Nearest Neighbor Classification from Multiple Feature Subsets

Stephen D. Bay
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697, USA
sbay@ics.uci.edu

February 1, 1999

Abstract

Combining multiple classifiers is an effective technique for improving accuracy. There are many general combining algorithms, such as Bagging, Boosting, or Error Correcting Output Coding, that significantly improve classifiers like decision trees, rule learners, or neural networks. Unfortunately, these combining methods do not improve the nearest neighbor classifier. In this paper, we present MFS, a combining algorithm designed to improve the accuracy of the nearest neighbor (NN) classifier. MFS combines multiple NN classifiers each using only a random subset of features. The experimental results are encouraging: On 25 datasets from the UCI Repository, MFS significantly outperformed several standard NN variants and was competitive with boosted decision trees. In additional experiments, we show that MFS is robust to irrelevant features, and is able to reduce both bias and variance components of error.

Keywords: multiple models, combining classifiers, nearest neighbor, feature selection, voting.

1 Introduction

The nearest neighbor (NN) classifier is one of the oldest and simplest methods for performing general, non-parametric classification. It can be represented by the following rule: to classify an unknown pattern, choose the class of the nearest example in the training set as measured by a distance metric. A common extension is to choose the most common class in the k nearest neighbors (kNN).

Despite its simplicity, the nearest neighbor classifier has many advantages over other methods. For example, it can learn from a small set of examples, can incrementally add new information at runtime, and can give competitive performance with more modern methods such as decision trees or neural networks.

Since its inception by Fix and Hodges [25], researchers have investigated many methods for improving the NN classifier, but most work has concentrated on changing the distance metric or manipulating the patterns in the training set [4, 19]. Recently, researchers have begun experimenting with general algorithms for improving classification accuracy by combining multiple versions of a single classifier, also known as a *multiple model* or *ensemble* approach. The outputs of several classifiers are combined in the hope that the accuracy of the whole is greater than the parts. Unfortunately, many combining methods do not improve the NN classifier at all. For example, neither Bagging nor Error Correcting Output Coding (ECOC) improves the NN classifier [9, 44].

In this paper, we present a new method of combining nearest neighbor classifiers with the goal of improving classification accuracy. Our approach uses random features for the individual classifiers. In contrast, other combining algorithms usually manipulate the training patterns (Bagging, Boosting) or the class labels (ECOC).

In the next section, we discuss current ensemble approaches and their limitations with respect to the nearest neighbor classifier. In Section 3 we describe the algorithm and investigate its time and space complexity. In Section 4 we evaluate the algorithm on datasets from the UCI repository for accuracy, computational complexity, and robustness to irrelevant features. Next, in Section 5 we analyze the algorithm's bias and variance components of error. In Section 6, we discuss related work, and follow it by conclusions and future directions in Section 7.

2 Limitations of Existing Ensemble Methods

An ensemble of classifiers must be both diverse and accurate in order to improve accuracy of the whole. We require diversity to ensure that all the classifiers do not make the same errors. If the classifiers make identical errors, then these errors will propagate through voting to the whole ensemble. We require accuracy to ensure that in the ensemble vote, too many poor classifiers do not drown out the votes of correct classifiers.

Hansen and Salamon [28] formulated these diversity and accuracy requirements precisely: They showed that under simple voting, if all models have the same probability of error, the probability of error is less than 50%, and the errors are independent,

then the overall error by voting will decrease monotonically with increasing number of models. Ali and Pazzani [5] verified these results by showing that uncorrelated errors led to improved ensemble performance.

There are many methods for creating an ensemble with the above properties. We will examine two main families of ensemble generation algorithms: sampling and error correcting output codes. We will discuss each one in the context of nearest neighbor classifiers.

2.1 Sampling Training Patterns

The most straightforward method for developing a set of classifiers is to train each one on a different set of patterns. The implicit assumption is that using different training sets will decorrelate the errors enough to improve performance.

Many researchers have experimented with sampling from the original training set to generate new datasets for the classifiers to learn on. Some sampling methods include: sampling with replacement [9], sampling without replacement [33], adaptive sampling [26], and mutually exclusive partitioning [13].

Bagging [9], which uses sampling with replacement, is one of the best known methods for generating a set of classifiers. Analyzing Bagging provides some insights as to why sampling the training set is not effective with nearest neighbor classifiers.

Bagging is a general method of combining classifiers that can be applied to any base method. In Bagging we create n datasets by sampling patterns with replacement from the original training set. Each of the n datasets has the same number of patterns as the original training set. This sampling method (bootstrap sampling) results in each pattern appearing in a given dataset with approximately 63.2% probability (see [9] for a detailed explanation). We then train a classifier on each dataset and combine their outputs using simple voting.

Bagging has obtained impressive error reductions with decision trees such as CART [9] and C4.5 [41, 26] on a wide range of datasets. However, when Breiman [9] applied Bagging to the nearest neighbor classifier he found no difference in performance between the bagged and non-bagged classifiers. He makes the following comment on when we can expect Bagging to improve classification:

> The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy.

Perturbing the training set by bootstrap sampling does not change the nearest neighbor classifier significantly. Since each pattern appears in approximately 63.2% of the n datasets, the nearest neighbor will be exactly the same in 63.2% of the NN classifiers. This results in high error correlation.

Alternatively, consider what happens when we add a single pattern to the training dataset. For the NN classifier the new pattern can change the classification in just a small local region around the point. In comparison, in a decision tree the addition of a single point may cause a different feature to be selected for splitting resulting in an

entirely different subtree. This is especially true as we descend the tree in a separate and conquer approach and each node has fewer patterns to measure the splitting criterion.

The stability of the NN classifier to the variations of patterns in the training set makes it unlikely that other methods that involve any significant degree of resampling or replication of patterns would work well.

Exclusive partitioning of patterns may help avoid some of the error correlation problems; however, it has problems with finite amounts of data. In particular, as we increase the number of partitions (and reduce the size of each) the training sets will become less similar. This should aid in decorrelating error, however, the accuracy of the individual classifiers will also decrease because they have less data to train from. Conversely, if we choose to use a small number of classifiers each with a large number of patterns the training sets become more similar and this may prevent the classifiers from generating different hypotheses.

A popular alternative to Bagging is Boosting, which uses adaptive sampling of patterns to generate the ensemble. In Boosting [26], the classifiers in the ensemble are trained serially, with the weights on the training instances set adaptively according to the performance of the previous classifiers. If the classifier does not directly support weighted instances, this can be simulated by sampling from the training set with probability proportional to an instances weight. The main idea is that the classification algorithm should concentrate on the difficult instances.

Boosting can generate more diverse ensembles of decision trees than Bagging does because of its ability to manipulate the input distributions [21]. However, it is not clear how one should apply Boosting to the NN classifier for the following reasons: (1) Boosting stops when a classifier obtains 100% accuracy on the training set, but this is always true for the NN classifier. (2) Increasing the weight on a hard to classify instance does not help to correctly classify that instance as each prototype can only help classify its neighbors, not itself.

