Combining Nearest Neighbor Classifiers Through Multiple Feature Subsets

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

Abstract

Combining multiple classifiers is an effective technique for improving accuracy. There are many general combining algorithms, such as Bagging or Error Correcting Output Coding, that significantly improve classifiers like decision trees, rule learners, or neural networks. Unfortunately, many 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 improved upon the NN, k nearest neighbor (kNN), and NN classifiers with forward and backward selection of features. MFS was also robust to corruption by irrelevant features compared to the kNN classifier. Finally, we show that MFS is able to reduce both bias and variance components of error.

1 INTRODUCTION

The nearest neighbor (NN) classifier is one of the oldest and simplest methods for performing general, nonparametric 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 NN 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 often gives competitive performance with more modern methods such as decision trees or neural networks.

Since its inception by Fix and Hodge (1951), 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 (Dasarathy, 1991). 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, in Breiman's (1996) experiments with Bagging, he found no difference in accuracy between the bagged NN classifier and the single model approach. His results suggest that other combining methods that involve any significant degree of resampling or replication of patterns will not work with the NN classifier. Kong and Dietterich (1996) also concluded that Error Correcting Output Coding (ECOC), a method of combining classifiers by decomposing multi-class problems into multiple two-class problems, will not improve classifiers that use local information because of high error correlation. For example, with the NN classifier we predict the class of the closest pattern. This pattern is the same in all of the two-class problems, and hence if it gives an incorrect prediction, all the predictions in the ECOC ensemble will be in-

*Research performed while at the University of Waterloo, Department of Systems Design Engineering, Waterloo, Ont., N2L 3G1, Canada.

correct [1].

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

In the next section, we describe the MFS algorithm for combining multiple NN classifiers. In Section 3, we evaluate the algorithm on datasets from the UCI Repository for accuracy, computational complexity, and robustness to irrelevant features. In Section 4, we analyze the algorithm's bias and variance components of error. In Section 5, we discuss related work, and follow it by conclusions and future work in Section 6.

2 CLASSIFICATION FROM MULTIPLE FEATURE SUBSETS

We start by describing the MFS algorithm and then we discuss the motivation behind it and the dangers in using it. We then explain how we set the algorithm's parameters.

2.1 THE MFS ALGORITHM

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. 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 is equivalent to 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 a tuning dataset (see Section 2.2). Each time a pattern is presented for classification, we select a new random subset of features for each classifier.

As an example of MFS classification, consider Fisher's iris plant classification problem (Fisher, 1936; Duda and Hart, 1973). 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}.

The idea of using only a random subset of features may seem counter intuitive, as we are throwing away potentially valuable information. The accuracy of the NN classifiers is likely to decrease compared to a classifier that has access to all the features. Should we not use all the information and make each classifier as accurate as possible? Why should we create a set of classifiers each less accurate than a single one trained on all the information?

The answer to these questions lies in the dynamics of simple voting among a set of classifiers. The individual models do not need to be very accurate for the system as a whole to achieve high accuracy, if the models make different errors. In particular, Hansen and Salamon (1990) showed that under simple voting if the models make independent errors, then the overall error will decrease monotonically with increasing numbers of classifiers. Ali and Pazzani (1996) verified empirically that combining models with uncorrelated errors could significantly reduce the overall error. Selecting different features is an attempt to force the NN classifiers to make different and uncorrelated errors. We are trading off accuracy for error diversity.

There is no guarantee that using different feature sets for the NN classifiers will decorrelate error. However, Tumer and Ghosh (1996) 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 (1996) 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.

[1] Recently Ricci and Aha (1998) have developed a method for combining NN classifiers and ECOC which solves the correlation problem. We discuss this in section 5.

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 (Breiman, 1996) 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 (1993) 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.

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 major dangers that we should be aware of when using 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 the parity problem, a domain with highly interacting features, the Bayes error rate in any proper subset of features is 50% (as opposed to 0% for the full feature space). There is no guarantee that random subsets will have the necessary information for accurate classification.

3. By using the nearest neighbor classifier in the MFS scheme we lose its asymptotic optimality properties. Specifically, as the number of training examples approaches infinity the NN classifier is bounded by twice the Bayes error rate (Cover, 1967). The kNN classifier is Bayes optimal in the limit with proper choice of k (Fix and Hodges, 1951). We can make no such claims about MFS.

2.2 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 MFS's 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 EXPERIMENTS

3.1 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 four other algorithms: nearest neighbor (NN), k nearest neighbor (kNN), nearest neighbor with forward (FSS) and backward (BSS) sequential selection of features (Aha and Bankert, 1994).

