.
5. Repeat steps (1 to 4) *t* times.
6. Choose the set of instances with the highest classification accuracy as prototypes.

The steps above are performed by *d* different secondary classifiers separately and the prototypes with the highest accuracy are selected. Also the secondary classifier that generates prototypes is referred to as the prototype classifier. Note that the prototype classifier is denoted as *prototype_classifier* in PS-ND of Figure 4.5.

We build a model on the prototypes created by the prototype classifier and classify the instances in the set of noise candidates using this model. Then the correctly classified instances are removed from the set of candidates and returned to the remaining data set, assuming that GA-ND has identified them incorrectly as candidates. The instances which remain in the set of

candidates until the end are detected as actual noisy instances, and the instances in the remaining data set which are considered noise-free are used by GA-FS later.

GA-FS: Identifying Relevant Features

GA-FS is also implemented with a genetic algorithm that identifies relevant features using instances selected as noise-free by ND.

1. Representations

Similarly to GA-ND, GA-FS can use binary or integer encoding as representations. The size of the individuals is *F* that is the number of features (attributes), and each position in an individual is associated to the number of a feature in the training data set. For binary representation, the value 0 at each position represents selection of the feature and the value 1 indicates elimination of the feature from the training data set. For integer representation, the size of individuals is *(F+1)*. If a value at the j^{th} position of the individual is greater than or equal to the threshold, the j^{th} feature in the training data set is considered as relevant; otherwise, the j^{th} feature is irrelevant or redundant.

2. Populations

For binary representation, individuals are initialized with the value 0 or 1 with probability *p3* or *p4* respectively. For integer representation, we initialize each position of an individual with a random integer value between 0 and *(F-1)*. The size of populations for both representations is set to *N2*.

Algorithm GA-FS

INPUT : data, noisy_instances
OUTPUT : relevant_features

```
create training_data by removing noisy_instances from data

re-initialize population with the best individual in the previous population
calcuate the fitness values using training_data

for i = 0 to n2
   do selection operation
   do crossover operation
   do mutation operation
   calcuate the fitness values using training_data
   update population
end for

create relevant_features with features associated to the best individual
return relevant_features
```

Figure 4.6 Algorithm GA-FS

3. Evaluation Functions

GA-FS uses only the instances selected as noise-free by the ND algorithm. GA-FS maximizes the following evaluation function:

FS-Fitness = %RemainingCorrect + %RemainingMeritCorrelation,

where the *%RemainingCorrect* is the percentage of the instances classified correctly by the primary classifier in *f2*-fold cross-validation on the training data set using only the features selected by the individual. The *%RemainingMeritCorrelation* indicates the merit percentage of correlation from the selected features which is used as the evaluation measure in the correlation-based feature selection (filter) method. The features selected as relevant by GA-FS are sent (back) to GA-ND.

4. Selection and Variation Operators

For the genetic operators of selection, crossover, and mutation, the same schemes as those for GA-ND are applied. However, unlike GA-ND, elitism is used to retain the best relevant features through each generation.

Re-initialization of Populations

The populations of GA-ND and GA-FS are re-initialized whenever they exchange their results. The reason for this is as follows. GA-ND uses the new features which GA-FS has optimized on the basis of the remaining data set created from the best individual in the previous generation of GA-ND. Therefore individuals other than the previous best one in the current generation of GA-ND are not related to the new features. So the population of GA-ND is re-initialized whenever it receives new features from GA-FS. The population of GA-FS is also reinitialized for similar reasons. These processes are performed by GA-ND (Figure 4.4) and GA-FS (Figure 4.6) respectively.

Re-initialization is implemented probabilistically. Each individual in the binary population is reinitialized with probability *p10* with the values of the previous best individual. For integer representation, individuals are reinitialized with probability *p11* with values computed by adding a creep constant *a1* to the value at each position of the previous best individual.

Implantation of Features Pre-selected by a Filter Method into Initial Population

According to NDFS in Figure 4.2, ND is initially triggered using all features of the original data and GA-ND performs noise detection initially on the basis of the set of full features.

If GA-ND were to use features pre-selected by a filter method in the first generation, it may achieve better performance. The process could be implemented by implanting an individual associated with features pre-selected by a filter method into the initial population for GA-FS, and initially passing these features to ND so that GA-ND uses them in the first generation.

This process is different from Cantu-Paz's method [12] in that while it initializes the whole population with the output of a filter method, NDFS implants one individual associated with features selected by a filter method into the initial population. If the implanted individual does not have a high fitness value, GA-FS is likely to throw it away after the first generation.

Extension to Regression Problems

For GA-ND in regression problems, the concept of incorrectly classified instance is defined as follows. If the absolute difference between the predicted value and the original target value is greater than *IncorrectThresholds*, then we consider the instance to be classified incorrectly. Otherwise, the instance is correct. The *IncorrectThresholds* is defined as follows:

$IncorrectThresholds = v \times Std$,

where v is a threshold constant and *Std* is the standard deviation of the target values in the original training data set. A low value v results in aggressively removing instances as noise, and so GA-ND may suffer from false noise detection. In the meantime, GA-ND with a high value v may not detect many true noisy instances.

To determine the threshold constant v, we introduce an *adaptive threshold scheme*. Initially the original training data set is classified and its relative absolute error (*error*) is measured, using *f3*-fold cross-validation. This step is repeated e times and the average *error* is calculated. Then v is set to *error* and the value of $[error \times Std]$ is assigned to *IncorrectThresholds*.

For example, if the relative absolute error is 25%, *IncorrectThresholds* is *[0.25× Std]*. We also empirically limit the minimum value of *v* to *C*.

However, there are two possibilities for data sets with high error rate. The first case is that although the error rate is initially high since the data is contaminated by high levels of noise, the classifier can build a more accurate model as noise decreases. The second case is that regardless of noise level, the classifier is inaccurate on the data set. In the former case, we need to decrease the threshold constant as noise is removed in order to detect as many true noisy instances as possible. However in the latter case, we want to increase the threshold constant in order to reduce the rate of false noise detection. To distinguish between these two cases, we look into the relationship between the primary classifier in GA-ND and the secondary classifier in PS-ND. In the first case, PS-ND agrees to the noise removal by GA-ND. However in the latter case, PS-ND returns many candidate instances to the remaining data set, considering them as false noise.

Therefore after *g1* and *g2* generations, the threshold constant *v* is updated by GA-ND. If PS-ND disagrees with candidates at a rate greater than *r1*, *IncorrectThresholds* is set to *[(v+0.1) × Std]*. If the recovery rate of PS-ND is below *r2*, the value of *[(v-0.1) × Std]* is assigned to *IncorrectThresholds*. Otherwise *IncorrectThresholds* is not changed.

For classification problems, *TreeSize* is included in the evaluation function of GA-ND. For regression problems, it may disturb the evaluation function since the threshold value also affects the evaluation function. Therefore we implement two different evaluation functions separately and compare their results. One is the same evaluation function as that of classification problems and the other is the evaluation function without the term *TreeSize*.

We do not consider multiple individuals in the final generation of GA-ND in regression problems. Only the best individual is selected to provide the noisy candidates. For the *%RemainingCorrect* in evaluation function of GA-FS, the value of 100 minus the relative absolute error percentage was used. Except for these differences, NDFS uses the same algorithm as that used in classification problems.

CHAPTER 5

DATA SETS AND EXPERIMENTAL SETUP

Data Sets

1. Classification Problems

We downloaded four real world data sets from the UCI machine learning repository and then generated synthetic data sets artificially from them. The downloaded data sets were Quadruped Mammals (Mammals), MUSK (Musk), Waveform (Waveform) and Wisconsin Diagnostic Breast Cancer (Wdbc). They vary in the number of features and class labels. Also they contain numeric values for most features. Therefore we can generate our synthetic data sets based on them with ease.

Table 5.1 shows the summary characteristics of the data sets. For example the Wdbc data set has 569 instances and 32 features. The class variable has two possible values as the data set includes 357 benign instances and 212 malignant instances.

Table 5.1 Characteristics of data sets

Data Sets	Number of Instances	N of Features	Number of Class Labels
Mammals	1000 (241, 250, 257, 252)	72	4 (giraffe, dog, cat, horse)
Musk	6598 (1017, 5581)	168	2 (musk, non-musk)
Waveform	5000 (1657, 1647, 1696)	21	3 (0, 1, 2)
Wdbc	569 (357, 212)	32	2 (benign, malignant)

If the original labels of the data are used, we cannot measure the performance of our algorithm since the testing data sets could be contaminated with noise. Therefore to generate synthetic data sets, we removed non-numeric features from the data sets. As a result, one feature from Wdbc and two features from Musk were eliminated. Then we generated synthetic data sets from the new data sets which include only numeric features. First we selected ten features at random and generated real numbers from the selected features using the formulas in Table 5.2. X_i refers to the value of the i^{th} feature. Each formula contains ten X variables since ten features were selected randomly. Note that all the coefficients and powers in these formulas were generated at random.

Table 5.2 Formulas to create synthetic data sets 1 for classification problems

Data Sets	Formula
Mammals 1	$- 0.68 - 0.60\ X_{10} + 0.29\ X_{11} + 0.85\ X_{25}^2 - 0.72\ X_{34}^2 + 0.88\ X_{43} + 0.56\ X_{45}^2 - 0.47\ X_{46}^2 - 0.06\ X_{52} + 0.03\ X_{66}^2 + 0.52\ X_{70}$
Musk 1	$0.63 - 0.41\ X_{14} + 0.11\ X_{37}^2 + 0.43\ X_{41} + 0.72\ X_{70} - 0.18\ X_{81}^2 - 0.08\ X_{94}^2 - 0.23\ X_{109}^2 + 0.60\ X_{110} + 0.96\ X_{149}^2 - 0.76\ X_{164}$
Waveform 1	$- 0.08 - 0.88\ X_2 + 0.79\ X_4^2 - 0.99\ X_6^2 + 0.64\ X_9 - 0.96\ X_{10} + 0.02\ X_{11} + 0.47\ X_{13}^2 - 0.90\ X_{16} - 0.04 X_{17}^2 - 0.95\ X_{20}^2$
Wdbc 1	$0.306 + 0.45\ X_4^2 - 0.32\ X_8^2 + 0.64\ X_{12}^2 - 0.92\ X_{16}^2 + 0.32\ X_{19}^2 + 0.21\ X_{23} + 0.23\ X_{27}^2 + 0.51\ X_{28}^2 - 0.64\ X_{29} + 0.30\ X_{30}^2$

We then set thresholds for the real numbers based on the class distribution of the original UCI data sets and labeled the class values by using the threshold. For example, for the Waveform data set, the 1657 instances with the highest values were assigned to *class 0*, the next highest 1647 instances were labeled *class 1*, and the 1696 instances with the lowest values were labeled *class 2*.