Freund and Schapire [26] applied a modified version of Boosting to the NN classifier that worked around these problems by limiting each classifier to a small number of prototypes. However, their goal was not to improve accuracy, but to improve speed while maintaining current performance levels. In their boosted NN classifier, each weak hypothesis was defined by a set of prototypes and a mapping to class labels. In each round of Boosting, additional training examples were added to the prototype set until a prespecified size was reached. They used 10 prototypes for the first hypothesis, 20 for the next, 40 for the third, and 80 for the next five, and 100 for any additional hypotheses. They did not report any accuracy improvements on the NN classifier using all of the training patterns.

2.2 Error Correcting Output Codes

Error-correcting output coding (ECOC) proposed by Dietterich and Bakiri [22] is a method for creating and combining classifiers for multi-class problems by decomposition into multiple two class problems.

In ECOC we assign a codeword or binary string for each class. Each bit position

in the string corresponds to a binary function on the original space and represents the separation of the classes into two disjoint sets. For example, given four classes: A, B, C, and D, then the binary functions could correspond to separating {A} from {BCD}, {B} from {ACD}, or {AD} from {BC}, and so on. For each position we train a classifier which learns to discriminate the two sets.

To classify an unknown pattern we present it to all of the classifiers. With the classifier outputs we create a new string which corresponds to the unknown pattern. We choose the class whose codeword is closest as measured by Hamming distance to the new string. Given proper design, the output codes can tolerate errors in the codeword bits or corresponding classifiers, and hence improve accuracy in an ensemble of classifiers.

Dietterich and Bakiri showed that ECOC could improve decision trees and neural networks on several multi-class problems from the UCI repository. In an investigation of why ECOC improves performance for some classifiers, Kong and Dietterich [34] conclude that ECOC will not work with classifiers that use local information such as the kNN classifier. Wettschereck and Dietterich [53] verified that ECOC did not improve local learners (generalized radial basis function networks [39]), on the NETtalk problem [46].

ECOC fails to work for NN classifiers because the errors are correlated across the binary learning problems [1]. For example, with the NN classifier we predict the class of the closest pattern. This pattern will be the same in all of the binary problems, and hence if it gives the incorrect prediction, all the predictions will be incorrect. This is 100% error correlation, exactly what we would like to avoid.

3 Classification from Multiple Feature Subsets

The algorithm for nearest neighbor classification from multiple feature subsets (MFS) is simple and can be stated as:

> **Using simple voting, combine the outputs from multiple NN classifiers, each having access only to a random subset of features.**

We select the random subset of features by sampling from the original set of features. We use two different sampling functions: sampling with replacement, and sampling without replacement. In sampling with replacement, a feature can be selected more than once which we treat as increasing its weight.

Each of the NN classifiers uses the same number of features. This is a parameter of the algorithm which we set by cross-validation performance estimates on the training set (see Section 3.1 for more details). Each time a pattern is presented for classification, we select a new random subset of features for each classifier.

[1] There are variations on applying ECOC to NN classifiers which do decorrelate errors [3, 44]. We discuss these in Section 6.

As an example, consider Fisher's iris plant classification problem [24, 23]. In this domain, we try to classify iris plants into their specific species: iris-setosa, iris-virginica, and iris-versicolor, based on the following four features: petal length, petal width, sepal length, and sepal width. With MFS we might use three NN classifiers each using a random subset of features. The first NN classifier might use {petal length, sepal width, sepal length}, the second might use {petal width, petal length, sepal width}, and the third might use {petal width, sepal width, sepal width} which we would treat as {petal width, 2 $\times$ sepal width}.

Selecting different feature subsets is an attempt to force the NN classifiers to make different and hopefully uncorrelated errors. Although there is no guarantee that using different feature sets will decorrelate error, Tumer and Ghosh [52] found that with neural networks, selectively removing features could decorrelate errors. Unfortunately, the error rates in the individual classifiers increased, and as a result there was little or no improvement in the ensemble. Cherkauer [14] was more successful, and was able to combine neural networks that used different hand selected features to achieve human expert level performance in identifying volcanoes from images.

Error correlation is related to Breiman's [9] concept of stability in classifiers: One method of generating a diverse ensemble of classifiers is to perturb some aspect of the training inputs for which the classifier is unstable. For example, Bagging [9] perturbs the training patterns available to each classifier in the ensemble. Since decision trees are unstable to the patterns, Bagging generates a diverse and effective ensemble. Nearest neighbor classifiers are stable to the patterns, so Bagging generates poor NN ensembles. Nearest Neighbor classifiers, however, are extremely sensitive to the features used. For example, Langley and Iba [35] found that adding just a few irrelevant features could drastically change the NN classifier's outputs (and reduce accuracy). MFS attempts to use this instability to generate a diverse set of NN classifiers with uncorrelated errors.

As a final agrument for using random feature subsets to generate NN ensembles, we can use Kappa-Error diagrams [17, 36] to compare and contrast random selection of patterns and features. A Kappa-error diagram allows us to directly visualize the diversity and accuracy of an ensemble of classifiers. Each point represents a pair of classifiers in the ensemble: the x coordinate is the value of the Kappa statistic (how much the classifiers agree) and the y coordinate is the average error rate of the pair.

Formally, using Agresti's notation [1], Kappa (κ) is defined as follows: Let π_{ij} be the probability that the two classifiers choose class i and j respectively, then

$$\kappa = \frac{\Pi_o - \Pi_e}{1 - \Pi_e}$$

where $\Pi_o = \sum \pi_{ii}$ is the observed agreement of the two classifiers, and $\Pi_e = \sum \pi_{i+}\pi_{+i}$ is the expected agreement if the classifiers are independent of each other (π_{i+} and π_{+i} are the marginal probabilities of the first and second classifiers picking class i). Kappa is 0 if the classifiers agree as much as expected by chance, and 1 if the classifiers always agree. Diversity increases with smaller Kappa values.

Figure 1 shows the Kappa-Error diagram for NN ensembles generated for the Ionosphere dataset by Bagging, randomly selecting 50 prototypes, and randomly selecting

6 features. Bagging results in a cloud of points centered roughly about (0.825,0.15). Using a smaller number of prototypes (50) for each classifier increases diversity, but also increases error rates. In comparison, using random feature subsets increases diversity, but did not increase the error rates.

