Machine Learning 36, 33–58 (1999)

Using Correspondence Analysis to Combine Classifiers

CHRISTOPHER J. MERZ cmerz@uci.edu
Department of Information and Computer Science, University of California, Irvine, CA 92697-3425

Editors: Salvatore Stolfo, Philip Chan and David Wolpert

Abstract. Several effective methods have been developed recently for improving predictive performance by generating and combining multiple learned models. The general approach is to create a set of learned models either by applying an algorithm repeatedly to different versions of the training data, or by applying different learning algorithms to the same data. The predictions of the models are then combined according to a voting scheme. This paper focuses on the task of combining the predictions of a set of learned models. The method described uses the strategies of stacking and Correspondence Analysis to model the relationship between the learning examples and their classification by a collection of learned models. A nearest neighbor method is then applied within the resulting representation to classify previously unseen examples. The new algorithm does not perform worse than, and frequently performs significantly better than other combining techniques on a suite of data sets.

Keywords: classification, correspondence analysis, multiple models, combining estimates

1. Introduction

The machine learning and neural network communities have recently placed considerable attention on the task of generating and combining multiple learned models with the goal of forming an improved estimate. The learned models may be decision/regression trees, rule lists, neural networks, etc. The problem is to decide which models to rely upon for prediction and how much weight to give each. The goal of combining learned models is to obtain a more accurate prediction than can be obtained from any single source alone.

Techniques using multiple models consist of two phases: model generation and model combination. It is important to generate a set of models that are diverse in the sense that they make errors in different ways. On the other hand, the types of errors made by the model set directly determine the appropriate combining strategy (Perrone, 1994). When the errors are uncorrelated, the optimal approach is to take the majority vote (also referred to as plurality voting). However, when patterns exist in the errors of the learned models, a more elaborate combining scheme is necessary.

The focus of this research is on model combination. A general technique is presented for combining the predictions of a set of learned models, independent of how they are generated. Section 2 defines the problem and explores the caveats of solving it in more detail. The approach taken, called SCANN (Section 3), uses the strategies of stacking (Wolpert, 1992) and correspondence analysis (Greenacre, 1984) to model the relationship

between the learning examples and the way in which they are classified by a collection of learned models. The new representation is captured in a space of uncorrelated dimensions. A nearest neighbor method is then applied within the resulting representation to classify unseen examples.

Section 5 demonstrates the properties of SCANN by applying it to two artificially generated model sets. In an empirical evaluation on a suite of data sets (Section 6), the naive approach of taking the plurality vote (PV) frequently exceeds the performance of the constituent learners. SCANN, in turn, typically exceeds the performance of PV and several other stacking-based approaches. The analysis reveals that SCANN is not sensitive to having many poor constituent learned models, and it is not prone to overfit by reacting to insignificant fluctuations in the predictions of the learned models. Related work, limitations, and future work are discussed in Sections 7 and 8. Concluding remarks are given in Section 9. An expanded version of this work may be found in (Merz, 1998).

2. Problem definition and motivation

The problem of generating a set of learned models is defined as follows. Suppose two sets of data are given: a training set

$$\mathcal{L} = \{(\mathbf{x}_m, y_m), m = 1, \ldots, M\},$$

and a test set

$$\mathcal{T} = \{(\mathbf{x}_t, y_t), t = 1, \ldots, T\}.$$

Each set is a sample of the true underlying function $f(\mathbf{x})$. $\mathbf{x}_i$ is a vector of input values which are either nominal or numeric, and $y_i \in Y = \{\mathcal{C}_1, \ldots, \mathcal{C}_{|Y|}\}$ where $|Y|$ is the number of classes. Now suppose $\mathcal{L}$ is used to build a set of functions, $\mathcal{F} = \{\hat{f}_i(\mathbf{x}), i = 1, \ldots, N\}$, each element of which approximates $f(\mathbf{x})$. The goal is to find the best approximation of $f(\mathbf{x})$ using $\mathcal{F}$.

One approach is to use a particular learning algorithm and a data resampling technique to create a set of learned models and then combine their predictions according to a voting scheme. The idea behind data resampling is that models generated from different samples of the training data are likely to make errors in different ways. Data resampling techniques include bootstrapping (Efron & Tibshirani, 1993) and data partitioning (Meir, 1995). In the combining phase, the ith learned model's prediction or vote for a given class has a strength proportional to its assigned weight, α_i. The class receiving the most votes is the final class prediction. This is referred to as a *weighted majority* scheme, i.e.,

$$\hat{f}(\mathbf{x}) = \arg\max_{c \in Y} \sum_{i=1}^{N} \alpha_i \pi(\hat{f}_i(\mathbf{x}), c) \tag{1}$$

where $\pi(a, b)$ is one if a is equal to b, and zero otherwise. One example of this approach is Bagging (Breiman, 1996) where resampling occurs randomly with replacement and all

models are assigned equal weight. Another example of this approach is Boosting (Freund, 1995) where the training examples are resampled as a function of how well they were classified by the previously generated model. The models are then weighted according to their estimated error rate.

Another approach is to use a variety of learning algorithms on all of the training data and combine their predictions according to a voting scheme. This technique attempts to achieve diversity in the errors of the learned models by using different learning algorithms which vary in their method of search and/or representation. The intuition is that the models generated using different learning biases are more likely to make errors in different ways. Plurality voting with a model set consisting of a neural networks, decision trees, rule sets, and other models was shown to be effective in (Merz, 1995). The search strategy of a learning algorithm may also be modified to diversify the model set. Maclin and Shavlik (1995) accomplished this by strategically initializing the weights of a neural network. Ali and Pazzani (1995) generated decision lists (i.e., list of rules) where conditions are added to a rule stochastically. As with the first approach, the models are typically combined using variants of the weighted majority strategy or plurality voting.

Though these approaches are effective, they emphasize the model generation and not the model combination process (e.g., Opitz & Shavlik, 1996). As a result, most combining methods are rather limited in their ability to identify the unique contributions of each model and, at the same time, remain insensitive to the inherent redundancy in the model set. For example, the strategy of taking the majority vote has been shown to be fairly effective (Merz, 1995), however, it may perform poorly in two scenarios: when a subset of redundant and less accurate models comprise the majority, and when a dissenting vote is not recognized as an area of specialization for a particular model. An effective combining strategy must be able to override the majority vote for examples where the dissenting vote is more likely to be correct.

This paper presents a technique for effectively addressing the issues described above. First, the relationship between the examples and the way in which the learned models classify them is represented in a set of uncorrelated dimensions using a technique known as correspondence analysis (Greenacre, 1984). In each of the resulting dimensions, every model is assigned one weight per possible class label: $\alpha_{i,c,k}$ denotes the weight of the ith model for the cth class in the kth dimension. The weighting scheme derived from this representation may be as simple as the majority vote when the errors of the learned models are uncorrelated. However, more elaborate weighting schemes are produced when the errors of the models have patterns of correlation.

3. The SCANN algorithm

A learning algorithm can be broken down into four parts: representation, classification, search, and evaluation. Section 3.1 discusses the first two components by describing how the predictions of the learned models can be mapped to a new representation using correspondence analysis (Greenacre, 1984), and how test examples can be classified using a nearest neighbor algorithm. The search and evaluation aspects of SCANN are covered in Section 3.2. A detailed trace of SCANN on a example problem is given in Section 4.

3.1. Representation and classification

The representation used in SCANN is based on the variates derived using correspondence analysis (Greenacre, 1984). Sections 3.1.1 and 3.1.2 show how stacking and CA are used to generate the new representation. A nearest neighbor strategy is then used to locate and classify test examples using the new representation (Section 3.1.3). Together, Stacking, Correspondence Analysis, and Nearest Neighbor make up the core of the SCANN algorithm which is summarized in Section 3.1.4.