The use of FSS and BSS should provide an interesting contrast with MFS. FSS and BSS try to find a single good subset of features, while MFS uses multiple random subsets without regard to their performance.

All classifiers used unweighted Euclidean distance for continuous features and Hamming distance for symbolic features. Missing values were treated as informative and considered to be a specific symbolic value. In the case of continuous features (normalized to [0,1]), 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 (Kohavi and John, 1996)).

We evaluated the algorithms on twenty-five datasets from the UCI Repository of Machine Learning Databases (Merz and Murphy, 1998). We first normalized the datasets so that continuous features ranged

from $[0, 1]$, and then we ran thirty trials where the training set contained 2/3 of the patterns (randomly selected) and the test set contained the remaining 1/3.

There were a few exceptions to this procedure. For Waveform, we used 300 training cases and 4700 test cases to maintain consistency with reported results (Quinlan, 1996). For Satimage, we used the original division into a training and test set, so the results represent one run of each algorithm. For the Musk dataset, which has 166 features, FSS and BSS took too long to run (over 24 hours for a single trial) and no results were obtained.

3.2 ACCURACY

The accuracy and parameter selection results (average k or number of features selected) are shown in Table 1. The first seven datasets were used in the development of the MFS algorithm. The default accuracy is the frequency of the most common class.

The results show that MFS is promising: MFS1 and MFS2 were about 2% more accurate over all domains than it's nearest competitor kNN. MFS1 was best on 16 domains out of 25 (not including MFS2). MFS2 was best on 14 domains and tied in 3 (not including MFS1). For a formal comparison, we used the Wilcoxon signed rank test and found that MFS1 and MFS2 were significantly better than all others with a confidence level greater than 99%.

MFS only performed poorly on two datasets: Iris and Tic-Tac-Toe. For Iris, both MFS1 and MFS2 gave the lowest accuracy out of all the classifiers. This can possibly be explained by the small number of features in the Iris dataset. With only four features, many of the feature subsets would be identical. This would lead to identical errors and high error correlation. For Tic-Tac-Toe, MFS1 performed extremely poorly, having an error rate almost five times that of the NN and kNN classifiers. MFS1 probably performed poorly because in the Tic-Tac-Toe domain the features have a high amount of interaction. 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 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 degenerated into selecting all the features and hence performing identically to NN.

Comparing MFS1 to MFS2, it is not clear which classifier performed better. MFS1 was better than MFS2 on 15 domains, worse on 7, and tied in 3. However, MFS2 had a slightly better average accuracy as it did not have a catastrophic failure on Tic-Tac-Toe. The Wilcoxon test did not detect a significant difference between them.

3.3 COMPUTATIONAL COMPLEXITY

The nearest neighbor classifier is often criticized for slow runtime performance, so we will briefly comment on the complexity of MFS and then present actual running times from the experiments.

The NN classifier computes the distance between the test pattern and every pattern in the training set. This requires $O(ef)$ time, where e is the number of examples, and f is the number of features. For MFS, we use n NN classifiers, so its complexity is $O(nef)$. For training, we use cross-validation and MFS requires $O(ne^2fv)$ time, where v is the number of folds (Bay, 1997).

This analysis shows how the computational requirements of MFS change as a function of the number of examples and features. However, it does not give any indication of actual running times on real datasets. Therefore in Table 2 we list the actual running times on an Intel Pentium Pro processor for NN and MFS on the three slowest datasets.

Table 2: Time Requirements for NN and MFS1

	Classification		Training
Domain	NN	MFS1	MFS1
Satimage	0.080s/pat	0.415s/pat	4.6h
Segment	0.015s/pat	0.075s/pat	19.9m
Annealing	0.018s/pat	0.073s/pat	5.5m

Note that even though we are combining 100 classifiers in MFS, it was only about five times as slow as the NN classifier. We attribute this speed up to caching 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 cache $(x_f - y_f)^2$).

3.4 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? If the accuracy of the individual NN classifiers drops too low, simple voting can increase the error rate. Since

Table 1: Accuracy and Parameter Selection Results (average k or number of features selected)