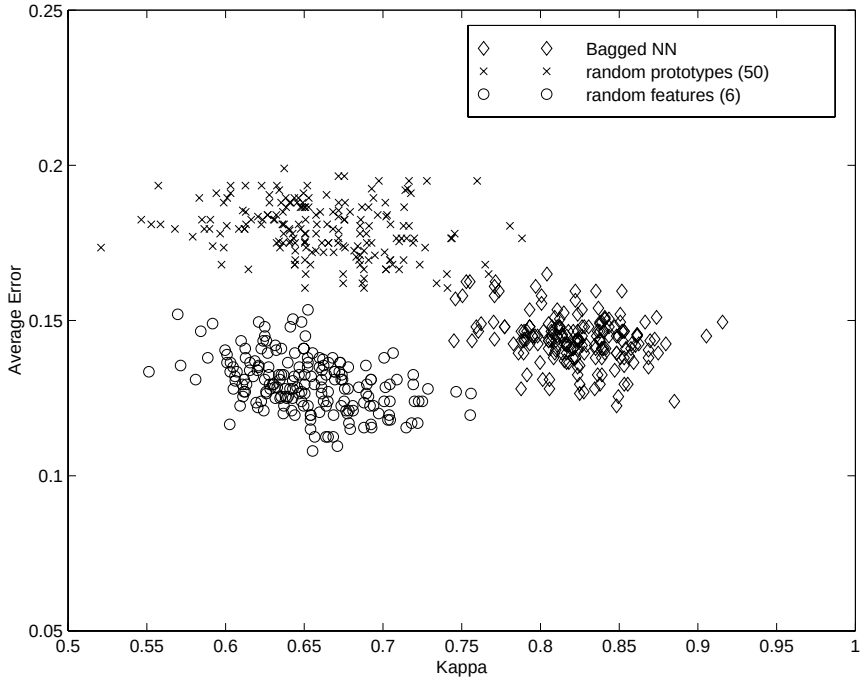

Figure 1: Kappa-Error diagram of NN ensembles generated by Bagging, randomly selecting 50 prototypes, and randomly selecting 6 features on the Ionosphere dataset.

The above discussion hopefully provides motivation for why we expect that MFS will improve the accuracy of the nearest neighbor classifier. However, there are three dangers that we should be aware of when using the MFS:

1. Simple voting can only improve accuracy if the classifiers select the correct class more often than any other class. Breiman refers to this as *order correctness*. If the classifiers are not order correct, then simple voting will increase the expected error. For two class problems, we require slightly more than 50% accuracy in the voting classifiers to improve accuracy. With multiple classes, the required accuracy may drop as low as $\frac{1}{C}$ where C is the number of classes.

2. The Bayes error rate can only increase by using a subset of features. This may make it difficult for the NN classifiers used by MFS to meet the requirements in point 1. For example, in a domain with highly interacting features, such as the parity problem[2], the Bayes error rate in any proper subset of features will be 50% (as opposed to 0% for the full feature space). There is no guarantee

that our subsets will have the necessary information for accurate classification.

3. By using random subsets of features, the ensemble may lose the asymptotic properties of the NN classifier unless we allow the feature selection process to degenerate into selecting all the features. Specifically, as the number of training examples approaches infinity the NN classifier is bounded by twice the Bayes error rate[18]. The kNN classifier is Bayes optimal with proper choice of k [25].

3.1 Parameter Selection

The MFS algorithm has two parameter values that need to be set: the size of the feature subsets, and the number of classifiers to combine.

We set the subset size parameter based on cross-validation accuracy estimates on the training set for the entire ensemble. We evaluated ten evenly spaced intervals over the size of the original feature set. For example, if a domain had 34 features then the subset sizes at 3,7,10,. . . ,34 were evaluated. In the case of ties, the smaller value was chosen.

We set the number of classifiers by evaluating the performance of MFS on seven development datasets varying the number of classifiers from 10 to 1000. Based on the results, we set the number of classifiers to 100 as a reasonable trade-off between computational expense and accuracy.

3.2 Time and Space Complexity

The nearest neighbor classifier is often criticized for slow runtime performance and large memory requirements. Using multiple nearest neighbor classifiers could further worsen the problem. In this section we will analyze the MFS algorithm to find its time and space complexity.

3.2.1 Time Complexity

The time complexity of an algorithm is a measure of how much computation time is needed to run the algorithm. We are concerned with two measures: classification time and training time.

Classification time measures the computational expense in classifying a single pattern. For the NN algorithm, we measure the distance between the test pattern and each of the e examples in the training set (which may be edited to reduce its size [4, 19]). Calculating the distance between two patterns is dependent on the number of features f. This gives a total complexity of $O(ef)$ for NN classification[3]. For the MFS algorithm, we call upon a NN classifier n times, where n is the number of classifiers. We use only s features with $s \leq f$ so the total complexity is $O(nes)$.

[2]In the parity problem we want to classify a binary string as having an even or odd number of 1's

[3]This complexity assumes that the patterns are stored in a linear list or array. Researchers have investigated a number of other approaches to speed up NN classification for a given set of prototypes, including using other data structures [27, 49] and approximate approaches [37].

In addition to classification time, we are also interested in learning or training time. For the MFS algorithm, the only parameter that needs to be learned is an appropriate subset size. We set the subset size by trying several (10) values and estimating the future performance with leave one out cross-validation on the training set. The time to classify each pattern with cross-validation is at most $O(nef)$ since $s \leq f$. We have e total patterns and v values so the the total time is $O(nef(ev)) = O(ne^2fv)$ where v is the number of subset sizes we want to evaluate via cross-validation.

This training complexity is comparable to other learning algorithms: For example, CN2 [16, 15], a rule based system which use beam search requires in the worst case $O(be^2f^2)$ where b is the beam size [16]. ID3, a decision tree algorithm, requires time proportional to the product of the training set size, number of features, and the size of the induced tree [40]. Both these learning times assume that the features are discrete and can become much worse in continuous domains because of the need for repeated sorting operations [12].

3.2.2 Space Complexity

Nearest neighbor methods store a set of prototypes in memory. For methods without editing, this means that all the examples in the training set are stored taking up equivalent memory to the size of the data set. This is equivalent to $O(ef)$.

The MFS algorithm has the same requirements as the NN method – the entire dataset must be stored in memory. However, each of the NN classifiers in MFS does not need its own copy of the dataset and can share a common dataset. MFS then uses exactly the same amount of memory as the NN classifier, regardless of the number of classifiers used. (Recall that the features are selected randomly at runtime.)

In contrast, other multiple classifier systems with n models will usually require n times the memory of a single classifier. For many problems this amount of memory may not be significant, but Dietterich [20] notes that on the Letter Recognition dataset (available from the UCI repository) an ensemble of 200 decision trees obtained 100% accuracy but required 59 megabytes of storage! The entire dataset was only 712 kilobytes.

4 Experiments

Classification algorithms are complex programs that interact with data in many unpredictable ways. It is generally not possible to determine many properties of an algorithm simply by looking at its description. Therefore, we resort to experimental analysis.

4.1 Methods

We evaluated the performance of MFS using two different sampling functions: sampling with replacement (MFS1) and sampling without replacement (MFS2). We compared these to six other algorithms: nearest neighbor (NN), k nearest neighbor

(kNN), nearest neighbor with forward (FSS) and backward (BSS) sequential selection of features [2] and to a decision tree algorithm, C5.0, with (C5.0B) and without Boosting.