We then grabbed 10 noise-free training data sets with 100 instances each and the corresponding ten noise-free testing data sets with 100 instances each from the synthetic data sets at random. None of the noise-free training data sets or their corresponding noise-free testing data sets overlapped. Thus we had 10 noise-free training data sets of size 100 each and 10 corresponding testing data sets of size 100 each.

For noise levels of 5%, 10%, 15%, 20%, 25%, and 30%, we generated noisy training data sets from the noise-free training data sets created in the previous step. For each noise level, random instances, as many as the noise level implies, were selected and assigned to one of the other (wrong) class labels. For example, in order to generate the 5% noisy data sets, we selected 5 instances from each noise-free training data set and labeled them with one of the other class labels (randomly in the multi class domains). We generated 10 noisy training data sets for each noise level. Therefore, we had 10 noise-free training data sets, 10 corresponding noise-free testing data sets, and 10 noisy training data sets for each of the noise levels stated above.

To simulate various real world data sets, we generated another synthetic data set from each data set, following the same processes explained above but using different randomly generated formulas. Table 5.3 shows the formulas that were used to generate these additional synthetic data sets.

Finally, two different synthetic data sets were generated from each downloaded data set for classification problems. For each synthetic data set, we had 10 noise-free training data sets, 10 corresponding noise-free testing data sets, and 10 noisy training data sets for each of the noise levels stated above.

Table 5.3 Formulas to create synthetic data sets 2 for classification problems

Data Sets	Formula
Mammals 2	$0.34 + 0.79\ X_0^2 - 0.59\ X_4^2 - 0.78\ X_{16} - 0.63\ X_{20}^2 - 0.59\ X_{24} + 0.94\ X_{38}^2 + 0.04\ X_{40} + 0.65\ X_{48}^2 + 0.09\ X_{54} - 0.66\ X_{60}$
Musk 2	$- 0.83 - 0.49\ X_7 - 0.40\ X_{30}^2 - 0.78\ X_{40} + 0.56\ X_{49}^2 + 0.95\ X_{77} - 0.77\ X_{110}^2 + 0.18\ X_{125} - 0.58\ X_{140} + 0.78\ X_{148} - 0.84\ X_{154}$
Waveform 2	$- 0.14 + 0.70\ X_0^2 + 0.82\ X_2^2 - 0.26\ X_5 + 0.19\ X_7^2 + 0.26\ X_{10} + 0.59\ X_{11} - 0.77\ X_{12}^2 - 0.97\ X_{13}^2 + 0.49\ X_{16}^2 + 0.70\ X_{20}$
Wdbc 2	$0.86 - 0.63\ X_0^2 - 0.93\ X_1 + 0.67\ X_2 + 0.80\ X_{10}^2 - 0.07\ X_{11} + 0.29\ X_{14} + 0.78\ X_{17} + 0.57\ X_{18}^2 + 0.64\ X_{23}^2 - 0.16\ X_{27}$

Table 5.4 shows the classification accuracies after the primary classifier built a model using the training data sets created by the methods explained above and classified the noise-free testing data sets. The number in each cell of the table is the average of the results on 80 data sets ($80 = 4 \times 2 \times 10$; 4 original data sets, 2 different formula and 10 noisy data sets at each noise level). “Feature-selected” in the “Feature section” column means that irrelevant and redundant features were manually removed completely, and so there exists only ten relevant features in the data

sets. “Noise-removed” in the “Noise removal” column denotes that noisy instances were manually removed completely. The “Original and Noise-removed” data row shows results for the high dimensional training data whose noisy instances were manually removed completely. The “Feature-selected and Original” data row shows results for the noisy training data whose irrelevant and redundant features were manually removed completely. The “Feature-selected and Noise-removed” data row shows results for the training data with both noisy instances and irrelevant and redundant features manually removed completely. The “Feature-selected and Noise-free” data row shows results for the training data in which all instances had correct class values and all features were only relevant features. The main difference between the last two types was size. The number of instances in the “Feature-selected and Noise-removed” data row is smaller.

Table 5.4 Classification accuracies on synthetic data sets for classification problems

Feature selection	Noise removal	5%	10%	15%	20%	25%	30%
Original	Original	86.37	82.85	79.05	74.38	70.88	64.90
	Noise-removed	90.27	90.15	89.68	89.07	89.02	88.35
	Noise-free	90.32					
Feature-selected	Original	89.57	86.55	84.62	81.76	78.91	75.03
	Noise-removed	91.46	91.20	91.00	90.88	90.30	90.26
	Noise-free	91.61					

The table indicates that the primary classifier shows the highest overall accuracies for the feature-selected noise-free data sets and very close accuracies for the feature-selected noise-

removed data. However the classifier has poor accuracies for the original high dimensional noisy data. This result supports our hypothesis that removing noisy instances and selecting relevant features enhance the quality of the training data sets and improves classification accuracy. For example, the primary classifier shows a classification accuracy of 91.61% on the “Feature-selected and Noise-free” data and 90.26% on the “Feature-selected and Noise-removed” data at the noise level of 30%. However, it has an accuracy of only 64.90% on the “Original noisy” data at the same noise level.

2. Regression Problems

For regression problems, we used the same data sets as those for the classification problems. We generated synthetic data sets which included only numeric features, again as we did for the classification problems. We followed the formula of Table 5.5 to create the target values for the data set. Then we randomly grabbed ten noise-free training data sets with 100 instances each and their corresponding ten noise-free testing data sets with 100 instances each from the synthetic data set.

For noise levels of 5%, 10%, 15%, 20%, 25% and 30%, we generated noisy training data sets from the noise-free training data sets. For each noise level, random instances, as many as the noise level implies, were selected and noise values were added to the target values of the selected instances. The noise values were random numbers generated between 10% and 30% of the range of the target values in the training data set. Each noise value had a positive or negative sign randomly assigned. Therefore, we had 10 noise-free training data sets, 10 corresponding noise-free testing data sets, and 10 noisy training data sets for each of the noise levels stated above.

Table 5.5 Formulas to create synthetic data 1 for regression problems

Data Sets	Formula
Mammals 1	$-0.12 + 0.38\ X_{10}^2 - 0.77\ X_{12}^2 + 0.64\ X_{15}^2 - 0.66\ X_{16} - 0.13\ X_{32}^2 - 0.58\ X_{33}^2 - 0.15\ X_{40}^2 - 0.24\ X_{48} + 0.12\ X_{57}^2 + 0.01\ X_{62}$
Musk 1	$0.29 + 0.72\ X_{57} + 0.34\ X_{68} - 0.10\ X_{75} + 0.01\ X_{92}^2 - 0.18\ X_{104}^2 - 0.48\ X_{108}^2 - 0.25\ X_{109} + 0.80\ X_{112} - 0.70\ X_{157}^2 - 0.92\ X_{158}^2$
Waveform 1	$-0.55 - 0.92\ X_0^2 + 0.93\ X_2 + 0.63\ X_5 - 0.38\ X_6 + 0.21\ X_{10} - 0.50\ X_{11} + 0.85\ X_{15}^2 + 0.75\ X_{16} + 0.30\ X_{19}^2 - 0.61\ X_{20}$
Wdbc 1	$0.45 + 0.31\ X_1 + 0.53\ X_4 + 0.03\ X_6^2 - 0.44\ X_8 - 0.63\ X_{12}^2 + 0.18\ X_{13} + 0.08\ X_{19} + 0.01\ X_{24} + 0.60\ X_{26}^2 - 0.11\ X_{29}$

As in classification problems, we generated additional synthetic data sets from each data set, following the process explained above but using different formulas. Table 5.6 shows the formulas that are used to generate these additional data sets.

Finally, two different synthetic data sets were generated from each downloaded data set for regression problems. For each synthetic data set, we had 10 noise-free training data sets, 10 corresponding noise-free testing data sets, and 10 noisy training data sets for each of the noise levels stated above.

Table 5.7 shows the classification accuracies after the primary classifier built a model using the training data sets that were created by the methods explained above and classified the noise free testing data sets. The table's layout and description are similar to that for the classification problems.

Table 5.6 Formulas to create synthetic data 2 for regression problems

Data Sets	Formula
Mammals 2	$- 0.14 + 0.96\ X_0 - 0.62\ X_{10} - 0.41\ X_{16}^2 - 0.77\ X_{17} + 0.44\ X_{42} + 0.33\ X_{47} + 0.01\ X_{55}^2 - 0.59\ X_{58}^2 + 0.63\ X_{61} - 0.68\ X_{66}^2$
Musk 2	$- 0.96 - 0.18\ X_0 - 0.70\ X_{37} - 0.49\ X_{45}^2 + 0.51\ X_{65} - 0.35\ X_{68}^2 + 0.61\ X_{80} - 0.86\ X_{102} + 0.07\ X_{140} + 0.71\ X_{155}^2 - 0.80\ X_{163}$
Waveform 2	$0.15 + 0.002\ X_2^2 + 0.22\ X_3^2 + 0.79\ X_6 - 0.98\ X_7^2 - 0.48\ X_9 - 0.11\ X_{11}^2 + 0.15\ X_{12}^2 + 0.82\ X_{16} + 0.45\ X_{17}^2 + 0.18\ X_{18}$
Wdbc 2	$0.26 + 0.98\ X_2 + 0.98\ X_3 - 0.45\ X_5^2 + 0.17\ X_6^2 + 0.81\ X_{17}^2 - 0.03\ X_{23}^2 - 0.98\ X_{24} - 0.94\ X_{26}^2 + 0.58\ X_{28}^2 - 0.20\ X_{29}$

Table 5.7 Classification accuracies on synthetic data sets for regression problems

Feature selection	Noise removal	5%	10%	15%	20%	25%	30%
Original	Original	81.26	78.77	76.63	74.90	72.17	69.89
	Noise-removed	84.45	84.02	83.09	82.29	82.56	81.70
	Noise-free	84.22					
Feature-selected	Original	84.28	82.16	80.17	78.82	77.68	76.43
	Noise-removed	87.06	86.93	86.85	86.32	85.76	85.29
	Noise-free	87.21					

Performance Measures

1. Classification Accuracy

The goal of this research is to enhance the quality of the training data by removing noisy instances and selecting relevant features from high dimensional noisy training data. Therefore the classification accuracy is the main measure of performance. For classification problems, the classification accuracy is the percentage of test instances classified correctly by the model built on the training data set. For regression problems, the classification accuracy is 100 minus the relative absolute error estimated by the model built on the training data set. Also the average error reduction percentages over all noise levels are computed from the classification accuracies.