3.1.1. Stacking. Once a diverse set of models has been generated, the issue of how to combine them arises. Wolpert (1992) provided a general framework for doing so called *stacked generalization* or *stacking*. The goal of stacking is to combine the members of $\mathcal{F}$ based on information learned about their particular biases with respect to $\mathcal{L}$.[1]

The basic premise of stacking is that this problem can be cast as another induction problem where the input space is the (approximated) outputs of the learned models, and the output space is the same as before, i.e.,

$$\mathcal{L}_1 = \{((\hat{f}_1(\mathbf{x}_i),\ \hat{f}_2(\mathbf{x}_i), \ldots,\ \hat{f}_N(\mathbf{x}_i)), y_i), i = 1, \ldots, I\}$$

The approximated outputs of each learned model, represented as $\hat{f}_n(\mathbf{x}_i)$, are generated using the following in-sample/out-of-sample approach:

1. Divide the $\mathcal{L}_0$ data into V partitions.
2. For each partition, v,
   - Train each algorithm on all but partition v to get $\{\hat{f}_n^{-v}\}$.
   - Test each learned model in $\{\hat{f}_n^{-v}\}$ on partition v.
   - Pair the predictions on each example in partition v (i.e., the new *input space*) with the corresponding output, and append the new examples to $\mathcal{L}_1$.
3. Return $\mathcal{L}_1$

3.1.2. Correspondence analysis. Correspondence analysis (CA) (Greenacre, 1984) is a method for geometrically modeling the relationship between the rows and columns of a matrix whose entries are categorical. The goal here is to explore the relationship between the training examples and their classification by the learned models. To do this, a model is built using the prediction matrix, $\mathbf{M}$, where $\mathbf{M}_{i,n} = \hat{f}_n(\mathbf{x}_i)$ ($1 \le i \le I$, and $1 \le n \le N$). It is also important to see how the predictions for the training examples relate to their true class labels, so the class labels are appended to form $\mathbf{M}'$, an $(I \times J)$ matrix (where $J = N + 1$). For proper application of CA, $\mathbf{M}'$ must be converted to an $(I \times (J \cdot |Y|))$ *indicator matrix*, $\mathbf{N}$, where $n_{i,(j \cdot J + c)}$ is a one exactly when $m_{ij} = \mathcal{C}_c$, and zero otherwise.

The calculations of CA may be broken down into three stages (see Table 1). Stage one consists of some preprocessing calculations performed on $\mathbf{N}$ which lead to the *standardized residual matrix*, $\mathbf{A}$. In the second stage, a singular value decomposition (SVD) is performed on $\mathbf{A}$ to redefine it in terms of three matrices: $\mathbf{U}_{(I \times K)}$, $\Gamma_{(K \times K)}$, and $\mathbf{V}_{(K \times J)}$, where

Table 1. Correspondence analysis calculations.

Stage	Symbol	Definition	Description
1	$\mathbf{N}$	$(I \times (J \cdot \lvert Y\rvert))$ indicator matrix	Records votes of learned models
	n	$\sum_{i=1}^{I}\sum_{j=1}^{J}\mathbf{N}_{i,j}$	Grand total of table $\mathbf{N}$
	$\mathbf{r}$	$\mathbf{r}_i = \mathbf{N}_{i+}/n$	Row masses
	$\mathbf{c}$	$\mathbf{c}_j = \mathbf{N}_{+j}/n$	Column masses
	$\mathbf{P}$	$(1/n)\mathbf{N}$	Correspondence matrix
	$\mathbf{D_c}$	$(J \times J)$ diagonal matrix	Masses $\mathbf{c}$ on diagonal
	$\mathbf{D_r}$	$(I \times I)$ diagonal matrix	Masses $\mathbf{r}$ on diagonal
	$\mathbf{A}$	$\mathbf{D_r}^{-1/2}(\mathbf{P} - \mathbf{r}\mathbf{c}^T)\mathbf{D_c}^{-1/2}$	Standardized residuals
2	$\mathbf{A}$	$\mathbf{U\Gamma V^T}$	SVD of $\mathbf{A}$
3	$\mathbf{F}$	$\mathbf{D_r}^{-1/2}\mathbf{U\Gamma}$	Principal coordinates of rows
	$\mathbf{G}$	$\mathbf{D_c}^{-1/2}\mathbf{V\Gamma}$	Principal coordinates of columns

$K = \min(I-1, J-1)$. These matrices are used in the third stage to determine $\mathbf{F}_{(I\times K)}$ and $\mathbf{G}_{(J\times K)}$, the coordinates of the rows and columns of $\mathbf{N}$, respectively, in the new space. Note that not all K dimensions are necessary. Section 3.1.4, describes how the final number of dimensions, $K*$, is determined.

In the new geometric representation, rows $\mathbf{f}_{p*}$ and $\mathbf{f}_{q*}$ in $\mathbf{F}$, corresponding to rows p and q in $\mathbf{N}$, will lie close to one another when examples p and q receive similar predictions from the collection of learned models. Likewise, rows $\mathbf{g}_{s*}$ and $\mathbf{g}_{t*}$ in $\mathbf{G}$, corresponding to columns s and t in $\mathbf{N}$, will lie close to one another when the learned models corresponding to s and t make similar predictions for the set of examples. Finally, the relationship between a row and column in $\mathbf{N}$ is captured as follows. Each column, s, in $\mathbf{N}$ records when a learned model, j', predicts a particular class label, c'. An example, p, with the associated point $\mathbf{f}_{p*}$ will lie closer to $\mathbf{g}_{s*}$ when model j' predicts class c'. An example illuminating the entire SCANN algorithm will be given in Section 4.

3.1.3. Nearest neighbor. The nearest neighbor algorithm is used to classify points in a weighted Euclidean space. In this scenario, each possible class will be assigned coordinates in the space derived by correspondence analysis. Unclassified examples will be mapped into the new space (as described below), and the class label corresponding to the closest class point is assigned to the example.

Since the actual class assignments for each example reside in the last $|Y|$ columns of $\mathbf{N}$, their coordinates in the new space can be found by looking in the last $|Y|$ *rows* of $\mathbf{G}$. For convenience, these class points will be called $\mathbf{C}_1, \ldots, \mathbf{C}_{|Y|}$.

To classify an unseen example, $\mathbf{x}_{\text{Test}}$, the predictions of the learned models on $\mathbf{x}_{\text{Test}}$ must be converted to a *row profile*, $\tilde{\mathbf{r}}$, of length $J \cdot C$, where $\tilde{r}_{(j\cdot J+c)}$ is $1/J$ exactly when $\mathbf{M}_{ij} = \mathcal{C}_c$, and zero otherwise. However, since the example is unclassified, $\mathbf{x}_{\text{Test}}$ is of length $(J-1)$ and can only be used to fill the first $((J-1)\cdot C)$ entries in $\tilde{\mathbf{r}}$. For this reason, C different versions are generated, i.e., $\tilde{\mathbf{r}}_1, \ldots, \tilde{\mathbf{r}}_C$, where each one "hypothesizes" that $\mathbf{x}_{\text{Test}}$ belongs

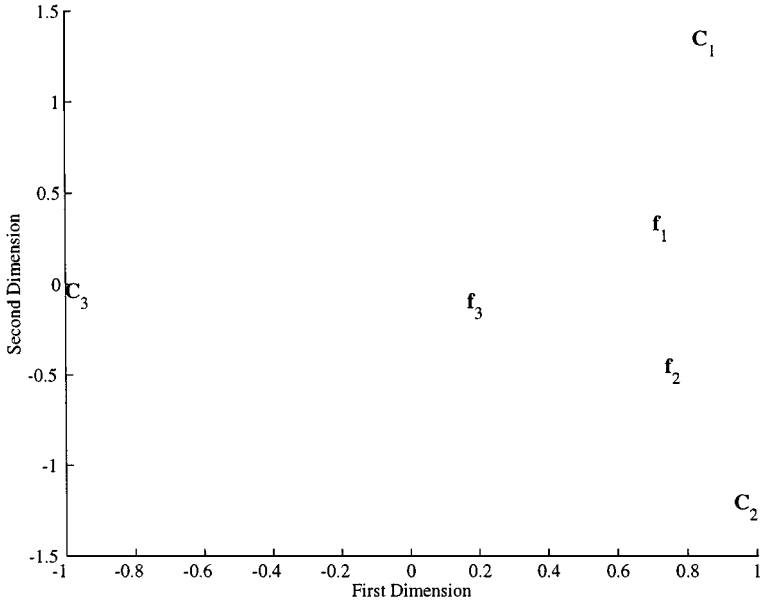

Figure 1. Sample classification of a test example, $\mathbf{x}_{\text{Test}}$, in two dimensions. $\mathbf{C}_1$, $\mathbf{C}_2$, and $\mathbf{C}_3$ denote the true class points as extracted from $\mathbf{G}$. The points $\mathbf{f}_1$, $\mathbf{f}_2$, and $\mathbf{f}_3$ are the hypothesized locations of $\mathbf{x}_{\text{Test}}$ for each possible class.

to one of the C classes (by putting $1/J$ in the appropriate column). Locating these profiles in the scaled space is a matter of simple matrix multiplication, i.e., $\mathbf{f}_c = \tilde{\mathbf{r}}_c \mathbf{G} \boldsymbol{\Gamma}^{-1}$. The $\mathbf{f}_c$ which lies closest to a class point, $\text{Class}_{c'}$, is considered the "correct" class, and $\mathbf{x}_{\text{Test}}$ is assigned the class label c'.