All of the NN classifiers used unweighted Euclidean distance for continuous features and Hamming distance for symbolic features. Missing values were treated as informative [50] and considered to be a specific symbolic value. In the case of continuous features, a missing value is considered to have a distance of 1 to all non-missing values. For the kNN classifier, the value of k was set using cross-validation performance estimates on the training set. For feature selection, we used cross-validation accuracy on the training set for our objective function (also known as a wrapper approach [32]). C5.0 was run with the default parameter settings. For C5.0 with Boosting, we used 100 trials.

We evaluated the algorithms on twenty-five datasets from the UCI Repository of Machine Learning Databases [8]. The datasets were chosen to include a wide range of domains and are summarized in Table 1. The first seven datasets were used in the development of the MFS algorithm.

In our experiments, we used 10-fold cross-validation without stratification. For the NN classifiers we scaled the continuous features in the training set of each fold to $[0, 1]$ and then applied the same transformation to the test set.

There were a few exceptions to this procedure: For Waveform, we used ten trials with a random sample of 300 training cases and 4700 test cases to maintain consistency with reported results [41]. For Satimage and Vowel, we used the original division into a training and test set, so the results represent one run of each algorithm.

4.2 Accuracy

The error rates of the classifiers are shown in Table 2. Table 3 presents the pairwise win-loss-tie statistics between the algorithms and indicates which results are statistically significant. The parameter selection results from training are listed in Appendix A.

The results show that MFS is promising: MFS was much better than the single model NN variants and C5.0, both in terms of the number of wins and in the size of the accuracy improvements (which could be as large as 10%). MFS was also competitve with boosted C5.0. Out of 25 domains, both MFS1 and MFS2 each did better than C5.0B in 10 domains and were tied in several others. The differences in the win-loss-tie statistics were not significant under a two-tailed sign test. Finally both MFS1 and MFS2 had a slightly lower error rate than C5.0B averaged over all domains.

The accuracy results indicate that although multiple model approaches can mitigate the base algorithm's biases (by generally improving accuracy), significant differences still show up in the combined models. For example, MFS2 was about 7% more accurate than C5.0B on Sonar; the reverse is true for Automobile.

MFS1 performed poorly on Tic-Tac-Toe, having an error rate many times that of the NN and kNN classifiers. This is probably caused by the high amount of interaction between the features[4]. We need to examine all the features to determine which side has won. Taking a random subset of features does not make sense and would probably

Table 1: Description of datasets: The percentage of missing values is the portion of all feature-value pairs that are missing.

Name	#examples	#classes	#attributes disc	#attributes cont	% missing values
Glass	214	6	-	9	-
Hepatitis	155	2	13	6	5.7
Ionosphere	351	2	-	34	-
Iris	150	3	-	4	-
Liver-Disorders	345	2	-	7	-
Pima Diabetes	768	2	-	8	-
Sonar	208	2	-	60	-
Annealing	898	6	29	9	65.0
Automobile	205	7	10	15	1.2
Breast Cancer	286	2	9	-	0.3
Credit	690	2	9	6	0.6
German	1000	2	7	13	-
Horse-Colic	368	2	15	7	23.8
Labor	57	2	8	8	35.7
Lymphography	148	4	18	-	-
Primary-Tumor	339	21	17	-	3.9
Satimage	4435/2000	6	-	36	-
Segment	2310	7	-	19	-
Soybean	683	19	35	-	9.8
Tic-Tac-Toe	958	2	9	-	-
Vehicle	846	4	-	18	-
Vote	435	2	16	-	5.6
Vowel	528/462	11	-	10	-
Waveform	5000	3	-	21	-
Wine	178	3	-	13	-

lead to a greatly increased Bayes error rate for the individual classifiers. MFS2 did not experience the same degradation as MFS1 because sampling without replacement usually degenerated into selecting all the features and hence performing similarly to NN.

Comparing MFS1 to MFS2, it is not clear which classifier performed better. MFS1 was better than MFS2 on 15 domains, worse on 8, and tied in 2. However, MFS2 had a slightly better average error rate as it did not have a catastrophic failure on Tic-Tac-Toe. The sign test did not detect a significant difference between them.

[4]Tic-tac-toe is a board game played on a 3 by 3 grid. There are two players X and O, who alternate putting X's and O's on the board. In order to win a player must get three X's or O's in a row. The Tic-Tac-Toe dataset contains the 958 legal tic-tac-toe endgame boards (assuming X moves first). The classification task is to determine if X won the game.

Table 2: Classifier error rates.

Domain	NN	kNN	FSS	BSS	MFS1	MFS2	C5.0	C5.0B
Glass	34.1	32.2	30.4	28.0	22.4	22.4	31.8	23.4
Hepatitis	18.7	20.7	18.1	21.3	17.4	18.7	23.9	17.4
Ionosphere	13.1	13.1	12.0	11.1	5.7	6.6	8.8	5.4
Iris	4.7	4.7	6.0	6.7	4.7	5.3	4.7	4.7
Liver-Disorder	35.9	40.0	49.0	38.6	35.4	36.8	35.1	31.6
Pima Diabetes	30.1	27.5	32.2	31.1	28.0	27.5	26.2	25.4
Sonar	16.4	16.4	19.7	15.4	12.5	11.1	25.6	18.4
Annealing	1.2	1.2	0.8	1.1	0.9	0.9	8.9	8.9
Automobile	24.4	24.4	21.5	21.0	21.5	20.0	18.5	13.0
Breast Cancer	29.7	30.4	28.3	29.7	26.6	29.0	25.5	25.5
Credit	19.9	15.2	14.1	20.0	14.8	15.2	15.4	15.2
German	28.3	26.5	29.1	29.9	26.2	25.6	27.5	24.5
Horse-Colic	26.6	19.3	15.2	23.4	16.6	17.4	15.0	15.0
Labor	7.0	5.3	19.3	8.8	5.3	3.5	17.5	14.0
Lymphography	18.9	18.2	25.7	16.9	18.2	16.2	27.0	17.6
Primary-Tumor	60.2	53.4	64.0	62.2	56.3	56.9	57.8	57.8
Satimage	10.5	9.6	12.0	10.6	8.5	9.0	14.6	9.3
Segment	5.8	5.8	3.3	2.6	2.6	2.7	3.5	1.6
Soybean	8.2	8.6	7.6	8.1	6.9	7.5	7.9	7.9
Tic-Tac-Toe	0.9	0.9	9.1	0.9	7.5	1.2	16.6	0.8
Vehicle	30.4	29.1	33.7	27.4	26.8	27.9	29.6	24.0
Vote	7.6	8.1	4.4	5.5	5.3	5.5	3.2	3.2
Vowel	43.7	43.7	52.4	45.9	39.8	39.6	61.5	50.9
Waveform	24.8	18.0	28.3	25.7	18.5	18.8	28.9	17.9
Wine	5.1	2.8	10.1	5.1	1.1	1.7	6.2	2.8
average	20.2	19.0	21.9	19.9	17.2	17.1	21.6	17.4

4.3 Time Complexity

In Section 3.2, we analyzed the time complexity of MFS to show how the requirements would change as a function of the number of examples and features. The analysis, however, does not give any indication of actual running times for datasets on real computers. Therefore, in Table 4 we show the following experimental timing results for the three largest datasets used: (1) the classification time for NN and MFS1, (2) the training times for FSS, BSS, and MFS1, and (3) the combined times for C5.0 and C5.0B. The times for the NN and variants were measured on an 200 MHz Intel Pentium Pro processor; the times for C5.0 and C5.0B were measured on a Sun SPARCserver-1000 and represent both training and classification time. Classification time for kNN, FSS, BSS are not shown here but were almost identical to the NN times [7].