To evaluate the performance of NDFS, we chose two noise detection techniques and two feature selection methods as our opponents for each problem. For noise detection, Weka's noise filtering method with Random forests and Sign noise detection techniques were selected. The "RemoveMisclassified" filtering of Weka removes incorrectly classified instances, which is used for noise detection in Weka [68]. A C4.5 tree and a Model tree were selected as its classifier for classification and regression problems respectively. Also Random forests noise detection was applied to classification problems, while Sign method was applied to regression problems (refer to Chapter 2 for detailed descriptions of these algorithms). Both of Random forests and Sign methods were operated in the statistical software R. For feature selection, *correlation-based* and *ReliefF* filter methods are selected for both the classification and regression problems (refer to Chapter 3 for detailed descriptions of these algorithms).

2. Noise Detection Accuracy

Table 5.8 presents the confusion matrix as applied to noise detection. A confusion matrix is a good tool for evaluating correctness [68]. While true noisy and true noise-free instances are instances which get classified correctly, false noisy and false noise-free instance get classified incorrectly. False noisy instances are instances that get classified as noisy but are actually noise-free, and false noise-free instances are instances that get classified as noise-free but are actually noisy.

We derive the noise detection accuracy from the confusion matrix as follows :

Noise detection accuracy = (true noisy + true noise-free) /

(true noisy + false noisy + true noise-free + false noise-free)

Table 5.8 Confusion matrix for noise detection

Confusion Matrix		Instance Classification	
		Noisy	Noise-free
Actual Instance	Noisy	True noisy	False noise-free
	Noise-free	False noisy	True noise-free

3. Run Time

Since the drawback of genetic algorithms is speed, describing the run time of NDFS as it uses genetic algorithms is meaningful. We show the average run times of NDFS algorithms.

Experimental Setup

Table 5.9 shows the parameter settings used to implement the NDFS algorithm. For Weka, we used its default setup, and a threshold of [0.5×standard deviation of original target values] for regression problems. For ReliefF feature selection, the top ten ranked features were selected since the correct number of features is ten in our synthetic data sets. Also for Random forests, the instances with the noise measure greater than 10 were selected as noise.

Table 5.9 Parameter setting

Parameters	Classification	Regression
Primary classifier	C4.5 tree	Model tree
Secondary classifier	SVM, multilayer perceptron	
B	0.5	-
C	-	0.2
F	Feature size of each data set	
I	100	
N1	500	200
N2	100	80
a1	±1 (select + or – at random)	
a2	±10 (select + or – at random)	
b	100	-
d	2	
e	-	5
f1	if (generation < 80), f1=2, else f1=5	f1=2
f2	5	
f3	-	10
g1, g2	-	g1=40, g2=60
k	2	
m	10	
n, n1, n2	10	
p1, p2	p1=0.95, p2=0.05	
p3, p4	p3=0.5, p4=0.5	p3=0.02, p4=0.98
p5	p5=0.8	
p6, p7, p8, p9	p6=0.6, p7=0.6, p8=0.95, p9=0.05	
p10, p11	(current generation/ number of generations)	
r	2	
r1, r2	-	r1=0.25, r2=0.1
t	if (last generation) t=500, else t=200	if (last generation) t=200, else t=100

CHAPTER 6

EXPERIMENTAL RESULTS

Classification Problems

Tables 6.1 and 6.2 present the classification accuracies after the primary classifier built a model based on the training data sets and classified the noise-free testing data sets for classification problems. The numbers in each cell of the tables are the averages of the results on 80 data sets at each noise level (80 = 4×2×10; 4 original data sets, 2 different formula and 10 noisy data sets at each noise level; refer to Chapter 5 for the number of data sets) and the numbers in parentheses indicate the standard deviation of the results. Also the tables show the average error reduction percentages over all noise levels. Figures 6.1 and 6.2 compare the performances of noise detection techniques and feature selection methods on the behalf of the error reductions graphically.

1. Effect of Noise Removal Only and Features Selection Only

In Table 6.1, the accuracies and the average error reduction results from applying separately the noise detection techniques and feature selection methods are shown. In the table, “Weka” and “Random forests” data sets are the training data sets that Weka and Random forests offer respectively, after applying their noise detection techniques to the original high dimensional noisy training data sets. “Correlation” and “ReliefF” data sets are the training data sets whose relevant features are selected by correlation-based and ReliefF feature selection methods respectively. As seen in the table, the primary classifier shows the accuracies of 66.03%

on the data sets for Weka, 64.70% on data sets for Random forests, 75.81% for correlation-based feature selected data sets, and 70.48% for ReliefF feature selected data sets at the level of 30% noise.

Table 6.1 Effects of noise removal only and feature selection only in classification problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Weka	87.10 (4.24)	83.35 (5.78)	79.48 (5.98)	75.17 (7.16)	70.85 (6.65)	66.03 (8.21)	2.51
Random forests	85.82 (4.19)	82.75 (5.60)	79.01 (6.18)	74.03 (6.56)	71.17 (7.21)	64.70 (8.10)	-0.67
Correlation	90.28 (3.04)	87.26 (4.70)	85.48 (5.59)	81.92 (6.18)	80.77 (6.26)	75.81 (10.3)	30.45
ReliefF	85.77 (5.78)	82.01 (6.92)	81.80 (8.31)	80.31 (7.39)	77.18 (9.03)	70.48 (11.0)	13.51

According to Table 6.1 and Figure 6.1, feature-selection-only methods work unexpectedly well on our synthetic data sets for classification problems. Correlation-based feature selection reduces the error rates by 30.45% and ReliefF results in an average error reduction by 13.51%.

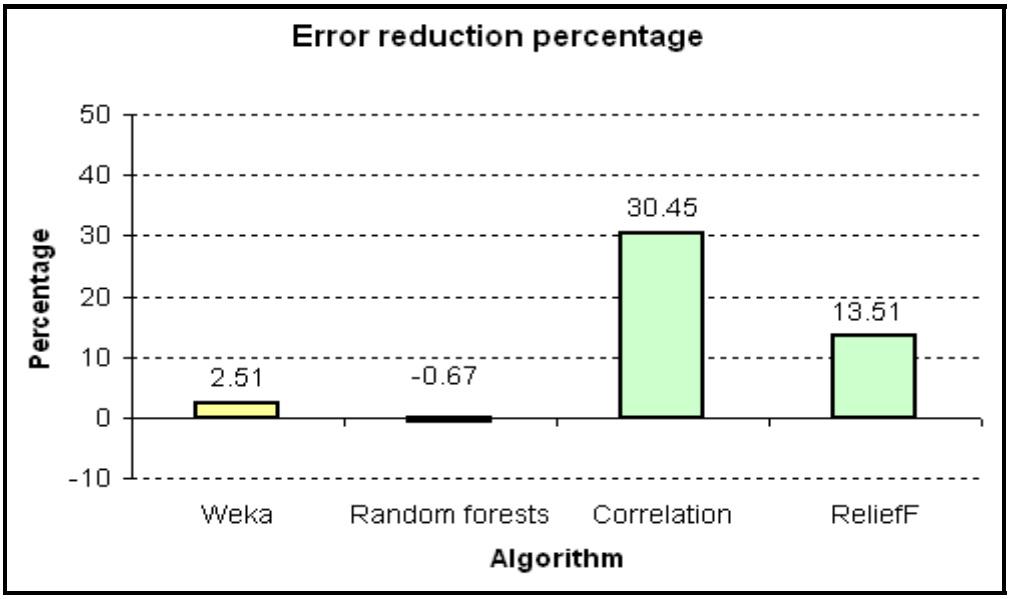

Figure 6.1 Average error reduction percentages of noise removal only and feature selection only in classification problems

However, noise-removal-only techniques do not seem to work well. Noise detection in Weka reduces the error rates by only 2.51%. Interestingly, the Random forests method resulted in a negative error reduction rate of –0.67%. A negative error reduction percentage means that the detection method is inaccurate and the noise removal by the method degrades the training data sets. This result supports our hypothesis that most noise detection algorithms cannot be used for high dimensional data sets.

2. Effects of Noise Removal and Feature Selection

Table 6.2 shows the classification accuracies and the average error reduction percentages resulting from the combinations of noise detection techniques and features selection methods. In the table, “Weka-Correlation” and “Weka-ReliefF” data sets are the training data sets whose relevant features are selected by correlation-based and ReliefF feature selection methods respectively after Weka applied its noise detection algorithm to the training data set. Similarly “Rf-Correlation” and “Rf-ReliefF” are the combinations of Random forests and correlation-based feature selection, and Random forests and ReliefF feature selection respectively. The “Correlation-Weka” data set is the training data set that Weka provided after applying its noise detection algorithm to the training data set with features pre-selected using correlation-based feature selection. The “ReliefF-Weka” data set is the training data set that Weka provided after it removed noise with relevant features selected using ReliefF feature selection. “Correlation-Rf” and “ReliefF-Rf” are the combinations of correlation-based feature selection and Random forests, and ReliefF feature selection and Random forests respectively. “NDFS-Binary” and “NDFS-Integer” mean the proposed NDFS algorithms using binary and integer representations in their genetic algorithms respectively. Also “NDFS-Binary-Cfs” is the “NDFS-Binary” algorithm with correlation-based feature implantation (refer to Chapter 4 for algorithm details).

The primary classifier has classification accuracies of 72.94% on the data sets for correlation-based feature selection with Weka, 70.28% on the data sets for ReliefF with Random forests, 76.20% on the data sets for Weka with correlation-based features selection, 70.62% on the data sets for Random forests with ReliefF, and 77.36% on the data sets for NDFS with binary representation at the 30% noise level.