		Accuracy							Average Parameter Settings				
Domain	Pat/F	Def.	NN	kNN	FSS	BSS	MFS1	MFS2	kNN	FSS	BSS	MFS1	MFS2
Glass	214/9	35.5	67.9	66.8	72.3	72.5	75.8	**76.1**	1.7	4.8	5.5	4.4	3.6
Hepatitis	155/19	79.4	79.2	80.4	80.3	77.2	**82.7**	82.6	6.7	2.4	12.8	8.1	7.0
Ionosphere	351/34	64.1	86.5	85.5	88.2	87.9	**93.5**	92.7	1.8	4.6	21.9	6.9	6.5
Iris	150/4	33.3	94.3	**95.1**	93.7	93.5	92.5	92.7	6.1	1.4	2.3	2.8	2.8
Liver-Disorders	345/7	58.0	60.4	61.3	56.8	60.0	**65.4**	64.4	9.7	1.9	4.2	4.1	3.2
Pima Diabetes	768/8	65.1	69.7	**73.6**	67.7	68.5	72.5	72.3	11.5	2.0	6.5	4.8	4.2
Sonar	208/60	53.4	85.0	85.1	76.0	84.3	**87.3**	87.0	1.1	6.3	38.2	15.4	13.2
Annealing	898/38	76.2	98.0	98.0	**98.8**	**98.8**	98.6	98.6	1.0	8.2	9.0	31.6	21.3
Automobile	205/25	32.7	70.9	70.9	**74.2**	72.8	72.5	73.3	1.0	3.3	10.3	8.7	6.3
Breast Cancer	286/9	70.3	65.9	**74.3**	71.0	70.0	74.0	74.0	8.0	1.9	5.0	6.7	4.6
Credit	690/15	55.5	81.6	85.5	85.7	81.6	**86.3**	85.8	12.4	3.2	10.5	8.8	6.3
German	1000/20	70.0	70.5	73.1	70.6	68.8	**74.4**	74.2	10.8	3.0	15.7	15.4	11.2
Horse Colic	368/22	63.0	76.8	79.8	**83.9**	76.5	80.2	79.8	15.1	2.4	14.8	9.8	7.8
Labor	57/16	64.9	92.1	90.4	78.6	89.5	94.2	**94.6**	2.3	2.8	7.5	6.7	5.1
Lymphography	148/18	54.7	74.6	77.0	74.8	76.7	**81.9**	80.4	8.7	3.7	12.1	11.6	8.3
Musk	476/166	56.5	84.3	83.9	na	na	**88.9**	88.6	1.4	na	na	18.1	19.1
Primary-Tumor	339/17	24.5	37.0	43.5	37.8	38.9	44.5	**45.0**	13.8	6.3	11.2	10.6	8.1
Satimage	6435/36	22.8	89.5	90.4	88.0	89.4	**91.5**	91.0	3	10	33	14	11
Segment	2310/19	14.3	93.5	93.0	96.5	96.6	**96.8**	96.6	4.6	4.8	9.9	10.3	7.9
Soybean-Large	683/35	13.0	90.7	90.5	93.2	90.7	**93.4**	93.2	1.5	11.9	20.2	21.9	14.9
Tic-Tac-Toe	958/9	65.3	**98.1**	**98.1**	87.8	**98.1**	91.1	**98.1**	1.0	6.6	9.0	9.0	9.0
Vehicle	946/18	25.8	68.1	67.7	66.6	70.4	**71.4**	**71.4**	5.7	5.4	12.5	9.7	6.8
Vote	435/16	54.8	92.9	93.1	**95.8**	94.6	94.9	94.5	4.3	2.8	9.2	11.8	8.4
Waveform	5000/21	33.9	74.9	**81.4**	70.3	74.4	81.0	80.9	13.7	7.4	16.8	10.0	8.1
Wine	178/13	39.9	95.2	96.7	92.8	94.8	97.6	**97.9**	9.8	4.1	7.8	3.8	3.5
average		49.1	79.9	81.4	79.2	80.3	83.3	**83.4**	6.3	4.6	12.8	10.6	8.3

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 3.1. We added 10, 20, and 30 boolean irrelevant features to each of the datasets and then measured the accuracy of kNN and MFS1. We chose boolean irrelevant features because they are more difficult for nearest neighbor methods to handle than continuous irrelevant features. This is because while they both have the same range and mean, boolean variables have greater variance.

Table 3 shows the results for several domains. The remaining results (Bay, 1997) are not shown here for space reasons, but they follow a similar pattern.

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 Vehicle and Wine with 10 added irrelevant features, kNN drops in accuracy by over 20% while MFS drops by less than 2%. 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 (from 93.5% to 90.1%), it is still better on the dataset corrupted by 30 irrelevant features, than all of the other classifiers on the original 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.

4 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 of error to show how multiple model approaches work. For example, both Breiman (1996b) and Schapire et al. (1997) showed that Bagging improves performance by reducing the variance component of error. Kong and Dietterich (1996) 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 (Kong and Dietterich, 1996; Breiman, 1996b; James and Hastie, 1997; Tibshirani, 1996; Kohavi and Wolpert, 1996).

We used Kong and Dietterich's (1996) definitions. 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 makes a lucky guess and predicts the correct class.