Table 3: Win-Loss-Tie statistics: Each cell lists the number of wins, losses, and ties between the row and column algorithm. Cells in bold indicate differences significant at the 95% confidence level using a two-tailed sign test. Asterisks (*) represents differences that are significant with Bonferroni corrections to control Type I error for multiple tests: i.e. $\alpha = 0.05/28$.

	kNN	FSS	BSS	MFS1	MFS2	C5.0	C5.0B
NN	5-12-8	14-11-0	11-11-3	**1-23-1***	**3-21-1***	10-14-1	**4-20-1***
kNN		14-11-0	12-12-1	**4-18-3**	**4-19-2**	14-10-1	**5-17-3**
FSS			9-16-0	**4-20-1***	**6-19-0**	12-13-0	**3-22-0***
BSS				**3-21-1***	**3-21-1***	13-12-0	**5-20-0**
MFS1					15-8-2	**18-6-1**	10-13-2
MFS2						**18-7-0**	10-14-1
C5.0							**0-18-7***

The results are surprising because MFS was much faster in classification than we expected. MFS uses 100 NN classifiers, and given our serial implementation, we expected MFS to take 100 times as long as the NN classifier. However, in our experiments MFS was only slower by a factor of five. We attribute this speed-up to storing the difference in feature values between the test pattern and all patterns in the training set (i.e. in $d(\mathbf{x}, \mathbf{y}) = (\sum_f (x_f - y_f)^2)^{\frac{1}{2}}$, we calculate $(x_f - y_f)^2$ only once and then use the result for all the classifiers that require it).

Without knowledge of the intended application, we cannot say if the speed of MFS is adequate. However, given these timing results, clearly if one is willing to wait a few seconds for a classification decision, we can feasibly apply MFS to datasets with tens of thousands of patterns.

Table 4: Time Requirements for NN and C5.0 algorithms: Classification time is per pattern; training time is for one trial (using either the given train/test or cross-validation split). Note the time for C5.0 and C5.0B include both the training and classification time, but the classification portion is minimal.

	Classification		Training				
Domain	NN	MFS1	FSS	BSS	MFS1	C5.0	C5.0B
Satimage	0.080s/pat	0.415s/pat	29.6h	12.2h	4.6h	22.8s	42.3m
Segment	0.022s/pat	0.136s/pat	1.3h	1.8h	51.2m	4.9s	8.5m
Annealing	0.026s/pat	0.103s/pat	1.8h	4.1h	15.5m	1.4s	1.1m

4.4 Robustness to Irrelevant features

A major drawback of the NN classifier is its sensitivity to irrelevant features. This concerns us because the MFS algorithm uses multiple NN classifiers and hence raises the question: how will the ensemble behave when the dataset contains many irrelevant features? If the accuracy of the individual NN classifiers drops too low, simple voting can increase the error rate. Since we are unsure of how the ensemble will behave, we experimentally investigated the robustness of MFS to irrelevant features.

We used the same basic procedure in Section 4.1. We added 10, 20, and 30 boolean irrelevant features to each of the datasets and then measured the accuracy of kNN and MFS1.

Table 5 shows the results for five domains; additional results can be found in [7]. We did not directly compare FSS and BSS, because in initial test runs FSS performed similarly to the case where there are no irrelevant features, and BSS performed very badly being unable to overcome its poor starting position (i.e. many irrelevant features).

As expected, irrelevant features always hurt both kNN and MFS to some degree. However, the results are surprising because they reveal that on some domains kNN is critically sensitive while MFS is stable. For example, on the Wine domain, adding 10 irrelevant features drops kNN's accuracy by over 20%, while MFS drops by about 3%. In general, MFS had only minor degradations in accuracy and was occasionally very robust. For example, MFS's accuracy on Ionosphere degrades by so little, it is still better on the dataset corrupted by 30 irrelevant features, than any of the other NN variants on the original uncorrupted dataset.

One possible explanation for MFS's performance lies in how random voters affect the margins of victory in simple voting. For simplicity, let us divide all voters into two types: informed (using relevant features) and uninformed (random) voters. The informed voters cast their ballots, and the winner will have a given margin of votes compared to the next closest competitor. The uninformed, random voters then cast their ballots. The random voters vote with equal probability and equal expectation for all competitors (according to a multinomial distribution). In order for random voting to change the outcome, the number of random votes for class X must meet the following inequality: $randvotes(X) - randvotes(trueclass) > margin(trueclass, X)$. Unless the margins from the informed voters are small, this is unlikely to occur since the $E(randvotes(X)) = E(randvotes(trueclass))$.

As a numerical example, consider a two class problem with fifty informed voters and fifty random voters. The fifty informed voters cast their ballots and the outcome is 30 votes for class A and 20 votes for class B. The fifty uninformed voters then cast their ballots. In order for the uninformed voters to change the outcome of the vote (class A wins) at least 30 must vote for class B. The probability that the decision will change is approximately 8%.

This situation is analogous to what occurs when MFS is applied to domains with irrelevant features. The NN classifiers are the voters, and can become uninformed and random when both of the following conditions are met: (1) the randomly selected features are irrelevant, and (2) the occurrence of the classes in the training set are

roughly equal (this is true in many of the UCI datasets). Note that if only the first condition is met, the NN classifier will be random but will choose classes roughly in proportion to their frequencies in the training set.

Table 5: Error rates of kNN and MFS under corruption by irrelevant features.

	kNN				MFS1			
Domain	0	10	20	30	0	10	20	30
Ionosphere	13.1	22.5	27.9	31.6	5.7	9.4	8.6	9.1
Iris	4.7	22.7	31.3	41.3	4.7	6.0	8.7	17.3
German	26.5	29.2	28.3	30.4	26.2	27.4	28.7	30.4
Vehicle	29.1	60.4	65.8	65.3	26.8	31.1	31.2	33.7
Wine	2.8	23.0	32.6	36.5	1.1	3.9	6.7	10.1

5 Bias-Variance Analysis of Error

The expected error of an algorithm can be divided into two components: *bias* which is the consistent error that the algorithm makes over many different runs, and *variance* which is error that fluctuates from run to run. This decomposition is a useful method for explaining how changes to an algorithm affect the final error rates. It allows us to decompose the error into meaningful components and to see how the error components change with variations in the algorithm.

Several researchers have used the bias-variance analysis error to show how multiple model approaches work. For example, both Breiman [10] and Schapire et al. [45] showed that Bagging improves performance by reducing the variance component of error. Kong and Dietterich [34] showed that ECOC could reduce both bias and variance.