Table 6.2 Effects of noise removal and feature selection in classification problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Weka-Correlation	90.24 (3.23)	87.41 (5.57)	85.42 (5.46)	82.70 (6.81)	78.32 (7.53)	72.94 (11.4)	27.27
Weka-ReliefF	85.72 (5.33)	83.37 (7.17)	81.32 (7.97)	79.48 (8.37)	75.50 (9.54)	69.75 (10.2)	11.81
Rf-Correlation	89.94 (3.99)	87.85 (4.35)	86.07 (5.02)	82.25 (5.83)	79.84 (6.36)	75.00 (10.4)	30.03
Rf-ReliefF	86.01 (6.07)	82.42 (6.18)	82.10 (8.04)	78.51 (8.14)	77.61 (8.60)	70.28 (11.0)	13.07
Correlation-Weka	90.56 (3.04)	87.46 (4.85)	86.12 (5.18)	82.68 (6.07)	81.26 (6.82)	76.20 (10.4)	32.39
Correlation-Rf	89.70 (3.31)	87.05 (4.56)	86.23 (5.13)	81.72 (6.34)	80.42 (6.40)	75.75 (10.2)	29.98
ReliefF-Weka	86.17 (6.08)	82.66 (7.54)	82.02 (8.48)	79.71 (8.06)	77.25 (9.01)	71.16 (10.8)	14.51
ReliefF-Rf	85.65 (5.81)	81.75 (6.71)	82.21 (8.02)	78.35 (8.51)	77.56 (9.05)	70.62 (10.9)	12.50
NDFS-Binary	89.68 (3.26)	89.20 (4.02)	88.52 (4.27)	85.72 (5.35)	82.13 (7.92)	77.36 (9.38)	38.28
NDFS-Binary-Cfs	90.41 (3.62)	89.86 (3.43)	88.77 (4.23)	86.03 (5.45)	84.26 (6.97)	80.22 (8.39)	43.18
NDFS-Integer	90.28 (3.42)	89.26 (3.47)	88.61 (4.29)	86.27 (5.12)	82.86 (8.73)	79.57 (8.74)	41.28

As seen in Table 6.2 and Figure 6.2, the average error rate decreases by 41.28% for NDFS with integer representation, by 32.39% for Weka with correlation-based feature selection, 29.98% for Random forests with correlation-based feature selection, and 30.04% for correlation-based feature selection with Random forests.

All NDFS algorithms show higher accuracies and average error reduction percentages than the other methods. Also the binary NDFS algorithm with correlation-based feature implantation archives the best performances by providing training data sets with high qualities to the primary classifier.

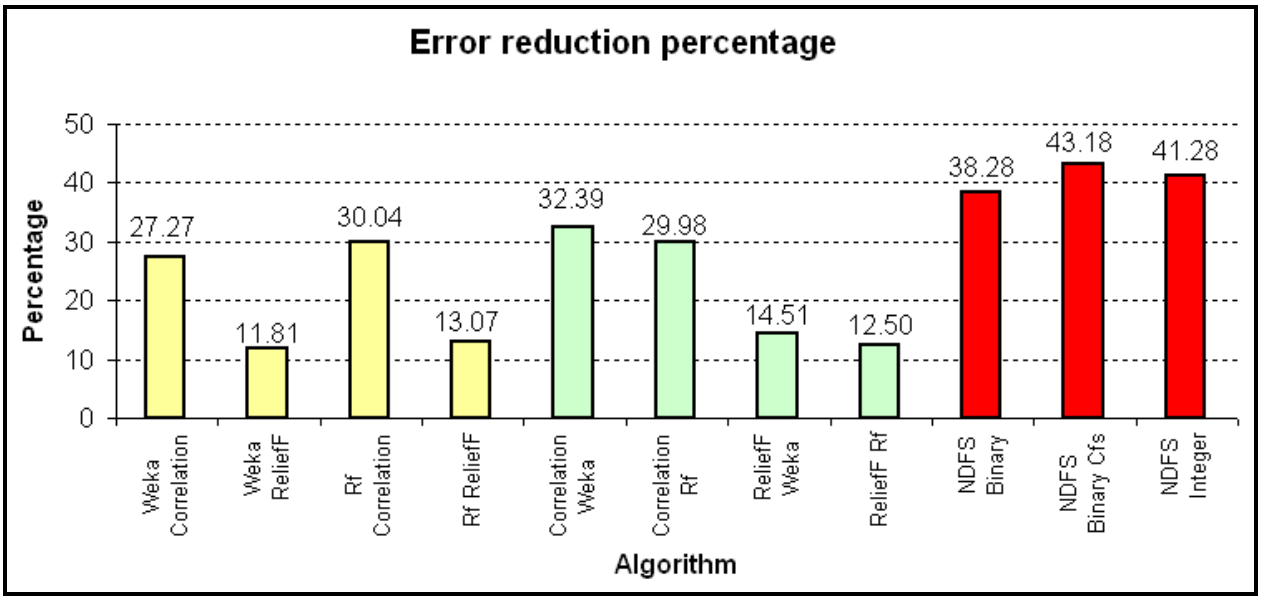

Figure 6.2 Average error reduction percentages of noise removal and feature selection in classification problems

Figures 6.1 and 6.2 appear to indicate that noise detection techniques combined with feature selection do not produce additional error reduction. Most error reductions result from feature selection. The average error reduction of correlation-based feature selection alone in

Figure 6.1 is 30.45%, while the combinations of correlation-based feature selection with Weka and Random forests by 27.27% and 30.04% respectively, and the combinations of Weka and Random forests with correlation-based feature selection reduced the average error rates by 32.39% and 29.98% respectively. Also while ReliefF shows an error reduction of 13.51% in Figure 6.1, ReliefF with Weka and Random forests only by 11.81% and 13.07% respectively, and Weka and Random forests with ReliefF reduce the error rates only by 14.51% and 12.50% respectively. These observations indicate that the sequential application of noise removal and feature selection may not overcome the concurrent negative effects of noise and high dimensionality. In the meantime, NDFS algorithms achieve high performances, overcoming these negative effects through simultaneous noise removal and feature selection.

3. Noise Detection Accuracies and Run Times

Table 6.3 and Figure 6.3 show the noise detection accuracies. All NDFS algorithms exhibited higher noise detection accuracy than the other methods. In particular, the binary NDFS with correlation-based feature implantation showed the highest average noise detection accuracy.

The Random forests noise detection method displayed higher detection accuracy than Weka. However it does not work well with feature selection methods. The detection accuracies decreased when it was combined with feature selection methods. The Random forests method constructs many trees by randomly sampling instances with replacement from the training data set and randomly selecting features. It seems that filter-based feature selection methods reduce the diversity of the feature pool for Random forests and therefore decrease its noise detection accuracy.

Table 6.3 Noise detection accuracies in classification problems

Algorithms	5%	10%	15%	20%	25%	30%	Average
Weka	95.06	91.02	85.83	81.50	77.13	71.53	83.68
Random forests	95.57	92.67	89.02	81.27	75.45	70.12	84.02
Correlation-Weka	95.40	93.21	90.65	86.81	85.02	79.83	88.48
Correlation-Rf	91.00	91.02	87.57	80.35	75.20	69.76	82.48
ReliefF-Weka	93.13	89.93	88.02	85.12	81.92	75.43	85.59
ReliefF-Rf	93.06	91.05	86.55	80.17	74.98	69.98	82.63
NDFS-Binary	95.20	94.18	93.02	90.56	87.26	81.52	90.29
NDFS-Binary-Cfs	95.73	94.32	92.46	89.92	88.25	83.30	90.66
NDFS-Integer	95.38	93.82	92.45	89.65	85.53	80.06	89.48

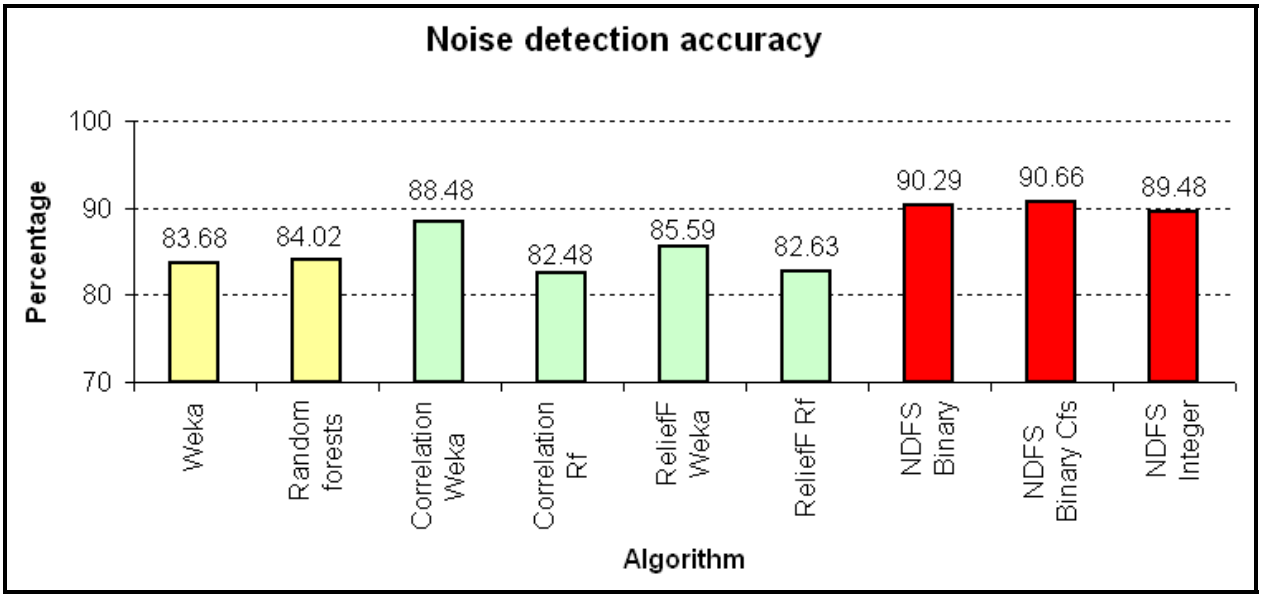

Figure 6.3 Average noise detection accuracies in classification problems

Table 6.4 describes the average run times of NDFS variants. We used a machine with 2.4 GHz and 64-bit architecture to run these experiments.

Table 6.4 Average run times per data set in classification problems

NDFS variant	NDFS-Binary	NDFS-Binary-Cfs	NDFS-Integer
Run Time	25 Min	11 Min	11 Min

Regression Problems

Tables 6.5 and 6.6 show the classification accuracies and the average error reduction percentages for the regression problems. As described above for the classification problems, the numbers in each cell of the tables are the averages of the results on 80 data sets and the numbers in parentheses indicate standard deviations for these results. Also Figure 6.4 and 6.5 compare the error reduction percentages of noise detection techniques and feature selection methods over all noise levels graphically.

1. Effects of Noise Removal Only and Features Selection Only

Similarly to the classification problems, the accuracies and average error reductions in Table 6.5 result from applying noise detection techniques and feature selection methods separately. The Sign noise detection method is used for regression problems as the Random forests method is only suited for classification problems (refer to Chapter 2 for algorithm details). However the Sign algorithm does not work if more than 50% of the values included in one or more features are equal. For our synthetic data, the Mammals and Musk data sets contain

such features, and so we could not apply the Sigh noise detection method to these data sets. Therefore the values for the Sign noise detection are the averages of the Waveform and Wdbc data sets only. In the table, the "Sign" data sets are the training data sets that Sign offered after applying its noise detection algorithm. Other terms are the same as those used in the classification problems.