Figure 1 shows a test example as it is mapped into the first two dimensions of the scaled space. In this case the point is classified as class 2 because $\mathbf{f}_2$ lies closest to the point associated with class 2, $\mathbf{C}_2$, in the space. A complete example of SCANN is given in Section 4.

3.1.4. The SCANN algorithm. Now that the three main parts of the approach have been described, a summary of the SCANN algorithm can be given as a function of $\mathbf{M}$, $\mathcal{L}_0$ and the constituent learning algorithms, $\mathcal{A}$ (see Table 2). The first step is to use $\mathcal{L}_0$ and $\mathcal{A}$ to generate the stacking data, $\mathcal{L}_1$, capturing the approximated predictions of each learned model. Next, $\mathcal{L}_1$ is used to form the indicator matrix, $\mathbf{N}$. A correspondence analysis is performed on $\mathbf{N}$ to derive the scaled space, $\mathbf{A} = \mathbf{U}\boldsymbol{\Gamma}\mathbf{V}^{\mathbf{T}}$. The number of dimensions retained from this new representation, $K*$, is the value which optimizes classification on $\mathcal{L}_1$ (see Section 3.2). The resulting scaled space is used to derive the row/column coordinates $\mathbf{F}$ and $\mathbf{G}$, thus geometrically capturing the relationships between the examples, the way in which they are classified, and their position relative to the true class labels. Finally, the nearest neighbor strategy exploits the new representation by predicting which class is most likely according to the predictions made on a novel example.

Table 2. The SCANN algorithm.

SCANN($\mathbf{M}$, $\mathcal{L}_0$, $\mathcal{A}$)
Input

$\mathbf{M}$: The matrix of predictions of the models in $\mathcal{F}$

$\mathcal{L}_0$: The Level-0 learning data

$\mathcal{A}$: A set of learning algorithms

Begin

1. Use $\mathcal{L}_0$ and $\mathcal{A}$ to generate stacking data, $\mathcal{L}_1$
2. Use $\mathcal{L}_1$ to form indicator matrix, $\mathbf{N}$
3. Perform CA on $\mathbf{N}$ to derive scaled space, $\mathbf{A} = \mathbf{U}\mathbf{\Gamma}\mathbf{V}^{\mathbf{T}}$
4. Choose number of dimensions to retain, $K*$, as the value which optimizes classification on $\mathcal{L}_1$
5. Derive row/column coordinates $\mathbf{F}$ and $\mathbf{G}$
6. Return: $\mathbf{U}$, $\mathbf{\Gamma}$, $\mathbf{V}^{\mathbf{T}}$, $\mathbf{F}$ and $\mathbf{G}$

End

3.2. Search procedure for finding $K*$

The search aspect of SCANN is to choose the number of components, $K*$, to retain from the set derived by the correspondence analysis component. The procedure begins by including only the first component (i.e., $K* = 1$) of the scaled space. Then the nearest neighbor classification algorithm is used to classify the examples in $\mathcal{L}_1$, and the error rate for $K* = 1$ is recorded. The procedure continues by increasing $K*$ until all the components are used in classifying the examples. The value of $K*$ with the lowest associated error rate is chosen.

4. An example of SCANN

This section provides a simple example to illustrate the SCANN algorithm on an artificial data set. The predictions and truth values were artificially generated in a fashion similar to the data set described later in Section 5.1. The procedure begins with the prediction matrix, $\mathbf{M}'$. Table 3 shows a collection of three models making predictions for 15 examples in a 3-class problem. The columns labelled f_1, f_2, and f_3, contain the predictions of the learned models for each example, $\mathbf{x}_i$. The "Truth" column contains the actual class values for each example.

Table 4 shows the indicator matrix, $\mathbf{N}$, generated from $\mathbf{M}'$. Note that each column in $\mathbf{M}'$ is expanded into one column for each possible class value. Column j in $\mathbf{N}$ is associated with a particular learned model,[2] f_h, and a particular class label, c_l. A 1 appearing in $\mathbf{N}_{i,j}$ indicates that $f_h(\mathbf{x}_i) = c_l$. Otherwise, $\mathbf{N}_{i,j} = 0$.

The probability matrix, $\mathbf{P}$, follows directly from $\mathbf{N}$ by dividing by the grand total of $\mathbf{N}$, i.e., $n = 60$.

Table 3. The prediction matrix, $\mathbf{M}'$, for 15 examples and 3 learned models.

	Prediction			
Example	f_1	f_2	f_3	Truth
x_1	c_1	c_1	c_1	c_1
x_2	c_2	c_2	c_2	c_2
x_3	c_2	c_2	c_2	c_2
x_4	c_2	c_2	c_1	c_1
x_5	c_3	c_1	c_3	c_3
x_6	c_3	c_3	c_3	c_3
x_7	c_1	c_1	c_3	c_1
x_8	c_2	c_3	c_3	c_3
x_9	c_2	c_2	c_2	c_2
x_{10}	c_1	c_2	c_3	c_3
x_{11}	c_3	c_3	c_3	c_3
x_{12}	c_2	c_1	c_1	c_1
x_{13}	c_3	c_3	c_2	c_3
x_{14}	c_2	c_2	c_3	c_2
x_{15}	c_3	c_3	c_3	c_3

Table 4. Excerpts from the indicator matrix, $\mathbf{N}$, associated with the prediction matrix, $\mathbf{M}'$.

	Prediction											
	f_1			f_2			f_3			Truth		
Example	c_1	c_2	c_3	c_1	c_2	c_3	c_1	c_2	c_3	c_1	c_2	c_3
x_1	1	0	0	1	0	0	1	0	0	1	0	0
x_2	0	1	0	0	1	0	0	1	0	0	1	0
	...											
x_{15}	0	0	1	0	0	1	0	0	1	0	0	1

The row and column masses, $\mathbf{r}$ and $\mathbf{c}$, are calculated by summing each row/column in $\mathbf{N}$ and dividing by n. Since each row in $\mathbf{N}$ has four 1s, $\mathbf{r}_i = 4/60 = 0.067$, for $i = 1, \ldots, 15$. The column masses all differ as a function of how often each class is predicted by each model (see Table 5). The vectors $\mathbf{r}$ and $\mathbf{c}$ are used to form the diagonal matrices, $\mathbf{D_r}$ and $\mathbf{D_c}$.

The calculations from above are now used to finish the first stage of CA (see Table 1) by computing the standardized residual matrix (see Table 6),

$$\mathbf{A} = \mathbf{D_r}^{-1/2}(\mathbf{P} - \mathbf{r}\mathbf{c}^T)\mathbf{D_c}^{-1/2}$$

Table 5. Column masses, **c**, for model prediction columns in **N**.

f_1			f_2			f_3			Truth		
c_1	c_2	c_3	c_1	c_2	c_3	c_1	c_2	c_3	c_1	c_2	c_3
.050	.117	.083	.067	.100	.083	.050	.067	.133	.067	.067	.117

Table 6. Excerpts from the standardized residual matrix, **A**.

	f_1			...	Truth		
Example	c_1	c_2	c_3	...	c_1	c_2	c_3
x_1	0.2309	−0.0882	−0.0745		0.1833	−0.0667	−0.0882
x_2	−0.0577	0.1008	−0.0745		−0.0667	0.1833	−0.0882
	...						
x_{15}	−0.0577	−0.0882	0.1491		−0.0667	−0.0667	0.1008

Stage two of the calculations of CA consists of a singular value decomposition of **A**:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$$

Only eight singular values were found on the diagonal of Σ, indicating that $K = 8$:

$$diag(\Sigma) = [\,0.8488\ 0.8111\ 0.4858\ 0.395\ 0.3353\ 0.2488\ 0.1897\ 0.1393\,]$$

Stage three of the CA calculations in Table 1 may now be completed by deriving the principal coordinates of the rows and columns, respectively:

$$\mathbf{F} = \mathbf{D}_{\mathbf{r}}^{-1/2}\mathbf{U}\Sigma$$

$$\mathbf{G} = \mathbf{D}_{\mathbf{c}}^{-1/2}\mathbf{V}\Sigma$$

Table 7 shows the first three dimensions of the principal coordinates of **G** in the scaled space. These numbers represent the weights for each model, for each class, for the first three dimensions.

The first two dimensions of **F** are plotted in figure 2. The class points $\mathbf{C}_1$, $\mathbf{C}_2$ and $\mathbf{C}_3$ from **G** are also included in the plot to show their relationship to the example points. The point labelled $\mathbf{x}_*$ denotes examples $\mathbf{x}_6$, $\mathbf{x}_{11}$ and $\mathbf{x}_{15}$, and the point labelled $\mathbf{x}_+$ denotes examples $\mathbf{x}_2$, $\mathbf{x}_3$ and $\mathbf{x}_9$. Notice that examples where the models are in total agreement, i.e., $\mathbf{x}_*$ and $\mathbf{x}_+$, lie closer to the correct class point indicating a higher degree of confidence for examples with unanimous predictions. On the other, points with mixed predictions lie closer to the origin, i.e., $\mathbf{x}_{10}$.