The bias variance decomposition of error originated in squared error for regression. For classification, 0-1 loss (misclassification rate) is commonly used, but this does not have a straightforward or unique decomposition. Recently, many authors have proposed similar decompositions [11, 31, 33, 34, 51].

We used Kong and Dietterich's definitions [34]. They define bias to be "the error of the ideal voted hypothesis," which is the result we would get from combining an infinite number of classifiers, each trained on an independent set of examples. Variance is the "difference between the expected error rate and the ideal voted hypothesis error rate." Formally, where A is the algorithm, m is the training set size, x is the unknown test point, $f(x)$ is the class of x, $f^*(x)$ is the ideal voted hypothesis of the algorithm A at x, and $Error(A, m, x)$ is the expected error of algorithm A at x using training sets of size m, then bias and variance are:

$$Bias(A, m, x) = \begin{cases} 0 & \text{if } f^*(x) = f(x) \\ 1 & \text{if } f^*(x) \neq f(x) \end{cases} \quad (1)$$

$$Variance(A, m, x) = Error(A, m, x) - Bias(A, m, x) \quad (2)$$

Note that the Bayes error is incorporated into the bias error. Also, the variance can be negative. This occurs when the algorithm is usually wrong, but sometimes predicts the correct class.

We investigated the bias-variance components of error on three datasets originally used by Breiman [11] and later by Schapire et. al [45] to evaluate multiple model approaches. The datasets are two class problems, with the individual classes composed of 20-dimensional gaussians (see Appendix B for exact parameter values).

We compared four classifiers: NN, kNN, MFS1 with 1 classifier (1-MFS1), and MFS1 with 100 classifiers. The NN classifier is the control, to which we can compare the kNN and MFS algorithms. 1-MFS1 should allow us to determine the changes to the error components that are caused by random feature selection and the changes that are caused by voting among multiple classifiers.

We used a test set of 3000 instances and 100 independent training sets of size 300 to estimate the bias, variance, and error of the four classifiers. We approximated $f^*(x)$ by voting over the classifiers trained on the 100 independent training sets. The results are shown in Table 6.

Table 6: Bias-Variance decomposition of error.

Domain	Optimal	NN	1-MFS1	MFS1	kNN
Twonorm					
bias	2.3	2.4	2.6	2.4	2.4
variance	-	4.9	17.8	1.3	1.0
error	2.3	7.3	20.4	3.7	3.4
Threenorm					
bias	10.5	10.5	11.6	10.4[5]	11.2
variance	-	13.6	22.5	6.3	4.4
error	10.5	24.1	34.1	16.8	15.6
Ringnorm					
bias	1.3	47.1	4.6	3.7	47.1
variance	-	-7.9	25.8	2.0	-7.9
error	1.3	39.2	30.4	5.7	39.2

In Twonorm and Threenorm, selecting a single random subset of features (1-MFS1) destabilizes the NN classifier and causes the variance error to significantly increase. During voting (MFS1) the variance error is reduced to a much smaller value than the variance of the original NN classifier, thus reducing the overall error significantly.

[5]The value for bias should always be greater than or equal to the Bayes error rate (10.5%), however, because of estimation error from finite sample sizes, it is possible to obtain bias estimates which are lower than the optimal bound.

For Ringnorm, the feature selection process does a dramatic trade of bias for variance. The bias error drops from 47.1% to only 4.6%, while the variance increases from -7.9% to 25.8%. Voting then drops the variance to only 2% greatly improving accuracy.

From these datasets, we see that MFS has two modes of operation: (1) decreasing variance through voting, and (2) trading bias for variance through random feature selection. Taken together, MFS is able to reduce both bias and variance components of error.

In comparison to MFS, the kNN classifier reduced only variance. On Twonorm and Threenorm the error of NN was dominated by variance (the bias error was nearly optimal) and like MFS, kNN was able decrease error by reducing the variance. In fact, kNN did a better job than MFS at variance reduction. On Ringnorm, the error of the NN classifier was dominated by bias and kNN was not able to improve performance.

6 Related Work

Although there is a large body of research on multiple model methods for classification, very little specifically deals with combining NN classifiers. We are only aware of Skalak's [47, 48] work on combining NN classifiers with small prototype sets, Alpaydin's [6] work with condensed nearest neighbor (CNN) classifiers [29], and Aha and Bankert's [3] initial work on combining NN, feature selection, and ECOC which was later expanded on by Ricci and Aha [44].

Skalak and Alpaydin approach the problem of combining NN classifiers similarly. They drastically reduce the size of each classifier's prototype set to destabilize the NN classifier. Skalak investigated several different strategies for finding a reduced prototype set and even pursued an approach called "radical destabilization" where the NN classifier has just a single prototype per class. He was able to improve accuracy over the baseline NN classifier in 10 of 13 UCI domains. Alpaydin used dataset partitioning (bootstrap or disjoint) in combination with the CNN classifier to edit and reduce the prototypes. He also reported improvements (on five domains from the UCI repository and one optical character recognition dataset) over the baseline NN classifier if the training sets were sufficiently small and thus able to generate diverse classifiers.

It is important to note that both of Alpaydin's and Skalak's work differ significantly from this article with respect to their performance goals. Their goals are to both increase classification accuracy and reduce computation time (by reducing the number of prototypes for each classifier), whereas MFS concentrates on improving accuracy only. Accuracy and efficiency are usually conflicting goals for nearest neighbor classifiers, since accuracy improves by increasing the number of prototypes, while efficiency improves by decreasing the number of prototypes.

Ricci and Aha [43, 42] (following [3]) applied ECOC to the NN classifier (NN-ECOC). Normally, applying ECOC to NN would not work as the errors in the two-class problems would be perfectly correlated; however, they found that applying feature selection to the two-class problems decorrelated errors *if different features were*

selected. With this method they were able to improve performance on 7 of 10 domains tested (7 from the UCI repository and 3 proprietary cloud classification datasets), and they noted that ECOC accuracy gains tended to increase with increased diversity among the features selected for the two-class problems.

NN-ECOC is similar to MFS as they both use NN classifiers with different features. They differ in that NN-ECOC uses active selection of features (and output coding) while MFS uses random selection. Ricci and Aha also analyzed NN-ECOC for bias and variance and concluded that NN-ECOC reduces bias but slightly increases variance. Unfortunately, because we used different a definition of bias and variance our results are not directly comparable.

Regardless of which method has better accuracy, MFS appears to have two main advantages over NN-ECOC: (1) MFS is the simpler algorithm, and (2) MFS is not constrained by ECOC to multiclass problems.

As a final note, Ho [30] independently proposed using random subsets of features to create ensembles of decision trees. She reported that this method worked best when the dataset had a large number of features and samples, and worked poorly when the dataset had few features combined with a small number of samples or a large number of classes. In contrast, for MFS, there was no clear pattern linking dataset characteristics to performance gains.

7 Conclusions and Future Directions

In this article, we introduced MFS, an algorithm for combining multiple NN classifiers. In MFS, each NN classifier has access to all the patterns in the original training set, but only to a random subset of the features.