We investigated the bias-variance components of error on three datasets originally used by Breiman (1996b) and later by Schapire et. al (1997) to evaluate multiple model approaches. The datasets are two class problems, with the individual classes composed of 20-dimensional gaussians.

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 4.

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.

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

Table 3: Accuracy of kNN and MFS Under Corruption by Irrelevant Features

	kNN				MFS1			
Domain	0	10	20	30	0	10	20	30
Breast Cancer	74.3	71.0	70.3	69.8	74.0	71.5	71.3	70.5
German	73.1	72.0	70.9	70.5	74.4	72.6	71.3	70.7
Ionosphere	85.5	73.7	71.7	69.5	93.5	91.3	91.4	90.1
Soybean-Large	90.5	80.6	75.2	71.1	93.4	87.7	81.2	76.9
Vehicle	67.7	37.8	35.5	34.1	71.4	69.7	66.0	64.2
Vote	93.1	91.8	91.1	90.9	94.9	93.0	92.0	91.3
Wine	96.7	72.5	62.2	61.2	97.6	96.9	93.7	91.8

Table 4: Bias Variance Decomposition of Error

Domain	Opt.	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[2]	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

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.

[2]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.

5 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 (1996) work on combining NN classifiers with small prototype sets, Alpaydin's (1997) work with condensed nearest neighbor (CNN) classifiers (Hart, 1968), and Ricci and Aha's (1998) work on combining NN, feature selection, and ECOC.

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 investigates several different strategies for finding a reduced prototype set and even pursues 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. Interestingly, MFS did well on Glass and Lymphography (average increase of over 7% compared to the NN classifier); these are two domains where Skalak reported that no combining algorithm improved performance. Alpaydin uses dataset partitioning (bootstrap or disjoint) in combination with the CNN classifier to edit and reduce the prototypes. He also reported improvements over the NN classifier if the training sets were sufficiently small and thus able to generate diverse classifiers.

Ricci and Aha (1998) 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 highly 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 in many of the domains tested, and they noted that ECOC accuracy gains tended to increase with in-

creased 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. A head to head comparison would be useful to determine if NN-ECOC and MFS achieve their accuracy gains in the same areas of the feature space. 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.

6 CONCLUSIONS AND FUTURE WORK

We introduced MFS, a new 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. 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. (The combined NN classifiers can share a common dataset, and the features are selected randomly at runtime.)

MFS has disadvantages and it should not be used indiscriminantly. In particular, MFS loses the asymptotic optimality properties of the NN and kNN classifiers. Additionally, 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. 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.

MFS is our first attempt at using random feature selection to generate effective NN ensembles, and although successful at improving accuracy, there are still many unanswered questions and open areas for future work:

1. *Why does MFS 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 as we cannot even characterize the domains MFS will do well on.

2. *Application to other classifiers.* We showed that random feature selection is useful for generating ensembles of NN classifiers. Can we apply this technique to other learning algorithms?

3. *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.

4. *Other Improvements.* In this paper, we kept the design of MFS as simple as possible; however, there are a number of obvious improvements that may help accuracy and speed. In particular, we would like to investigate: (1) different weighting schemes, (2) varying the number of features each classifier uses, (3) postpruning the ensemble, (4) combining more sophisticated versions of the NN classifier, and (5) editing the prototypes.

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 work was partially supported by an NSERC PGS A scholarship.

References

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

K. M. Ali and M. J. Pazzani. (1996). Error reduction through learning multiple descriptions. *Machine Learning*, 24:173–202.

E. Alpaydin. (1997). Voting over multiple condensed nearest neighbors. *Artificial Intelligence Review*, 11(1-5):115–132.

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

L. Breiman. (1996). Bagging predictors. *Machine Learning*, 24:123–140.

L. Breiman. (1996b). Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California, Berkeley.

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

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

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

R. O. Duda and P. E. Hart. (1973). *Pattern Classification and Scene Analysis.* John Wiley, New York.

R. A. Fisher. (1936). The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7:179–188.

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

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

P. E. Hart. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–516.

G. James and T. Hastie. (1997). Generalizations of the bias/variance decomposition for prediction error. http://stat.stanford.edu/∼gareth.

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

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

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

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

C. J. Merz and P. M. Murphy. (1998). UCI repository of machine learning databases. University of California, Irvine, Dept. of Information and Computer Science. http://www.ics.uci.edu/∼mlearn/.

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

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

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

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

R. Tibshirani. (1996). Bias, variance and prediction error for classification rules. Technical report, Department of Statistics, University of Toronto.

K. Tumer and J. Ghosh. (1996). Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8:385–404. Special issue on combining artificial neural networks: ensemble approaches.