Table 7. The principal coordinates, **G**, of the columns of **A**. The first three of eight axes are shown. These values represent the first three dimensions of the combining weights for each model for each class label.

		Weight	$k=1$	$k=2$	$k=3$
f_1	c_1	$\alpha_{1,1,k}$	0.313	1.228	1.355
	c_2	$\alpha_{1,2,k}$	0.737	−0.471	−0.332
	c_3	$\alpha_{1,3,k}$	−1.220	−0.077	−0.348
f_2	c_1	$\alpha_{2,1,k}$	0.310	1.315	0.078
	c_2	$\alpha_{2,2,k}$	0.784	−0.690	0.330
	c_3	$\alpha_{2,3,k}$	−1.188	−0.224	−0.458
f_3	c_1	$\alpha_{3,1,k}$	1.001	1.270	−0.998
	c_2	$\alpha_{3,2,k}$	0.525	−1.153	−0.171
	c_3	$\alpha_{3,3,k}$	−0.638	0.100	0.460
Truth	c_1	$\mathbf{C}_{1,k}$	0.821	1.330	−0.336
	c_2	$\mathbf{C}_{2,k}$	0.932	−1.226	0.271
	c_3	$\mathbf{C}_{3,k}$	−1.002	−0.059	0.037

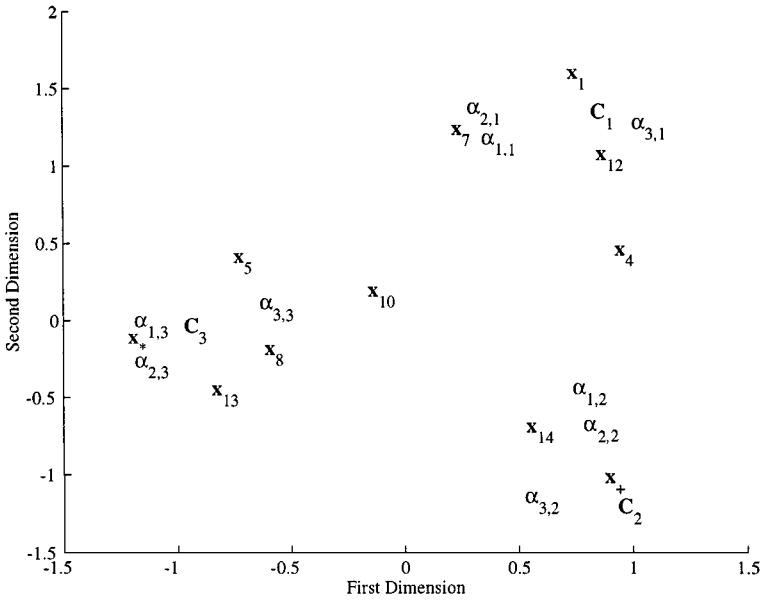

Figure 2. The first two dimensions of the principal coordinates in **F** and **G**. The point labelled $\mathbf{x}_*$ denotes examples $\mathbf{x}_6$, $\mathbf{x}_{11}$ and $\mathbf{x}_{15}$. The point labelled $\mathbf{x}_+$ denotes examples $\mathbf{x}_2$, $\mathbf{x}_3$ and $\mathbf{x}_9$. A point labelled $\alpha_{i,j}$ denotes the weight of $\hat{f}_i$ on class $\mathcal{C}_j$.

Table 8. The possible values of $K*$ and their associated error rates for classifying the examples in $\mathbf{M}'$.

$K*$	1	2	3	4	5	6	7	8
Error rate	0.222	0.111	0.111	0.055	0.055	0.055	0.055	0.055

The first two dimensions of the weights in $\mathbf{G}$ are also plotted in figure 2. A point labelled $\alpha_{i,j}$ denotes the weight of $\hat{f}_i$ on class $\mathcal{C}_j$. Each model's weight lies close to the appropriate class. When mapping an example into the space, the weight $\alpha_{i,j}$ serves to project the example part of the way (from the origin) towards $\alpha_{i,j}$. For instance, example $\mathbf{x}_1$ lies close to $\mathbf{C}_1$ because all of the models predicted its class as c_1. On the other hand, example $\mathbf{x}_{10}$ lies close to the origin because the weights cancel each other out when each class receives one vote.

The search procedure for finding $K*$ (described in Section 3.2) returned the values listed in Table 8. SCANN chooses the first occurrence of the lowest value for $K*$, i.e., $K* = 4$. Note that the error rate need not improve as $K*$ grows. The next section will illustrate how the correlation of the errors of the learned models affect the value of $K*$.

To classify a test example, $\mathbf{x}_{\text{Test}}$, where

$$\mathbf{m}^{\mathbf{x}_{\text{Test}}} = [c_2 \quad c_1 \quad c_2],$$

the following row vectors are generated to hypothesize each class:

$$\tilde{\mathbf{r}}_1 = [0 \;\; 0.25 \;\; 0 \quad 0.25 \;\; 0 \;\; 0 \quad 0 \;\; 0.25 \;\; 0 \quad 0.25 \;\; 0 \;\; 0],$$
$$\tilde{\mathbf{r}}_2 = [0 \;\; 0.25 \;\; 0 \quad 0.25 \;\; 0 \;\; 0 \quad 0 \;\; 0.25 \;\; 0 \quad 0 \;\; 0.25 \;\; 0],$$
$$\tilde{\mathbf{r}}_3 = [0 \;\; 0.25 \;\; 0 \quad 0.25 \;\; 0 \;\; 0 \quad 0 \;\; 0.25 \;\; 0 \quad 0 \;\; 0 \;\; 0.25].$$

The hypothesized row vectors are mapped into the scaled space according to

$$\mathbf{f}_c = \tilde{\mathbf{r}}_c \mathbf{G} \mathbf{\Sigma}^{-1}.$$

The hypothesized points in the four dimensions used (by $K*$) are:

$$\mathbf{f}_1 = [0.7047 \quad 0.3144 \quad -0.3919 \quad -0.7242] \tag{2}$$
$$\mathbf{f}_2 = [0.7375 \quad -0.4734 \quad -0.0794 \quad -0.8064] \tag{3}$$
$$\mathbf{f}_3 = [0.1678 \quad -0.1136 \quad -0.1998 \quad -0.6957] \tag{4}$$

Figure 1 shows a plot of the hypothesized points with respect to the class points in the first two dimensions. The distance between $\mathbf{f}_i$ and $\mathbf{C}_j$) is minimized when $i = 2$ and $j = 2$. Thus, the predicted class of example $\mathbf{x}_{\text{Test}}$ is c_2.

5. Understanding SCANN analytically

The models sets used by SCANN will have varying degrees of correlation in the errors committed. When there is no apparent pattern in the errors committed, the errors are said to be uncorrelated. If distinct patterns occur in the errors, e.g., $\hat{f}_i$ is particularly good at classifying class c, then the errors are said to be correlated. In the former case, a simple approach like PV is most effective (Perrone, 1994). In the latter case, a more complex combining scheme is needed. An effective combining strategy must be able to adjust for both situations. Sections 5.1 and 5.2 evaluate how SCANN handles these two scenarios. Finally, Section 5.4 discusses how the scaled space derived using correspondence analysis enhances the nearest neighbor classification algorithm.

5.1. Handling uncorrelated errors

To see how SCANN handles models sets with uncorrelated errors, an artificial data set, $A1$, was generated for a 3-class problem simulating the predictions of ten models. The true function, f, was represented by 300 examples where each class was equally represented. Model $\hat{f}_i$ $(1 \leq i \leq 10)$ was set equal to f for each example with a 10% chance of being wrong, in which case one of the incorrect classes was selected at random. The examples were randomly divided into a training $(2/3)$ and test $(1/3)$ partition.