Our experiments showed that MFS was effective in improving accuracy, and was even competitive with boosted decision trees, the current state of the art. But beyond accuracy improvements, MFS is a significant advance because it allows us to incorporate many desirable properties of the NN classifier in a multiple model framework. For example, one of the primary advantages of the NN classifier is its ability to incrementally add new data (or remove old data) without requiring retraining. MFS maintains this property and new data can be added (old data removed) at runtime. Another useful property of the NN classifier is its ability to predict directly from the training data without using intermediate structures. As a result, no matter how many classifiers we combine in MFS, we require only the same memory as a single NN classifier.

MFS has disadvantages and it should not be used indiscriminantly. In particular, on domains with highly interacting features, such as Tic-Tac-Toe, the error rate can increase too much in the feature subsets resulting in poor ensemble performance. MFS also requires training whereas the NN classifier does not. As with all multiple model approaches, we lose comprehensibility compared to a single model. The individual must judge if the potential accuracy increases is worth these disadvantages.

The analysis of MFS in this paper suggests several directions for future work:

1. *Why does using random feature subsets work?* We made an initial attempt at

answering this question with our analysis of irrelevant features and the bias-variance decomposition of error. But clearly more work needs to be done to characterize the domains for which MFS is appropriate.

2. *Implications for feature selection and feature weighting.* The experimental results showed that combining multiple random feature subsets can significantly improve performance over the single best subset of features found by FSS or BSS. This implies that instead of searching for the single best set of features, we should be searching for multiple feature sets that work well together.

3. *Accuracy Improvements.* MFS is a simple combining method, and if possible we would like to improve on it further. We investigated several alternate approaches to improve accuracy including: (1) allowing different number of features for each classifier, (2) weighting the votes by a function of accuracy, and (3) post pruning the ensemble to remove poor classifiers. However, initial experiments on the development datasets did not indicate any accuracy advantages.

4. *Speedup Improvements.* In this paper, we concentrated on improving the accuracy of the NN classifier and did not pay much attention to the computational requirements. An open research area is how can we speed up ensembles of NN classifiers? Some obvious techniques to investigate include editing the prototypes and postpruning the ensemble. Another approach is to take advantage of the reduced dimensionality of the feature set used for the individual NN classifiers by using structures like KD-trees. KD-trees offer fast $O(\log n)$ lookup for the nearest neighbor, but only for low dimensional (say $|F| < 10$) datasets [38]. Our results indicate that in many domains we can obtain accurate results by combining NN classifiers using very few features (on average MFS1 used 10.6 features, MFS2 used 8.3). This means that instead of using the NN classifier with all the features, we can use m NN classifiers, each with only a small number of features, and potentially obtain $O(m \log n)$ classification speed while improving accuracy.

Acknowledgements

I thank Michael Pazzani for his support and encouragement. I also thank Cathy Blake, Yang Wang, and the anonymous reviewers for providing many comments that improved this paper. This article represents research conducted both at the University of California, Irvine, and at the University of Waterloo, Department of Systems Design Engineering. It was partially supported by an NSERC PGS A scholarship.

Appendix A: Parameter Selection Results

Table 7: Parameter selection results: average k or number of features selected.

Domain	Average Parameter Settings				
	kNN	FSS	BSS	MFS1	MFS2
Glass	2.0	5.6	5.7	4.3	3.5
Hepatitis	7.4	2.3	14.5	10.8	8.6
Ionosphere	1.0	4.8	21.6	7.0	6.9
Iris	8.6	1.4	2.3	2.8	3.4
Liver-Disorders	9.4	1.0	4.7	4.0	3.2
Pima Diabetes	11.8	1.1	7.2	4.5	4.2
Sonar	1.0	8.1	47.1	15.6	13.8
Annealing	1.0	8.2	8.6	32.1	22.6
Automobile	1.0	3.6	8.7	8.2	8.0
Breast Cancer	10.8	2.9	5.8	6.4	5.5
Credit	8.6	4.1	11.2	8.2	7.1
German	10.4	3.6	18.9	14.6	9.0
Horse Colic	12.6	3.6	13.5	9.6	7.6
Labor	2.8	2.7	7.5	5.8	6.2
Lymphography	6.2	4.4	13.8	14.8	9.9
Primary-Tumor	16.0	6.0	12.0	10.1	8.4
Satimage	3	10	33	14	11
Segment	1.0	4.9	10.7	10.6	7.8
Soybean-Large	1.8	13.9	22.5	26.4	18.1
Tic-Tac-Toe	1.0	7.2	9.0	9.0	8.9
Vehicle	5.0	6.6	12.5	11.5	6.1
Vote	4.4	4.2	9.3	12.1	9.1
Vowel	1	7	7	8	6
Waveform	14.4	7.2	17.0	10.4	7.9
Wine	8.8	4.4	7.7	3.9	4.1
average	6.0	5.2	13.3	10.6	8.3

Appendix B: Parameters for Bias-Variance Datasets

Twonorm : $\sigma_1 = \sigma_2 = I, \mu_1 = (a, a, \ldots, a)$, $\mu_2 = (-a, -a, \ldots, -a)$ with $a = 2/\sqrt{20}$, $P(class1) = P(class2) = 0.5$.

Threenorm : $\sigma_1 = \sigma_2 = \sigma_3 = I$, $\mu_1 = (a, a, \ldots, a)$,$\mu_2 = (-a, -a, \ldots, -a)$, $\mu_3 = (a, -a, a, -a, \ldots, a, -a)$ with $a = 2/\sqrt{20}$. Class 1 is drawn with equal probability from $N_1 \sim (\mu_1, \sigma_1)$ and $N_2 \sim (\mu_2, \sigma_2)$. $P(class1) = P(class2) = 0.5$.

Ringnorm : $\sigma_1 = 4I$, $\sigma_2 = I$, $\mu_1 = (0, 0, \ldots, 0)$, $\mu_2 = (a, a, \ldots, a)$ with $a = 1/\sqrt{20}$, $P(class1) = P(class2) = 0.5$.

References

[1] Agresti, A., *Categorical Data Analysis*, John Wiley & Sons, 1990.

[2] Aha, D. W., and Bankert, R. L., Feature selection for case-based classification of cloud types: An empirical comparison, In *Proceedings of the AAAI-94 Workshop on Case-Based Reasoning*, 106–112, 1994.

[3] Aha, D. W., and Bankert, R. L., Cloud classification using error-correcting output codes, *Artificial Intelligence Applications: Natural Resources, Agriculture, and Environmental Science 11*, 1, 13–28, 1997.

[4] Aha, D. W., Kibler, D., and Albert, M. K., Instance-based learning algorithms, *Machine Learning 6*, 37–66, 1991.

[5] Ali, K. M., and Pazzani, M. J., Error reduction through learning multiple descriptions, *Machine Learning 24*, 173–202, 1996.

[6] Alpaydin, E., Voting over multiple condensed nearest neighbors, *Artificial Intelligence Review 11*, 1-5, 115–132, 1997.