Unlike the classification problems, the feature selection methods did not perform well in the regression problems. For example, as seen in Table 6.5 and Figure 6.4, correlation-based and ReliefF feature selection methods produced average error reduction rates of only 7.65% and –7.44% respectively. This observation supports our hypothesis that many feature selection methods are not applicable to noisy data sets. However while Sign noise detection did not work well, Weka did reduce errors by 19.43%.

Table 6.5 Effects of noise removal only and feature selection only in regression problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Weka	83.44 (3.36)	81.73 (3.76)	81.26 (4.47)	79.81 (3.79)	79.35 (5.17)	76.46 (5.48)	19.43
Sign	79.07 (5.80)	75.57 (6.13)	73.15 (6.75)	71.27 (7.39)	67.32 (8.41)	62.67 (15.6)	-16.79
Correlation	81.18 (3.24)	79.25 (3.91)	77.65 (4.51)	76.86 (4.31)	75.93 (5.07)	73.95 (5.63)	7.65
ReliefF	79.43 (6.54)	76.08 (6.67)	73.66 (8.32)	74.02 (7.49)	69.99 (9.54)	69.56 (8.58)	-7.44

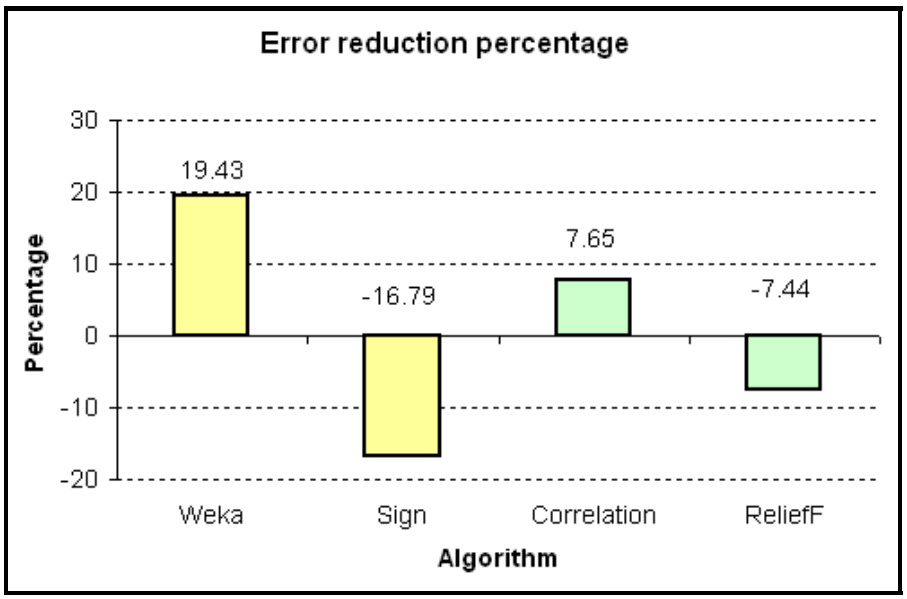

Figure 6.4 Average error reduction percentages of noise removal only and feature selection only in regression problems

2. Effects of Noise Removal and Feature Selection

As with the classification problems, the classification accuracies and the average error reduction percentages in Table 6.6 result from the combinations of noise detection techniques and features selection methods. In the table, “Sign-Correlation” and “Sign-ReliefF” data sets are the training data sets whose relevant features are selected by correlation-based and ReliefF feature selection methods respectively after Sign applied its noise detection algorithm to the training data set. Also the “Correlation-Sign” data sets are the training data sets that Sign provided after applying its noise detection algorithm to the training data set with the features pre-selected by the correlation-based feature selection method. “ReliefF-Sign” is the combination of the ReliefF feature selection method and Sign noise detection technique. As mentioned in the previous section, the Sign method could not be applied to the Mammals and Musk data sets. Therefore the values in these cells only include averages using the Waveform and Wdbc data

sets. “NDFS-Binary-Cfs 1” is the NDFS algorithm in combination with correlation-based feature implantation and using binary representation for its genetic algorithms. “NDFS-Binary-Cfs 2” is same as “NDFS-Binary-Cfs 1” except that the evaluation function used in GA-ND did not include *TreeSize* (refer to Chapter 4 for algorithm details). Other terms are the same as those used in the classification problems.

Since real valued targets are more sensitive than nominal target values, regression problems are more difficult than classification problems. According to Table 6.6, the error reduction percentages for the regression problems were not as high as for the classification problems.

Figure 6.5 shows error reduction performance for the algorithms over all noise levels graphically. Correlation-based feature selection with Weka, Weka with ReliefF, Sign with correlation-based feature selection, and Sign with ReliefF, all performed poorly. They reduced error rates only by 2.27%, 4.65%, 2.30%, and 2.79% respectively. In particular, correlation-based and ReliefF feature selections with Sign show negative error reductions. Also ReliefF with Weka achieved 11.82% error reduction, and Weka noise detection with correlation-based feature selection reduces error rates by 15.86%. However on Figures 6.4, the error reduction rate of Weka without any feature selection was 19.43%. Therefore as with the classification problems, the sequential application of noise removal and feature selection does not overcome the concurrent negative effects of noise and high dimensionality.

Table 6.6 Effects of noise removal and feature selection in regression problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Weka-Correlation	79.90 (3.41)	77.41 (5.44)	76.96 (3.72)	75.56 (3.86)	73.99 (5.26)	73.12 (4.65)	2.26
Weka-ReliefF	80.50 (6.42)	81.38 (5.66)	79.18 (6.54)	78.86 (6.03)	77.39 (8.16)	73.61 (9.35)	11.81
Sign-Correlation	79.31 (4.50)	77.60 (4.30)	75.73 (5.76)	75.11 (5.85)	72.28 (7.63)	70.74 (7.01)	-1.95
Sign-ReliefF	82.16 (4.70)	78.52 (4.85)	76.85 (6.88)	75.02 (6.13)	72.57 (7.02)	68.26 (10.7)	-0.16
Correlation-Weka	82.00 (3.07)	80.95 (3.60)	80.22 (4.14)	79.10 (3.89)	78.10 (4.88)	76.46 (5.30)	15.86
Correlation-Sign	80.97 (3.39)	78.25 (3.78)	77.18 (5.28)	75.73 (5.10)	73.32 (7.31)	71.54 (8.46)	2.30
ReliefF-Weka	80.60 (6.98)	78.23 (6.76)	76.59 (7.92)	76.94 (7.69)	74.40 (8.22)	73.67 (8.14)	4.65
ReliefF-Sign	82.43 (3.77)	79.02 (4.34)	77.62 (6.79)	75.60 (4.79)	72.78 (7.48)	70.25 (7.43)	2.79
NDFS-Binary	84.35 (2.91)	83.05 (2.86)	82.51 (3.32)	80.93 (2.72)	78.57 (4.97)	76.05 (5.53)	21.75
NDFS-Binary-Cfs 1	84.35 (3.59)	83.90 (2.81)	83.26 (2.87)	81.31 (3.29)	79.10 (5.42)	76.36 (4.67)	23.68
NDFS-Binary-Cfs 2	85.10 (2.47)	83.65 (3.49)	84.23 (2.55)	81.81 (3.62)	80.93 (4.97)	77.83 (5.49)	27.27
NDFS-Integer	83.71 (3.15	82.22 (3.24)	81.46 (3.82)	79.99 (3.87)	78.31 (4.51)	76.87 (4.46)	19.77

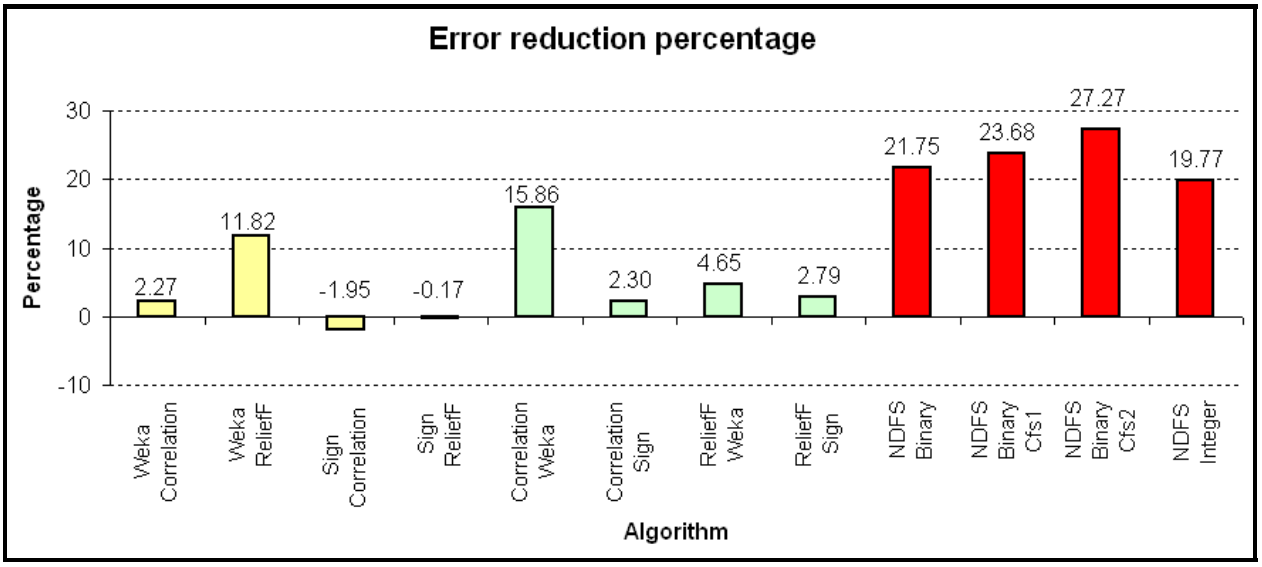

Figure 6.5 Average error reduction percentages of noise removal and feature selection in regression problems

All NDFS algorithms performed better than other methods. Most of them reduced the error rates by more than 20%. Unlike in the classification problems, NDFS with binary representation alone performed better than NDFS with integer representation. Also the binary NDFS with correlation-based feature implantation and no *TreeSize* produced the best performance by providing higher quality training data sets to the primary classifier than others.