Kappa-Error diagrams (Margineantu & Dietterich, 1997) were used to visualize the differences between the models (see figure 3). In a Kappa-Error diagram, the x coordinate is the κ statistic measured between two models, $\hat{f}_a$ and $\hat{f}_b$:

$$\kappa(\hat{f}_a, \hat{f}_b) = \frac{\Theta_1(\hat{f}_a, \hat{f}_b) - \Theta_2(\hat{f}_a, \hat{f}_b)}{1 - \Theta_2(\hat{f}_a, \hat{f}_b)}$$

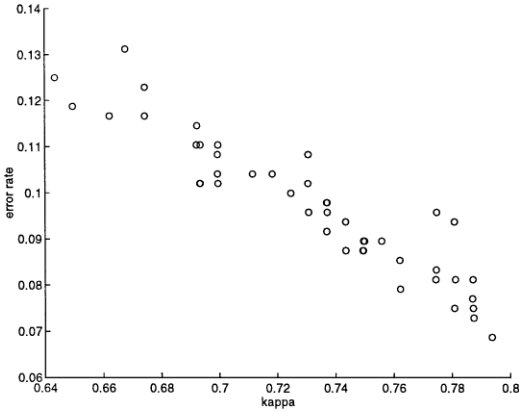

Figure 3. Kappa-Error diagram for models in the $A1$ data set.

where

$$\Theta_1(\hat{f}_a, \hat{f}_b) = \frac{\sum_{i=1}^{|Y|} \mathbf{B}_{i,i}}{M}$$

$$\Theta_2(\hat{f}_a, \hat{f}_b) = \sum_{i=1}^{|Y|} \left(\sum_{j=1}^{|Y|} \frac{\mathbf{B}_{i,j}}{M} \cdot \sum_{j=1}^{|Y|} \frac{\mathbf{B}_{j,i}}{M} \right)$$

and $\mathbf{B}_{i,j}$ is the number of examples $\mathbf{x}$ for which $\hat{f}_a = i$ and $\hat{f}_b = j$. κ is a value between zero and one measuring the level of agreement between two models on a set of examples; κ is zero when the level of agreement between the models is the same as that by chance, and κ is 1 when the two models agree on every example. The y coordinate in a Kappa-Error diagram is the average error of two models, i.e.,

$$\frac{\text{error}(f_i) + \text{error}(f_j)}{2}.$$

A point in the diagram reports the level of similarity and average error for two classifiers.

Figure 3 is a Kappa-Error diagram for all pairs of models in $A1$ where both κ and the error rates are calculated on the training set. Models in the lower right corner are very accurate but very similar. Models in the upper left are dissimilar but have higher error rates. The points in this diagram indicate that the models are evenly dispersed with respect to their level of agreement and error. In this case, one would expect plurality voting (PV) to do well.

SCANN and PV were run on the train/test partition from above, and both reported a test error of zero (and a *kappa* statistic of 1). Similar results were obtained for different random partitions. This indicates that SCANN behaves in the same way as PV when the errors are uncorrelated.

5.2. Discovering unique contributions

Combining strategies must also be able to handle model sets which make errors in a patterned way. For example, some models in the set may have a particularly low error rate in certain parts of the example space. An effective combiner will be able to isolate those areas of expertise and use them to form an improved estimate. To test SCANN's ability to handle this problem, a second artificial data set, $A2$, was created where three of the models have particular areas of expertise.

$A2$, is generated in the same way as $A1$ but the examples are exposed to another round of corruption. Each example has a 25% chance of being misclassified by every model except one, otherwise it remains unmodified. In the modified examples, the single correct model is chosen according to the correct class of the example, i.e., models $\hat{f}_1$, $\hat{f}_2$, and $\hat{f}_3$ are always correct for classes 1, 2, and 3, respectively. The challenge for a combiner is to capitalize on the areas of (relative) expertise for those three models. PV should not do well here because the plurality vote is incorrect for approximately 25% of the examples.

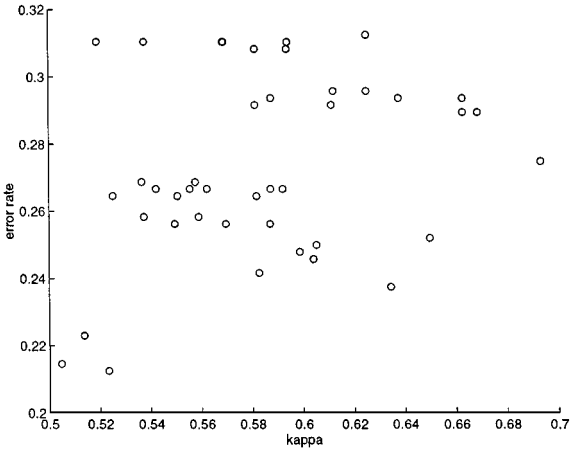

Figure 4. Kappa-Error diagram for models in the *A*2 data set.

Figure 4 shows the Kappa-Error diagram for the models on the training data. The diagram shows little uniformity between models for either error rate or κ. This indicates that the models do not make errors in an uncorrelated fashion. In fact, the three points in the lower left corner correspond to the pairings of models $\hat{f}_1$-$\hat{f}_2$, $\hat{f}_2$-$\hat{f}_3$, and $\hat{f}_1$-$\hat{f}_3$. These three points indicate that models $\hat{f}_1$, $\hat{f}_2$, and $\hat{f}_3$ are fairly different in the predictions they make, but have similar and low error rates.

SCANN and PV were run on the corresponding test partition to obtain error rates of 0.158 and 0.225, respectively. The κ statistic echoes the difference between SCANN and PV with a value of 0.88 on the test data. All of the examples which the two methods disagreed on were cases where SCANN relied upon the only correct model.

5.3. Computational complexity

The time complexity analysis begins by assuming that N models have been built from M examples with $|Y|$ possible classes (see Table 2). The time complexity for SCANN is broken down according to the three stages of the algorithm:

1. *Stacking*. To generate the $\mathcal{L}_1$ data, each learning algorithm must be run V times (see Section 3.1.1). Therefore, the time complexity of this stage is a function of the constituent learning algorithms, $\mathcal{A}$, i.e.,

$$\mathcal{O}\left(\arg\max_{n\in N} \mathcal{O}(\mathcal{A}_n)V\right).$$

2. *Correspondence analysis*. The dominating computation of this stage is the Singular Value Decomposition of $\mathbf{A}$ which takes $\mathcal{O}(\max(M, N|Y|)^3)$ time (Press, 1992). This

computation may be prohibitive for data sets with many examples. However, SCANN does not make use of the left singular vectors contained in **U** which provide the coordinates of the training examples in the scaled space. Therefore, for larger data sets, only the right singular vectors need to be computed, thus reducing this stage to $\mathcal{O}((N|Y|)^3)$ time. The SVD package used in this research (Dongarra & Grosse, 1998) allowed the user to specify which singular vectors were to be computed.

3. *Nearest neighbor*. The process of mapping an unclassified example into the scaled space has low time complexity; the $|Y|$ hypothesized points are each compared to the $|Y|$ actual class points, taking $\mathcal{O}(|Y|^2)$ time.

The overall time complexity of SCANN is determined by the stacking and correspondence analysis components of the algorithm, i.e.,

$$\mathcal{O}\left(\max\left(\arg\max_{n\in N}\mathcal{O}(\mathcal{A}_n)V,\ \ N^2\max(M,\ N|Y|)\right)\right).$$

This can be reduced to $\mathcal{O}((N|Y|)^3)$ if low complexity learning algorithms are used in $\mathcal{A}$ (such as decision trees and naive Bayesian classifiers) and only the right singular vectors are calculated in the correspondence analysis stage. Also, the stacking stage may be skipped and $\mathcal{L}_1$ can be determined using the predictions of the learned models built from all of $\mathcal{L}_0$.

5.4. Other aspects of SCANN

The properties of the correspondence analysis and nearest neighbor components of SCANN mix together well to produce an effective combining strategy for the following reasons:

1. The parameter $K*$ marks the last dimension of import, discarding the remaining irrelevant dimensions. This benefits nearest neighbor algorithms which are sensitive to irrelevant or noisy dimensions in their attribute space.
2. Nearest neighbor strategies work better when the attribute space chosen is compact, i.e., the dimensions used are filled with examples and not sparsely populated or irrelevant. Again, the choice of dimensionality, $K*$, will help to ensure that the dimensions retained contain relevant information about the predictions of the learned models.
3. The nearest neighbor algorithm is stable. Breiman (1994) defines the stability of an algorithm as its sensitivity to minor changes in the training data. Stable algorithms are not sensitive to small changes in the training data, unstable algorithms are. A general heuristic is to have the Level-0 learners be unstable, thus producing model set likely to make uncorrelated errors. However, instability is not a desired trait for a combining algorithm because one does not want the final prediction to be likely to change with small variations in the Level-1 data. Breiman's study revealed that the nearest neighbor algorithm is stable making it a desirable combining strategy.

6. Empirical evaluation of SCANN

This section contains the results of two experiments comparing SCANN to several other combining strategies on a collection of data sets. In the first experiment, the model set was generated using multiple learning algorithms. The goal here was to achieve diversity in the errors of the models by using completely different learning algorithms which vary in their method of search and/or representation. The model set in the second experiment was generated using Boosting (described in Section 7.1). Boosting generates a diverse model set by strategically resampling the training data. The second experiment has taken the typical Boosting approach of applying the same learning algorithm to all the data samples. Thus, the first experiment is an evaluation of SCANN's ability to combine a small set of models generated from a diverse collection of learning biases, whereas the second experiment is an evaluation of SCANN on a larger collection of models with the same learning bias.