[7] Bay, S. D., Nearest neighbour classification from multiple data representations, Master's thesis, University of Waterloo, Department of Systems Design Engineering, 1997.

[8] Blake, C., Keogh, E., and Merz, C. J., UCI repository of machine learning databases, 1998.

[9] Breiman, L., Bagging predictors, *Machine Learning 24*, 123–140, 1996.

[10] Breiman, L., Bias, variance, and arcing classifiers, Tech. Rep. 460, Statistics Department, University of California, Berkeley, 1996.

[11] Breiman, L., Heuristics of instability and stabilizations in model selection, *The Annals of Statistics 24*, 6, 2350–2383, 1996.

[12] Catlett, J., *Megainduction: machine learning on very large databases*, PhD thesis, Faculty of Arts, University of Sydney, 1991.

[13] Chan, P., and Stolfo, S., A comparative evaluation of voting and meta-learning on partitioned data, In *Proceedings of the Twelfth International Conference on Machine Learning*, 90–98, 1995.

[14] Cherkauer, K. J., Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks, In *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*, P. Chan, Ed., 15–21, 1996. Available from http://www.cs.fit.edu/∼imlm.

[15] Clark, P., and Boswell, R., Rule induction with CN2: Some recent improvements, In *Machine Learning - Proceedings of the Fifth European Conference*, 151–163, 1991.

[16] Clark, P., and Niblett, T., The CN2 induction algorithm, *Machine Learning 3*, 261–283, 1989.

[17] Cohen, J., A coefficient of agreement for nominal scales, *Educational and Psychological Measurement 20*, 1, 37–46, 1960.

[18] Cover, T. M., and Hart, P. E., Nearest neighbor pattern classification, *IEEE Transactions on Information Theory 13*, 1, 21–27, 1967.

[19] Dasarathy, B. V., *Nearest Neighbor, NN Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, CA., 1991.

[20] Dietterich, T. G., Machine learning research: Four current directions, *AI Magazine 18*, 4, 97–136, 1997.

[21] Dietterich, T. G., An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization, *submitted to Machine Learning*, 1998.

[22] Dietterich, T. G., and Bakiri, G., Solving multiclass learning problems via error-correcting output codes, *Journal of Artificial Intelligence Research 2*, 263–286, 1995.

[23] Duda, R. O., and Hart, P. E., *Pattern Classification and Scene Analysis*, John Wiley, New York, 1973.

[24] Fisher, R. A., The use of multiple measurements in taxonomic problems, *Annual Eugenics 7*, 179–188, 1936.

[25] Fix, E., and Hodges, J. L., Discriminatory analysis: Nonparametric discrimination: Consistency properties, Tech. Rep. Project 21-49-004, Report Number 4, USAF School of Aviation Medicine, Randolf Field, Texas, 1951.

[26] Freund, Y., and Schapire, R. E., Experiments with a new boosting algorithm, In *Machine Learning: Proceedings of the Thirteenth International Conference*, pp. 325–332, 1996.

[27] Friedman, J. H., Bentley, J. L., and Finkel, R. A., An algorithm for finding best matches in logarithmic expected time, *ACM Trans. Math. Software 3*, 209–226, 1977.

[28] Hansen, L. K., and Salamon, P., Neural network ensembles, *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 993–1001, 1990.

[29] Hart, P. E., The condensed nearest neighbor rule, *IEEE Transactions on Information Theory 14*, 515–516, 1968.

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

[31] James, G., and Hastie, T., Generalizations of the bias/variance decomposition for prediction error, http://stat.stanford.edu/~gareth, 1997.

[32] Kohavi, R., and John, G. H., Wrappers for feature subset selection, *Artificial Intelligence 97*, 1-2, 273–324, 1996.

[33] Kohavi, R., and Wolpert, D. H., Bias plus variance decomposition for zero-one loss functions, In *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996.

[34] Kong, E. B., and Dietterich, T. G., Error-correcting output coding corrects bias and variance, In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 725–730, 1996.

[35] Langley, P., and Iba, W., Average-case analysis of a nearest neighbor algorithm, In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, 889–894, 1993.

[36] Margineantu, D. D., and Dietterich, T. G., Pruning adaptive boosting, In *Proceedings of the 14th International Conference on Machine Learning*, 1997.

[37] Miclet, L., and Dabouz, M., Approximative fast nearest-neighbour recognition, *Pattern Recognition Letters 1*, 277–285, 1983.

[38] Moore, A. W., *Efficient Memory-based Learning for Robot Control*, PhD thesis, University of Cambridge, 1991.

[39] Poggio, T., and Girosi, F., A theory of networks for approximation and learning, Tech. Rep. AI Memo 1164, MIT Artificial Intelligence Laboratory, Cambridge, MA., 1989.

[40] Quinlan, J. R., Induction of decision trees, *Machine Learning 1*, 81–106, 1986.

[41] Quinlan, J. R., Bagging, Boosting, and C4.5, In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 725–730, 1996.

[42] Ricci, F., and Aha, D. W., Bias, variance, and error correcting output codes for local learners, Tech. Rep. AIC-97-025, Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, 1997.

[43] Ricci, F., and Aha, D. W., Extending local learners with error-correcting output codes, Tech. Rep. AIC-97-001, Navy Center for Applied Research in Artificial Intelligence, 1997.

[44] Ricci, F., and Aha, D. W., Error-correcting output codes for local learners, In *Proceedings of the 10th European Conference on Machine Learning*, 1998.

[45] Schapire, R. E., Freund, Y., Bartlett, P., and Lee, W. S., Boosting the margin: A new explanation for the effectiveness of voting methods, In *Machine Learning: Proceedings of the Fourteenth International Conference*, 1997.

[46] Sejnowski, T. J., and Rosenberg, C. R., Parallel networks that learn to pronounce English text, *Complex Systems 1*, 145–168, 1987.

[47] Skalak, D. B., *Prototype Selection for Composite Nearest Neighbor Classifiers*, PhD thesis, Department of Computer Science, University of Massachusetts, 1996.

[48] Skalak, D. B., The sources of increased accuracy for two proposed boosting algorithms, In *AAAI '96 Workshop on Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms*, 1996.

[49] Sproull, R. F., Refinements to nearest-neighbour searching in k-dimensional trees, *Algorithmica 6*, 579–589, 1991.

[50] Tabachnick, B. G., and Fidell, L. S., *Using Multivariate Statistics, Second Edition*, Harper & Row, 1989.

[51] Tibshirani, R., Bias, variance and prediction error for classification rules, Tech. rep., Department of Statistics, University of Toronto, 1996.

[52] Tumer, K., and Ghosh, J., Error correlation and error reduction in ensemble classifiers, *Connection Science 8*, 385–404, 1996.

[53] Wettschereck, D., and Dietterich, T. G., Improving the performance of radial basis function networks by learning center locations, In *Nerual Information Processing Systems*, J. Moody, S. Hanson, and R. Lippmann, Eds., vol. 4. Morgan Kaufmann, 1992.