3. Noise Detection Accuracies and Run Times

Tables 6.7 and 6.10 show the average noise detection accuracies and the run times of NDFS on the regression problems. In Table 6.7 and Figure 6.6, there is an interesting observation to be made on the detection accuracies of Weka, Weka with correlation-based feature selection, and Weka with ReliefF. All three show high noise detection accuracies relative to their error reduction rates. In particular, Weka with ReliefF exhibits an accuracy of 89.84% (see Figure 6.6) but an error reduction rate of only 4.65% (see Figure 6.5). These results are caused by low false noise detection rates and low true noise detection rates. If an algorithm has both low false noise detection rates and low true noise detection rates, it may report high detection accuracies and low error reduction rates (refer to Chapter 5 for detailed measurements of noise detection accuracy). For example, Tables 6.8 and 6.9 present confusion matrices based on the data set with 100 instances and 10% noise level. In Table 6.8, noise detection accuracy is 90% and true noise detection rate is 10%. In spite of the high detection accuracy, it may result in low error reduction because of low true noise detection rate. However in Table 6.9, although noise detection accuracy is only 85%, high error reduction can be produced due to a high true noise detection rate of 90%.

As with the classification problems, all NDFS algorithms achieved high noise detection accuracies.

Table 6.7 Noise detection accuracies in regression problems

Algorithms	5%	10%	15%	20%	25%	30%	Average
Weka	95.92	94.32	92.28	90.51	89.28	87.45	91.63
Sign	71.52	69.00	66.32	64.85	62.05	60.25	65.66
Correlation-Weka	94.70	93.26	91.77	90.62	89.13	87.46	91.16
Correlation-Sign	83.60	81.17	78.47	74.80	71.45	67.65	76.19
ReliefF-Weka	93.95	92.00	89.98	89.65	86.91	86.57	89.84
ReliefF-Sign	80.05	78.75	76.20	73.85	69.42	67.90	74.36
NDFS-Binary	95.60	94.80	93.63	91.97	89.35	85.42	91.79
NDFS-Binary-Cfs 1	95.35	95.26	94.31	92.58	90.51	86.33	92.39
NDFS-Binary-Cfs 2	96.18	95.08	94.76	92.63	90.82	87.82	92.88
NDFS-Integer	96.40	94.42	92.26	89.21	86.26	82.93	90.25

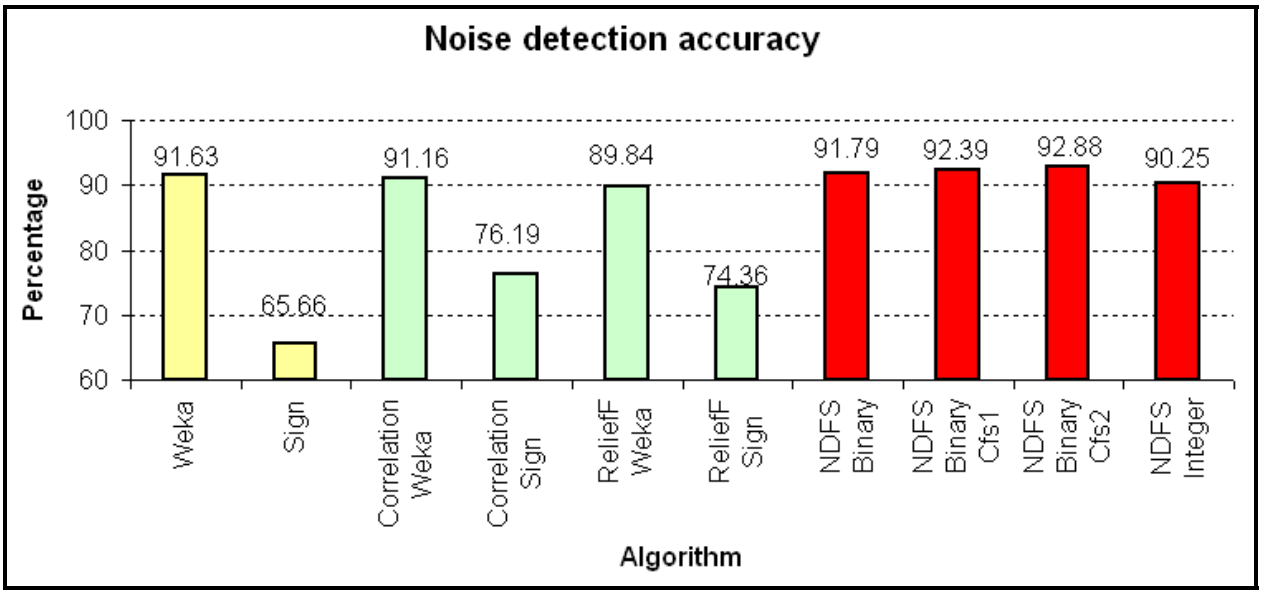

Figure 6.6 Average noise detection accuracies in regression problems

Table 6.8 Example confusion matrix with low true noise detection rate

Confusion Matrix		Instance Classification	
		Noisy	Noise-free
Actual Instance	Noisy	1	9
	Noise-free	1	89

Table 6.9 Example confusion matrix with high true noise detection rate

Confusion Matrix		Instance Classification	
		Noisy	Noise-free
Actual Instance	Noisy	9	1
	Noise-free	14	76

Table 6.10 Average run times per data set in regression problems

NDFS	NDFS-Binary	NDFS-Binary-Cfs 1	NDFS-Binary-Cfs 2	NDFS-Integer
Run Time	35 Min	28 Min	27 Min	28 Min

Results on Individual Data Sets

In the Appendix, Tables A.1 through A.8 present the classification accuracies and average error reduction percentages of each data set. The accuracies in each cell of the tables list the averages of the results of 20 data sets at each noise level (20 = 2×10; 2 different formula and 10 noisy data sets at each noise level (refer to Chapter 5 for details on the number of data sets). As seen in Tables A.1 through A.4, all NDFS algorithms exhibited the best average error reduction percentages in the classification problems. However according to Table A.7 on the Waveform data sets on the regression problems, Weka without any feature selection performed better with an average error reduction of 11.14%, while the binary NDFS with correlation-based feature implantation and no *TreeSize* only reduced the error rates by 7.55%. Meanwhile, many of the other methods returned negative error reduction percentages on the data sets. The Wdbc data sets had the smallest number of features and the primary classifier shows relatively low accuracies on the original data sets. The features selection of NDFS does not seem to work well due to low accuracies in the classifier. Therefore Weka without any feature selection may achieve better performance than NDFS algorithms. With the exception of the Waveform data sets, NDFS algorithms produce the best average error reductions in the regression problems.

CHAPTER 7

CONCLUSION

In classification and regression problems, classifiers for high dimensional noisy data suffer from the concurrent negative effects of noise and high dimensionality. Noise disrupts data, and high dimensionality prevents a classifier from focusing on relevant features; potentially reducing classification and regression accuracies.

We proposed the NDFS algorithm in order to enhance the quality of training data sets possessing noise and high dimensionality for use in classification and regression problems. NDFS relies on two genetic algorithms, one for noise detection (GA-ND) and the other for feature selection (GA-FS), and allows them to exchange their results at periodic generational intervals. Prototype selection (PS-ND) is used in conjunction with the genetic algorithm to improve the performance of our noise detection method.

We extrapolated synthetic data sets from the UCI machine learning repository to simulate real world data. Our synthetic data sets included a fixed number of instances (100) and various numbers of features (21 through 168). Then we generate noisy data sets by applying the noise levels from 5% to 30%.

According to our experimental results, most noise detection techniques are not effective on high dimensional data sets, and many feature selection methods are not applicable to noisy data sets. Also the sequential application of noise detection and feature selection algorithms may not overcome the concurrent negative effects of noise and high dimensionality.

However NDFS overcomes the concurrent effects of noise and high dimensionality, and achieves high performance by performing noise removal and feature selection simultaneously. We have shown that the NDFS algorithm significantly enhanced the quality of our high dimensional noisy data sets for classification and regression problems. As our results demonstrated, NDFS substantially increased the classification accuracies and reduced the error rates of our synthetic data sets.

However NDFS has two drawbacks. NDFS highly depends on the accuracy of its classifier. If the classifier returns lower accuracy on the original data set, the NDFS algorithm will exhibit lower performance. Another weakness of NDFS relates to feature selection. If the classifier produces high accuracy on binary classification data, NDFS will select only one or two features that are far less relevant than the real relevant features of our data sets. This is because a small number of features is enough to explain the target of the training data. In this case, the model overfits the training features and the classifier does not generalize well on testing data. The problem can be solved by increasing the number of instances.

REFERENCES

[1] C. C. Aggarwal and P. S. Yu, Outlier detection for high dimensional data, *Proceedings of ACM SIGMOD Conference*, 2001.

[2] C. C. Aggarwal, On abnormality detection in spuriously populated data streams, *Proceedings of SIAM International Conference on Data Mining*, 2005.

[3] H. Almuallium and T. G. Dietterich, Learning with many irrelevant features, *Proceedings of National Conference on Artificial Intelligence*, 1991.

[4] A. Arning, R. Agrawal and P. Raghavan, A linear method for deviation detection in large databases, *International Conference on Knowledge Discovery and Data Mining*, 1996.

[5] V. Barnett and T. Lewis, *Outliers in Statistical Data*, John Wiley and Sons, 1994.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

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

[8] L. Breiman, Manual on setting up, using, and understanding random forests v 3.1, Author's Web (http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf), 2002.

[9] M. M. Breunig, H. Kriegel, R. T. Ng and J. Sander, LOF: identifying density-based local outliers, *Proceedings of International Conference on Management of Data*, 2000.

[10] H. Brighton and C. Mellish, Advances in instance selection for instance-based learning algorithms, *Data Mining and Lnowledge Discovery,* 6: 153-172, 2002.

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

[12] E. Cantu-Paz, Feature subset selection, class separability, and genetic algorithms, *Genetic and Evolutionary Computation Conference*, 2004.

[13] C. Cardie, Using decision trees to improve case-based learning, *Proceedings of International Conference on Machine Learning*, 1993.

[14] S. Cateni, V. Colla and M. Vannucci, Outlier Detection Methods for Industrial Applications, *Advances in Robotics, Automation and Control*, 2008.

[15] V. Chandola, A. Banerjee, and V. Kumar, Outlier detection: a survey. *Technical Report of University of Minnesota*, 2007.

[16] V. Cherkassky and F. M. Mulier, *Learning from Data: Concepts, Theory, and Methods*, Wiley-IEEE Press, 2007.