6.1. Classification data sets

The data sets used were taken from the UCI Machine Learning Database Repository (Merz & Murphy, 1996), except for the unreleased medical data sets: *retardation* and *dementia*. A description of the data sets used is given in Table 9. The data sets with missing values were run only on the constituent learners capable of handling missing values (see Section 6.2).

Table 9. Data sets used in the empirical evaluation. The columns are, in order: name of data set; number of examples; number of attributes; number of numeric attributes; number of classes; and whether missing values exist.

Data set	Exs.	Atts.	Num.	Class	Missing
abalone	4177	8	7	3	No
balance	625	4	4	3	No
breast	286	9	4	2	Yes
credit	690	15	6	2	Yes
dementia	118	26	26	3	No
glass	214	10	10	7	No
heart	303	13	6	2	Yes
ionosphere	351	34	34	2	No
iris	150	4	4	3	No
krk	827	36	0	2	No
liver	345	6	6	2	No
lymphography	148	18	3	4	No
musk	476	168	168	2	No
retardation	700	20	0	2	No
sonar	208	60	60	2	No
vote	435	16	0	2	Yes
wave	300	40	40	3	No
wdbc	569	30	30	2	No

6.2. Constituent learners

The constituent learning algorithms, $\mathcal{A}$, spanned a variety of search and/or representation techniques. A standard implementation of Error Backpropagation (BP) (Rumelhart, Hinton, & Williams, 1986) was used to generate neural network models. All networks consisted of an input layer, a single hidden layer, and an output layer. For the input layer, a single input node was assigned to each numeric attribute. Nominal attributes were allocated one input node for each possible attribute value. When the ith value of a nominal attribute occurred in an example, the ith input node assigned to that attribute was assigned a value of one; the other input nodes assigned to that attribute were assigned a value of zero. The training examples were normalized to values between zero and one. Test patterns were normalized by the same transformation used on the training set. The initial weights of the networks were random values uniformly distributed in the interval $[-0.3, 0.3]$. A preliminary experiment was conducted for each data set (using only the training data) to determine the number of hidden units. Twenty percent of the training data was set aside as a validation set for determining when to stop training.

The CN2 algorithm (Clark & Niblett, 1989) was used to generate rule lists. Clark and Niblett's version 6.1 was used with the default parameters.

Decision trees were generated using C4.5 (Quinlan, 1993) and OC1 (Murthy et al., 1993). The default parameters were used for both algorithms. A second version of OC1 was run allowing only axis-parallel splits.

Two nearest neighbor approaches were used: PEBLS (Cost & Salzberg, 1993) and the first nearest neighbor (1-NN). For PEBLS, numeric attributes were discretized into ten bins spanning the range of possible values.

A naive Bayesian classifier (Duda & Hart, 1973) was also used. Numeric attributes were discretized in the same fashion as for PEBLS.

Depending on the data set, anywhere from five to eight algorithms were applied. OC1 and 1-NN were run only on data sets with all numeric attributes and no missing values (see Table 9). CN2 was not run on data sets with nominal attributes exceeding the maximum number of possible values.

6.3. Other combining methods

In addition to PV and SCANN, two other learners were evaluated using the stacking data, $\mathcal{L}_1$:

1. Stacking with Backpropagation (S-BP) is a good straw man because it is capable of capturing non-linear relationships in the predictions of the learned models.
2. Stacking with naive Bayesian classification (S-Bayes) is also a worthy straw man because it has proven to be a simple and effective Level-0 learner.

6.4. Experiment 1

Thirty runs per data set were conducted using a training/test partition of 70/30 percent. Table 10 contains comparisons of the combining strategies to the baseline combiner, PV.

Table 10. Comparison of combining strategies to PV.

	PV	SCANN vs. PV			S-BP vs. PV			S-Bayes vs. PV		
Data set	Error	Error	*w-l*	Ratio	Error	*w-l*	Ratio	Error	*w-l*	Ratio
abalone	80.35	39.38	**30-0**	**.490**	40.08	**30-0**	**.499**	39.11	**30-0**	**.487**
bal	13.81	12.43	**22-4**	**.900**	11.86	**21-2**	**.859**	13.71	14-12	.992
breast	4.31	3.82	**15-2**	**.886**	3.97	16-7	.920	3.80	**16-2**	**.881**
credit	13.99	13.98	10-8	.999	14.16	9-14	1.012	14.01	8-9	1.001
dementia	32.78	32.41	12-12	.989	33.98	11-14	1.037	30.56	16-9	.932
glass	31.44	31.69	11-15	1.008	36.41	**3-25**	**1.158**	38.21	**2-26**	**1.215**
heart	18.17	17.51	12-4	.964	18.13	12-15	.998	17.66	12-3	.972
ion	3.05	2.11	**30-0**	**.691**	3.93	**1-29**	**1.289**	3.96	**0-29**	**1.299**
iris	4.44	4.52	1-2	1.017	4.59	5-6	1.033	6.52	**1-18**	**1.467**
krk	6.67	6.87	3-10	1.030	7.20	7-20	1.080	7.66	**5-23**	**1.149**
liver	29.33	30.35	12-14	1.035	31.57	9-19	1.077	30.03	**2-16**	**1.024**
lymph	17.78	18.07	9-10	1.017	20.67	**6-21**	**1.162**	19.56	8-18	1.100
musk	13.51	10.97	**19-1**	**.812**	12.02	14-8	.889	11.28	**20-2**	**.835**
retard	32.64	31.66	**24-0**	**.970**	31.34	**21-3**	**.960**	32.30	**14-0**	**.990**
sonar	23.02	22.78	5-5	.990	24.84	5-12	1.079	23.17	1-2	1.007
vote	5.24	4.73	**19-3**	**.903**	4.76	16-7	.908	4.68	**20-5**	**.893**
wave	21.94	22.11	5-10	1.008	24.33	**3-14**	**1.109**	22.11	7-8	1.008
wdbc	4.27	4.27	0-0	1.000	4.71	3-11	1.103	4.30	2-3	1.007

The first column gives the mean error rate of PV. The next three columns compare SCANN to PV using the measures of error rate ("error"), wins and losses ("*w-l*"), and the error ratio of SCANN to PV ("ratio"). A ratio value less than 1 in the "*a* vs. *b*" columns represents an improvement by method *a* over method *b*. The remaining columns report the same measures for S-BP versus PV and S-Bayes versus PV. Table entries comparing method *a* to method *b* are reported in boldface when the difference between the methods is significant at least at the .01 level using a two-tailed sign test.[3]

SCANN posts seven statistically significant wins over PV which are echoed by the measure of wins and losses, and the error ratio comparisons. On average, SCANN reduces error by 7.1% with reductions between 3 to 50%. S-BP has three significant wins over PV and four significant losses. S-BP has an average increase in error over PV by 2.6%. S-Bayes scores five significant wins and losses with an average increase in error of 3.95%.

The most dramatic improvement of the combiners over PV came in the *abalone* data set. A closer analysis of the results revealed that 7 of the 8 learned models were very poor classifiers with error rates around 80%, and the errors of the poor models were highly correlated. This empirically demonstrates PV's known sensitivity to learned models with highly correlated errors. The other combining strategies were able to identify that two of the classes labels were frequently confused. The resulting weighting schemes reversed this

Table 11. Comparison of SCANN to the best individual model and the other combining strategies.

	SCANN	Best Ind. vs. SCANN			S-BP vs. SCANN			S-Bayes vs. SCANN		
Data set	Error	Error	*w-l*	Ratio	Error	*w-l*	Ratio	Error	*w-l*	Ratio
abalone	39.38	42.97	**2-28**	**1.091**	40.08	**8-22**	**1.018**	39.11	19-10	0.993
bal	12.43	12.59	10-16	1.013	11.86	**20-7**	**0.954**	13.71	**7-21**	**1.103**
breast	3.82	4.04	7-15	1.058	3.97	6-8	1.038	3.80	1-0	0.994
credit	13.98	14.75	**7-22**	**1.055**	14.16	9-13	1.013	14.01	6-8	1.002
dementia	32.41	34.35	9-17	1.060	33.98	9-15	1.049	30.56	16-10	0.943
glass	31.69	36.31	**6-22**	**1.146**	36.41	**0-25**	**1.149**	38.21	**1-29**	**1.206**
heart	17.51	17.47	10-17	0.998	18.13	7-15	1.036	17.66	5-7	1.008
ion	2.11	6.64	**1-29**	**3.149**	3.93	**0-29**	**1.866**	3.96	**0-30**	**1.881**
iris	4.52	5.11	11-12	1.131	4.59	4-5	1.016	6.52	**1-18**	**1.443**
krk	6.87	7.72	6-16	1.125	7.20	5-14	1.049	7.66	**5-22**	**1.115**
liver	30.35	33.37	**6-21**	**1.099**	31.57	7-16	1.040	30.03	13-14	0.989
lymph	18.07	17.48	13-10	0.967	20.67	**6-19**	**1.143**	19.56	6-15	1.082
musk	10.97	15.03	**0-21**	**1.371**	12.02	7-12	1.096	11.28	5-11	1.029
retard	31.66	30.57	**24-2**	**0.965**	31.34	15-9	0.990	32.30	6-16	1.020
sonar	22.78	24.13	6-11	1.059	24.84	3-12	1.091	23.17	4-6	1.017
vote	4.73	4.86	10-16	1.027	4.76	9-8	1.005	4.68	4-2	0.989
wave	22.11	26.33	**3-18**	**1.191**	24.33	**3-15**	**1.101**	22.11	8-10	1.000
wdbc	4.27	4.97	5-15	1.164	4.71	3-11	1.103	4.30	2-3	1.007

effect by counting a vote for one of the confused classes as a vote for the other, and vice versa. PV performs well on the *glass*, *lymph* and *wave* data sets where the errors of the learned models are measured (using the κ statistic) to be fairly uncorrelated. Here, SCANN performs similarly to PV, but S-BP and S-Bayes (except for *wave*) appear to be overfitting by making erroneous predictions based on insignificant variations on the predictions of the learned models. This demonstrates SCANN's ability to perform like PV when the errors of the models are less correlated.

Table 11 compares SCANN to the best individual model for each data set, as well as the combining strategies S-BP and S-Bayes. The best individual model was chosen based on the test results and represents the best that an oracle model selection strategy could do. The same measures as in Table 10 are used.

SCANN has seven significant wins and one significant loss (on *retardation*) with respect to the best individual model. This demonstrates SCANN's ability to exceed the performance of any individual model, even when selected via an oracle strategy.

Comparing the other combining strategies to SCANN reveals that S-BP outperforms SCANN on one data set but has five significant losses. Similarly, S-Bayes posts no wins but has five losses. The wins by SCANN are typically on data sets where the most appropriate weighting strategy is plurality voting. SCANN's ability to discard superfluous fluctuations in the models' predictions makes it less sensitive to overfitting than S-BP and S-Bayes.

Table 12. Summary of Section 6.5.

	PV	SCANN vs. PV			Boosting vs. PV			SCANN vs. Boosting	
Data set	Error	Error	*w-l*	Ratio	Error	*w-l*	Ratio	*w-l*	Ratio
abalone	88.29	**38.75**	**30-0**	**0.439**	88.41	11-18	1.001	**30-0**	**0.438**
bal	19.06	**14.96**	**29-0**	**0.785**	19.15	9-13	1.005	**29-0**	**0.781**
breast	4.36	4.34	5-4	0.994	4.41	7-7	1.011	5-5	0.983
credit	14.04	14.04	10-11	1.000	14.15	12-12	1.008	15-8	0.992
dementia	32.22	32.13	11-11	0.997	32.13	7-7	0.997	10-10	1.000
glass	27.03	27.03	13-12	1.000	27.13	11-13	1.004	14-11	0.996
heart	22.64	22.16	11-7	0.979	22.53	10-12	0.995	13-8	0.984
ionosphere	6.29	6.13	9-3	0.975	6.51	7-9	1.035	11-6	0.942
iris	5.33	5.11	4-1	0.958	5.56	1-4	1.042	7-1	0.920
krk	7.86	**9.58**	**5-22**	**1.219**	**8.68**	**3-22**	**1.104**	**7-22**	**1.104**
liver	30.83	**28.62**	**19-5**	**0.928**	29.87	16-5	0.969	18-8	0.958
lymph	17.19	16.89	9-8	0.983	17.19	8-8	1.000	10-12	0.983
musk	9.88	10.44	11-17	1.057	**10.72**	**5-19**	**1.085**	15-12	0.974
retard	36.18	**35.31**	**21-6**	**0.976**	**37.50**	**2-27**	**1.034**	**28-1**	**0.944**
sonar	22.49	22.80	9-13	1.014	23.60	7-17	1.049	14-12	0.966
vote	4.48	4.53	5-8	1.011	4.27	11-5	0.955	4-9	1.060
wave	19.78	19.85	10-16	1.004	19.67	12-8	0.994	11-16	1.009
wdbc	3.29	3.33	9-9	1.012	3.43	5-8	1.041	10-7	0.972

6.5. Experiment 2

Using the same train/test partitions as in Section 6.4, a model set was generated using Boosting and the C4.5 decision tree algorithm. A total of 50 trees were built. SCANN was compared to the standard Boosting combining scheme where each model is assigned a single weight as a function of its performance on its respective training sample.

Thirty trials were conducted using the data sets described above. A summary of the performances of PV, SCANN and Boosting are reported in Table 12. The error percentages for PV are contained in the second column. The columns under 'SCANN vs. PV' contain the error percentages, win/loss tallies, and error ratios (i.e., SCANN/PV) for SCANN versus PV. The same comparisons for Boosting and PV are contained in columns labeled 'Boosting vs. PV'. The last two columns contain a direct comparison of SCANN and Boosting.

In general, SCANN performs better than PV or Boosting. SCANN significantly improves upon PV on four of the data sets, and Boosting on three. Boosting posts three significant losses to PV. As with Section 6.4, the *abalone* data set is where the largest win occurs. The weighting strategies of PV and Boosting do not allow for the identification of confused classes (as described in the previous section) because the model's weight applies to all predictions. On the other hand, on the *krk* data set, SCANN increases error by 21.9% over PV and 10.4% over Boosting. This is one case where SCANN's

weighting scheme was unable to simplify to plurality voting. The overall improvement by SCANN stems from its ability to simplify to plurality voting when the errors are uncorrelated, or to derive a more elaborate combining strategy when the errors contain patterns of correlation.

7. Related work

Work done in combining classifiers can be broken down into two major categories: those which assign a fixed weight to each model, and those which allow the weight for each model to change as a function of the example being classified. Sections 7.1 and 7.2 delineate these approaches and relate them to SCANN.

7.1. Constant weighting functions

When combining classifiers with fixed weights, a model's prediction or vote for a given class has a strength proportional to its assigned weight. The class receiving the most votes is the final class prediction. This is referred to as a *weighted majority* scheme, i.e.,

$$\hat{f}(\mathbf{x}) = \arg\max_{c \in Y} \sum_{i=1}^{N} \alpha_i \| \hat{f}_i(\mathbf{x}) = c \|$$

where Y is the set of possible classes, and $\|a = b\|$ is one if a is equal to b, and zero otherwise.

The simplest way of choosing the weights is giving each model equal weight (i.e., $\alpha_i = 1/N$), and predicting the class with the most frequent vote. This was referred earlier as the plurality vote (PV) and is also known as the basic ensemble method (BEM) (Perrone & Cooper, 1993). This approach has frequently been used as a straw man combining scheme for comparing to other combining schemes (Merz, 1995), or as a simple combining scheme to evaluate model generation strategies (Breiman, 1996; Maclin & Shavlik, 1995). A more elaborate weighting scheme derived by Perrone and Cooper (1993) is the general ensemble method (GEM). GEM is different from SCANN in that models are assigned fixed weights, and GEM has difficultly dealing with models that make highly correlated errors.

One can also combine learned models using logistic regression. For each class, c, denote $P(Y_c \mid \mathbf{x})$ by $\pi(\mathbf{x})$ where $\mathbf{x} = \{x_1, x_2, \ldots, x_N\}$ represents whether classifiers $\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_N$ chose class c. For example, $x_j = 1$ if $\hat{f}_j$ predicted class c. Using the logistic response function,

$$\pi(\mathbf{x}) = \frac{\exp(\gamma + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_N x_N)}{1 + \exp(\gamma + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_N x_N)}$$

and

$$\log \frac{\pi(\mathbf{x})}{1 - \pi(\mathbf{x})} = \gamma + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_N x_N$$

where γ and $\{\beta_i\}$ are constant parameters. Solving for these parameters using a standard approach (i.e., iterative least squares) enables the calculation of the probability of each class. Using these probabilities as the weights, α, the class with the highest probability is selected.

To date, logistic regression has not been applied to learned models with class label output (versus class probabilities). However, Ho, Hull, and Srihari (1994) have successfully applied it to learned models with class rankings with positive results. Due to the large number of free parameters (i.e., $N(M+1)$), this approach is only good for a small number of learned models and classes with plenty of training data.