[17] K. D. Crawford and R. L. Wainwrit, Applying genetic algorithms to outlier detection, *Proceedings of International Conference on Genetic Algorithms*, 1995.

[18] M. Dash and H, Liu, Feature selection for classification, *Intelligent Data Analysis,* 1: 131-156, 1997.

[19] M. Dash and H, Liu, Consistency-based search in feature selection, *Artificial Intelligence,* 151: 155-176, 2003.

[20] M.Ester, H. Kriegel, J. Sander and X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, *Proceedings of International Conference on Knowledge Discovery and Data Mining*, 1996.

[21] P. Filzmoser, R. Maronna and M. Werner, Outlier identification in high dimensions, *Computational Statistics and Data Analysis,* 52(3): 1694-1711, 2007.

[22] I. K. Fodor, A survey of dimension reduction techniques, *Technical Report of the US Department of Energy*, 2002.

[23] M. Hall, Correlation-based feature selection for discrete and numeric class machine learning, *Proceedings of International Conference on Machine Learning*, 2000.

[24] Trevor Hastie, *The Elements of Statistical Learning*, Springer, 2001.

[25] S. Hawkins, H. He, G. Williams and R. Baxter, Outlier detection using replicator neural networks, *Proceedings of International Conference on Knowledge Discovery and Data Warehousing*, 2002.

[26] Z. He, X. Xu and S. Deng, Discovering cluster based local outliers, *Pattern Recognition Letters,* 24 (9-10): 1641-1650, 2003.

[27] V. Hodge and J Austin, A survey of outlier detection methodologies, *Artificial Intelligence Review,* 22: 85 – 126, 2004.

[28] A. Hyvarinen, Fast and robust fixed-point algorithms for independent component analysis, *EEE Trans. on Neural Networks,* 10: 626-634, 1999.

[29] A. Hyvarinen and E. Oja, Independent component analysis: algorithms and applications, *Neural Networks,* 13(4-5): 411-430, 2000.

[30] J. Jarmulak and S. Craw, Genetic algorithms for feature selection and weighting, *Proceeding of IJCAI*, 1999.

[31] G. H. John, R. Kohavi and K. Pfleger, Irrelevant features and the subset selection problem, *Proceedings of International Conference on Machine Learning*, 1994.

[32] G. H. John, Robust decision trees: removing outliers from databases, *Proceedings of International Conference on Knowledge Discovery and Data Mining*, 1995.

[33] L. Jourdan, C. Dhaenens and E. Talbi, A genetic algorithm for feature selection in data mining for genetics, *Metaheuristics International Conference*, 2001.

[34] E. M. Knorr and R. T. Ng, Algorithms for mining distance-based outliers in large datasets, *Proceedings of the 24rd International Conference on Very Large Data Bases,* 1998.

[35] J. D. Kelly and L. Davis, A hybrid genetic algorithm for classification, *Proceedings of International Conference on Artificial Intelligence*, 1991.

[36] I. Kononenko, Estimating attributes: analysis and extensions of RELIEF, *European Conference on Machine Learning,* 1:111-117, 1994.

[37] I. Kononenko and M. Kukar, *Machine Learning and Data Mining*, Horwood Publishing, 2007.

[38] S. Kotsiantis, D. Kanellopoulos and P. Pintelas, Data Preprocessing for Supervised Leaning, International Journal of Computer Science, 2006.

[39] H. Liu and R. Setiono, Feature selection and classification – a probabilistic wrapper approach, *Proceeding of Industrial and Engineering Applications of AI and ES*, 1996.

[40] H. Liu and R. Setiono, A probabilistic approach to feature selection – a filter solution, *Proceedings of International Conference on Machine Learning*, 1996.

[41] T. Lane and C. E. Brodley, Sequence matching and learning in anomaly detection for computer security, *AAAI Technical Report*, 1997.

[42] P. L. Lanzi, Fast feature selection with genetic algorithms: a filter approach, *Proceedings of International Conference on Evolutionary Computation*, 1997.

[43] Y. Li and H. Kitagawa, Example-based db-outlier detection from high dimensional datasets, *Proceedings of International Conference on Database Systems for Advanced Application*, 2008.

[44] L. C. Molina, L. Belanche and A. Nebot, Feature Selection Algorithms: A Survey and Experimental Evaluation, *Proceedings of International Conference on Data Mining*, 2002.

[45] F. Muhlenbach, S. Lallich and D. A. Zighed, Identifying and handling mislabeled instances, *Journal of Intelligent Information Systems*, 22(1): 89-109, 2004.

[46] R. T. Ng and J. Han, Efficient and effective clustering methods for spatial data mining, *Proceedings of International Conference on Very Large Data Bases*, 1994.

[47] F. Pernkopf and P. O'Leary, Feature selection for classification using genetic algorithms with a novel encoding", *Proceedings of International Conference on Computer Analysis of Images and Patterns*, 2001.

[48] W. F. Punch, E. D. Goodman, M. Pei, L. Chia-Shun, P. Hovland and R. Enbody, Further research on feature selection and classification using genetic algorithms, *Proceedings of International Conference on Genetic Algorithms*, 1993.

[49] S. Ramaswamy, R. Rastogi and K. Shim, Efficient algorithms for mining outliers from large data sets, *Proceedings of Conference on Management of Data*, 2000.

[50] M. L. Raymer, W. F. Punch and E. D. Goodman, Simultaneous feature extraction and selection using a masking genetic algorithm, *Presentation at International Conference on Genetic Algorithms*, 1997.

[51] M. L. Raymer, W. F. Punch, E. D. Goodman, L. A. Kuhn and A. K. Jain, Dimensionality reduction using genetic algorithms, *EEE transactions on evolutionary computation*, 4(2): 164-171, 2000.

[52] O. Ritthoff, R. Klinkenberg, S. Fischer and I. Mierswa, A hybrid approach to feature selection and generation using an evolutionary algorithm, *Proceedings of U.K. Workshop on Computational Intelligence*, 2002.

[53] S. J. Roberts, Novelty detection using extreme value statistics, *IEE Proceedings on Vision, Image and Signal Processing*, 1998.

[54] M. Robnik-Sikonja, I. Kononenko, An adaptation of Relief for attribute estimation in regression, *International Conference on Machine Learning*, 1997.

[55] R. Saunders and J. S. Gero, A curious design agent: a comutational model of novelty-seeking behavior in design, *Proceedings of Conference on Computer Aided Architectural Design Research in Asia*, 2001.

[56] M. Sebban, Prototype selection from homogenous subsets by a monte carlo sampling, *Proceedings of the International FLAIRS Conference*, 1998.

[57] B. Sierra, E. Lazkano, I. Inza, M. Merino, P. Larranaga and J. Quiroga, Prototype selection and feature subset selection by estimation of distribution algorithms, *Proceedings of Conference on AI in Medicine in Europe*, 2001.

[58] M. Singh and G. M. Provan, Efficient learning of selective Bayesian network classifiers, *Proceedings of International Conference on Machine Learning*, 1996.

[59] D. B. Skalak, Using a genetic algorithm to learn prototypes for case retrieval and classification, *Proceedings of the AAAI-93 Case-Based Reasoning Workshop*, 1993.

[60] D. B. Skalak, Prototype and feature selection by sampling and random mutation hill climbing algorithms, *Machine Learning: Proceedings of International Conference*, 1994

[61] L. I. Smith, *A tutorial on principal components analysis*, 2002.

[62] J. V. Stone, *Independent Component Analysis*, The MIT Press, 2004.

[63] D. M. J. Tax, A. Ypma and R. P. W. Duin, Support vector data description applied to machine vibration analysis, *Proceedings of ASCI*, 1999.

[64] P. H. S. Torr and D. W. Murray, Outlier detection and motion segmentation, *Proceedings of SPIE*, 1993.

[65] H. Vafaie and K. D. Jong, Genetic algorithms as a tool for feature selection in machine learning, *Proceeding of International Conference on Tools with Artificial Intelligence*, 1992.

[66] H. Vesanto, J. Himberg, M. Siponen and O. Simula, Enhancing SOM based data visualization, *Proceedings of International Conference on Soft Computing and Information Systems*, 1998.

[67] G. Williams, R. Baxter, H. He and S. Hawkins, A comparative study of RNN for outlier detection in data mining, *Proceedings of the IEEE International Conference on Data Mining*, 2002.

[68] I. H. Witten and E. *Frank, Data Mining*, Morgan Kaufmann Publishers, 2005.

[69] H. Xoing, G. Pandey and M. Steinbach, Enhancing data analysis with noise removal, *IEEE Transactions on Knowledge and Data Engineering*, 18(3): 304-319, 2006.

[70] J. Yang and V. Honavar, Feature subset selection using a genetic algorithm, *IEEE Intelligent Systems*, 13(2): 44-49, 1998.

[71] N. Ye and Q Chen, An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems, *Quality and Reliability Engineering International*, 17:105-112, 2001.

[72] A. Ypma and R. P. W. Duin, Novelty detecting using self-organizing maps, *Progress in Connectionist-Based Information Systems*, 2: 1322-1325, 1997.

[73] L. Yu and H. Liu, Feature selection for high-dimensional data: a fast correlation-based filter solution, *Proceedings of International Conference on Machine Learning*, 2003.

[74] T. Zhang, R. Ramakrishnan and M. Livny, BIRCH: an efficient data clustering method for very large databases, *Proceedings of International Conference on Management of Data*, 1996.

APPENDIX

Table A.1 Classification accuracies and average error reduction percentages on Mammals for classification problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Original	92.15	86.40	82.55	76.65	73.55	64.80	-
Weka	92.40	87.15	83.45	76.95	74.25	64.95	2.47
Random forests	91.00	86.00	81.45	76.60	74.60	64.05	-1.93
Correlation	95.10	90.00	88.75	86.40	80.25	73.20	30.45
ReliefF	95.10	92.00	91.05	87.25	83.05	75.55	38.75
Weka-Correlation	94.15	91.15	90.15	86.60	79.45	74.65	32.32
Weka-ReliefF	94.70	92.75	91.30	85.90	82.25	77.80	39.22
Rf-Correlation	94.35	90.25	89.25	87.60	79.55	73.85	31.27
Rf-ReliefF	94.95	91.35	89.10	86.60	83.50	75.85	36.52
Correlation-Weka	95.25	90.25	89.25	86.55	81.55	74.40	33.30
Correlation-Rf	95.05	89.50	90.20	86.15	79.25	73.15	29.99
ReliefF-Weka	95.20	92.50	91.00	86.95	83.10	76.70	39.91
ReliefF-Rf	95.25	91.10	90.70	87.05	84.40	75.45	38.69
NDFS-Binary	94.95	94.50	94.10	92.55	86.00	84.15	56.82
NDFS-Binary-Cfs	94.95	94.75	94.10	91.30	87.40	85.10	57.67
NDFS-Integer	95.70	93.95	93.95	91.60	88.35	83.35	57.19

Table A.2 Classification accuracies and average error reduction percentages on Musk for classification problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Original	86.50	83.65	77.90	74.45	67.85	61.75	-
Weka	87.05	83.35	78.40	73.85	67.75	65.05	2.20
Random forests	86.35	83.85	78.05	72.95	67.60	61.55	-1.21
Correlation	90.65	88.65	86.85	83.45	84.00	79.55	41.40
ReliefF	90.15	85.70	84.70	86.90	80.00	73.65	33.29
Weka-Correlation	91.30	88.60	86.00	82.55	79.90	71.95	32.58
Weka-ReliefF	89.15	86.50	84.45	84.10	79.50	70.85	28.70
Rf-Correlation	90.20	89.00	87.70	83.55	83.15	79.45	41.21
Rf-ReliefF	90.25	87.30	84.85	82.65	80.85	73.65	32.08
Correlation-Weka	90.75	88.35	87.75	84.35	84.00	79.90	42.73
Correlation-Rf	90.20	88.55	87.60	83.10	84.00	79.55	41.29
ReliefF-Weka	90.05	86.05	84.80	81.80	80.25	73.80	30.45
ReliefF-Rf	90.20	85.50	85.60	81.90	80.00	74.10	30.81
NDFS-Binary	89.85	89.65	87.40	83.85	79.90	75.00	36.38
NDFS-Binary-Cfs	91.25	90.05	88.85	85.75	84.65	81.05	47.14
NDFS-Integer	90.20	91.05	89.20	85.90	83.20	78.95	45.11

Table A.3 Classification accuracies and average error reduction percentages on Waveform for classification problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Original	71.05	68.10	66.65	60.30	58.10	56.50	-
Weka	72.95	69.70	66.20	63.45	58.05	56.75	2.83
Random forests	70.05	67.75	66.55	60.40	58.10	56.65	-0.59
Correlation	76.90	72.50	70.65	63.00	65.45	64.40	14.56
ReliefF	73.45	70.20	68.60	59.90	63.60	56.40	5.47
Weka-Correlation	76.45	73.40	70.70	68.70	63.50	61.55	15.32
Weka-ReliefF	74.80	72.05	68.15	64.75	62.70	57.90	8.96
Rf-Correlation	75.35	73.10	70.40	62.10	65.45	64.50	13.77
Rf-ReliefF	72.60	70.55	68.60	58.95	63.60	56.50	4.60
Correlation-Weka	77.75	73.35	71.60	65.05	66.45	64.40	17.26
Correlation-Rf	75.35	72.70	70.70	63.00	65.55	64.20	13.90
ReliefF-Weka	75.00	72.00	69.10	63.05	63.65	57.50	9.10
ReliefF-Rf	72.90	70.60	68.25	58.85	63.65	56.60	4.76
NDFS-Binary	76.55	75.10	74.15	69.45	68.40	62.35	20.36
NDFS-Binary-Cfs	77.45	77.05	74.50	70.45	70.20	64.65	24.31
NDFS-Integer	77.20	74.25	73.00	71.05	68.35	64.30	21.42

Table A.4 Classification accuracies and average error reduction percentages on Wdbc for classification problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Original	95.80	93.25	89.10	86.15	84.05	76.55	-
Weka	96.00	93.20	89.90	86.45	83.35	77.40	1.16
Random forests	95.90	93.40	90.00	86.20	84.40	76.55	1.84
Correlation	98.50	97.90	95.70	94.85	93.40	86.10	56.52
ReliefF	84.40	80.15	82.85	87.20	82.10	76.35	-57.04
Weka-Correlation	98.40	96.65	94.65	91.95	90.80	84.90	43.20
Weka-ReliefF	83.30	81.70	82.80	84.45	79.70	75.15	-50.33
Rf-Correlation	97.70	98.10	96.90	95.25	92.85	86.10	55.92
Rf-ReliefF	86.45	79.95	86.85	86.50	82.10	76.35	-35.55
Correlation-Weka	98.50	97.90	95.90	94.80	93.05	86.10	56.29
Correlation-Rf	98.20	97.45	96.45	94.65	92.90	86.10	55.33
ReliefF-Weka	84.45	80.10	83.20	87.05	82.00	76.65	-56.67
ReliefF-Rf	84.25	79.80	84.30	85.60	82.20	76.35	-57.68
NDFS-Binary	97.40	97.55	98.45	97.05	94.25	87.95	64.62
NDFS-Binary-Cfs	98.00	97.60	97.65	96.65	94.80	90.10	67.70
NDFS-Integer	98.05	97.80	98.30	96.55	91.55	91.70	65.43

Table A.5 Classification accuracies and average error reduction percentages on Mammals for regression problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Original	93.47	91.95	90.48	89.55	87.31	86.90	-
Weka	94.73	93.89	93.30	91.81	90.59	90.03	24.01
Correlation	91.98	90.65	89.18	89.70	88.70	87.70	-2.13
ReliefF	89.71	86.15	83.81	84.82	80.46	82.27	-54.40
Weka-Correlation	92.62	92.66	90.61	89.99	89.45	88.25	7.46
Weka-ReliefF	89.88	88.94	89.26	88.97	84.78	86.85	-18.16
Correlation-Weka	93.30	92.13	90.67	91.17	89.56	88.92	10.86
ReliefF-Weka	91.14	86.77	85.44	85.60	81.47	83.79	-43.21
NDFS-Binary	97.45	96.64	95.65	95.58	94.90	93.24	55.16
NDFS-Binary-Cfs 1	97.70	96.44	96.17	96.34	95.50	94.08	60.02
NDFS-Binary-Cfs 2	97.11	97.11	97.13	96.71	96.22	95.70	66.46
NDFS-Integer	97.41	96.76	95.68	94.83	94.01	02.80	52.11

Table A.6 Classification accuracies and average error reduction percentages on Musk for regression problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Original	67.25	65.63	64.16	61.04	58.77	56.57	-
Weka	67.75	65.45	66.19	62.96	64.07	59.39	5.57
Correlation	70.34	66.56	65.70	63.59	64.01	60.41	7.77
ReliefF	61.99	58.33	53.72	56.50	51.59	49.28	-19.16
Weka-Correlation	68.36	61.79	65.76	62.04	61.97	62.20	3.84
Weka-ReliefF	61.84	68.26	59.27	61.42	61.16	47.74	-6.05
Correlation-Weka	70.65	67.61	67.70	63.92	63.79	61.69	9.91
ReliefF-Weka	60.61	58.55	53.99	56.02	53.23	52.08	-17.82
NDFS-Binary	68.76	65.40	64.88	64.28	59.79	58.58	3.86
NDFS-Binary-Cfs 1	69.82	70.05	68.24	65.57	62.15	59.52	9.81
NDFS-Binary-Cfs 2	69.59	68.38	70.07	67.20	64.00	59.81	11.47
NDFS-Integer	70.09	66.91	66.48	65.54	62.23	61.47	8.62

Table A.7 Classification accuracies and average error reduction percentages on Waveform for regression problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Original	71.66	68.60	68.39	65.70	63.51	61.53	-
Weka	73.92	71.85	71.36	69.89	69.26	65.30	11.14
Sign	70.98	67.24	65.53	64.14	61.72	58.43	-5.66
Correlation	68.47	67.67	65.88	65.93	63.43	63.61	-2.22
ReliefF	71.32	68.29	67.31	66.90	61.76	62.39	-0.68
Weka-Correlation	68.18	67.72	67.42	65.71	64.15	64.17	-1.00
Weka-ReliefF	72.13	71.35	71.42	68.88	68.35	65.49	9.09
Sign-Correlation	67.94	67.44	64.82	65.54	62.09	61.59	-4.95
Sign-ReliefF	72.29	67.38	67.61	66.06	63.18	59.77	-1.54
Correlation-Weka	66.44	67.01	66.14	65.67	63.95	63.34	-3.33
Correlation-Sign	68.12	67.38	64.86	65.79	63.14	63.49	-3.32
ReliefF-Weka	72.45	69.94	69.84	70.15	66.90	64.13	7.06
ReliefF-Sign	71.92	67.26	66.22	65.86	62.19	60.29	-2.84
NDFS-Binary	73.77	72.83	71.68	67.12	64.65	60.81	5.77
NDFS-Binary-Cfs 1	72.86	71.89	71.35	66.64	62.70	59.32	2.73
NDFS-Binary-Cfs 2	75.35	71.66	71.87	66.44	66.79	62.29	7.55
NDFS-Integer	69.91	68.55	68.05	64.82	62.87	62.84	-1.14

Table A.8 Classification accuracies and average error reduction percentages on Wdbc for regression problems

Algorithms	5%	10%	15%	20%	25%	30%	Error reduction %
Original	92.66	88.91	83.51	83.31	79.12	74.89	-
Weka	97.39	95.75	94.21	94.62	93.51	91.16	65.97
Sign	87.18	83.91	80.78	78.42	72.94	66.93	-35.03
Correlation	93.97	92.16	89.87	88.26	87.59	84.09	34.23
ReliefF	94.72	91.59	89.81	87.91	86.18	84.31	32.27
Weka-Correlation	98.13	97.16	96.36	95.32	95.07	92.82	74.25
Weka-ReliefF	98.16	96.96	96.78	96.19	95.27	94.37	77.20
Sign-Correlation	90.68	87.76	86.65	84.68	82.47	79.89	9.98
Sign-ReliefF	92.04	89.67	86.09	83.99	81.95	76.76	8.31
Correlation-Weka	97.65	97.08	96.41	95.67	95.15	91.92	73.90
Correlation-Sign	93.84	89.14	89.52	85.68	83.52	79.61	19.08
ReliefF-Weka	98.21	97.70	97.12	96.01	96.04	94.71	79.69
ReliefF-Sign	92.95	90.80	89.04	85.35	83.39	80.23	18.48
NDFS-Binary	97.45	97.35	97.85	96.77	94.98	91.58	75.81
NDFS-Binary-Cfs 1	97.04	97.25	97.30	96.72	96.08	92.56	77.01
NDFS-Binary-Cfs 2	98.38	97.47	97.87	96.90	96.72	93.53	80.75
NDFS-Integer	97.48	96.68	95.66	94.81	94.16	90.39	68.88