More ambitious methods incorporate the estimated accuracy of a learned model in choosing its weight. Opitz and Shavlik (1996) do so as follows,

$$\alpha_i = \frac{(1 - E_i)}{\sum_{j=1}^{N}(1 - E_j)}$$

where E_i the estimate of model i's accuracy based on performance on a validation set. Intuitively, model i gets more weight as its estimated performance increases relative to the estimated cumulative performance of the other models. The weight assignment scheme in this work is limited in that it handles redundancy in the model set poorly, i.e., several very similar models will receive the same weight, possibly overpowering the vote of another model making a unique contribution.

Two other methods for assigning fixed weights to each model are Bagging (Breiman, 1994) and Boosting (Schapire, 1990). These methods are tightly coupled to the model generation phase rather than being general combining techniques. The goal is to generate a set of models which are likely to make uncorrelated errors (or to have higher variance) thus increasing the potential payoffs in the combining stage. Each model is generated using the same algorithm, but different training data. The data for a particular model is obtained by sampling from the original training examples according to a probability distribution. The probability distribution is defined by the particular approach.

Bagging exploits the variance of a learning algorithm by applying it to various version of the data set, and averaging the predictions of the models produced uniformly to reduce prediction error due to variance in the models. Variations on the training data are obtained by sampling from the original training data with replacement. The probability of an example being drawn is uniform, and the number of examples drawn is the same as the size of the original training set. The underlying theory of this approach indicates that the models should be weighted uniformly. This approach appears to be effective (Freund & Schapire, 1996; Quinlan, 1996), but may be limited by the particular algorithm being bagged. SCANN is more broadly applicable because it can work with multiple learning algorithms at the same time.

Another resampling method has its roots in what is known as Boosting, initially developed by Schapire (1990). Boosting is based on the idea that a set of moderately inaccurate rules-of-thumb (i.e., learned models) can be generated and combined to form a very accurate prediction rule. Freund and Schapire (1995, 1996) have developed several algorithms for Boosting. This technique assigns a weight to each example in the training data and

adjusts it after learning each model. Initially, the examples are weighted uniformly. For learning subsequent models, examples are reweighted as follows: "easy" examples which are predicted with low error by previously learned hypotheses (i.e., learned models) get lower weight, and "hard" examples that are frequently misclassified are given higher weight. The data sets for each learned model are resampled with replacement according to the weight distribution of the examples.[4]

A combining strategy for Boosting is described in (Freund & Schapire, 1995) AdaBoost.M1 algorithm. The ith model's vote for a given class is a function of its error, ϵ_i, i.e.,

$$\alpha_i = \log \frac{(1 - \epsilon_i)}{\epsilon_i}$$

In this scheme, learned models which make fewer errors (on the distribution of examples they see) get higher weights. Like Bagging, Boosting places more emphasis on generating a diverse model set. It is possible that a more elaborate non-constant weighting scheme like SCANN could improve upon the combining approach above.

Several other resampling techniques have been explored in the literature (Meir, 1995; Krogh & Vedelsby, 1995; Chan & Stolfo, 1995). However, they are not discussed in detail because the emphasis here is on the combining stage.

7.2. Non-constant weighting functions

The most prevalent method in the literature for dynamically deciding how to weight a collection of classifiers is the "mixture of experts" approach (Jacobs et al., 1991) which consists of several different expert learned models (i.e., multilayer perceptrons) plus a gating network that decides which of the experts should be used for each case. Each expert reports a class probability distribution for a given example. The gating network selects one or a few experts which appear to have the most appropriate class distribution for the example. During training, the weight changes are localized to the chosen experts (and the gating network). Experts which classify the example well[5] are given more responsibility for that example and experts which do not classify the example well are given less responsibility. The weights of other experts which specialize in quite different cases are unmodified. The experts become localized because their weights are decoupled from the weights of other experts, and they will end up specializing on a small portion of the input space.

Jordan and Jacobs (1994) expanded on this approach allowing the learned models/experts to be generalized linear models. The experts are leaves in a tree-structured architecture whose internal nodes are gating functions. These gating functions make soft splits allowing data to lie simultaneously in multiple regions. The mixture of experts approach is different than SCANN in that it is more involved in the model generation phase. SCANN deals with the models after they have been learned.

Tresp and Taniguchi (1995) derived a collection of non-constant weighting functions which can be used to combine regressors or classifiers. The proposed methods weigh a learned model according to its reliability in the region of the given example. Reliability is defined in terms of either the model's accuracy in the region of the given example, or

the amount of variability of the model's predictions in that region. All of the approaches require that the weights be positive and sum to one. The methods proposed have not been evaluated empirically, but may prove useful in extending methods like SCANN to allow for more elaborate non-constant weighting functions.

8. Limitations and future work

The most significant limitation of SCANN is that the final combining scheme results in a loss of interpretability. One of the biggest advantages of using a single symbolic learning algorithm is that an interpretable result is produced, such as a decision tree or a rule list. This limitation is common to all approaches to model combination to date. Shannon and Banks (1997) have developed a technique for producing a single interpretable tree from a set of trees, but the technique is limited to decision trees. Producing a single model from a homogeneous model set is a logical place to start working on the interpretability problem. A more ambitious approach would be to try to identify the most reliable rule, decision tree path, etc., from a model set for a given example.

The SCANN algorithm may be extended in several ways. The nearest neighbor component of SCANN could be replaced. A possible substitute combining strategy would be to fit a set of Gaussians functions (Duda & Hart, 1973) (i.e., one per class) to the intermediate representation derived by the correspondence analysis stage. The space derived by CA serves as a good set of attributes for Gaussian models because it is compact and fairly free of noise. This extension could provide class probability output, as opposed to simple class label output, even if all of the constituent models produced class label output.

SCANN is a non-constant weighting scheme in the sense that each model has one weight per class. A more dynamic extension would be to incorporate some of the proposed non-constant weighting schemes of Tresp and Taniguchi (1995) which derive weights according to the example being classified.

In this work SCANN is only applied to learned models which report class labels. The analysis needs to be extended to model sets which report class probabilities and class rankings. The inclusion of probabilistic models may lead to a more robust combining scheme, e.g., as class probabilities more accurately reflect the confidence a learned model has in its predictions. Class probabilities could easily be used by SCANN by filling the indicator matrix with the probabilities instead of a single '1' for the predicted class. Applying SCANN to model sets that report class rankings may also be fruitful for tasks such as character recognition (Ho, Hull, & Srihari, 1994) and information retrieval.

9. Conclusion

A novel method has been introduced for combining the predictions of heterogeneous or homogeneous classifiers. It draws upon the methods of stacking, correspondence analysis and nearest neighbor. In an empirical analysis, the method proves to be insensitive to poor learned models and matches the performance of plurality voting as the errors of the learned models become less correlated.

Acknowledgment

This research was supported by AFOSR grant F49620-96-1-0224.

Notes

1. Henceforth $\mathcal{L}$ will be referred to as $\mathcal{L}_0$ for clarity.
2. Note that j may also be associated the Truth column.
3. The author realizes that significance tests on resampled data have a higher probability of falsely detecting significant differences (Type I error) (Dietterich, 1996; Salzberg, 1997). To guard against this, a high confidence level was used in conjunction with multiple evaluation measures.
4. Note that this resampling technique can be replaced by a reweighting technique when the learning algorithm is capable of directly accepting a *weighted* set of examples.
5. Here, to classify an example well means to have less error than the weighted average of the errors of all the experts (using the outputs of the gating network to decide how to weight each expert's error). To not classify an example well means to have more error than the weighted average.

References

Ali, K., & Pazzani, M. (1995). Learning multiple relational rule-based models. In D. Fisher & H. Lenz (Eds.), *Learning from data: Artificial intelligence and statistics* (Vol. 5). Fort Lauderdale, FL: Springer-Verlag.

Breiman, L. (1994). *Heuristics of instability in model selection* (Technical Report). Department of Statistics, University of California at Berkeley.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Chan, P., & Stolfo, S. (1995). A comparative evaluation of voting and meta-learning on partitioned data. *Proceedings of the 12th International Conference on Machine Learning* (pp. 90–98). Morgan Kaufmann.

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*(4), 261–283.

Cost, S., & Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, *10*(1), 57–78.

Dietterich, T.G. (1996). Statistical tests for comparing supervised classification learning algorithms (Technical Report). Corvallis, OR: Dept. of Computer Science, Oregeon Statue University.

Dongarra, J., & Grosse, E. (1998). Netlib repository.. http://www.netlib.org/.

Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis*. Addison-Wesley.

Efron, B., & Tibshirani, R. (1993). *An introduction to the bootstrap*. London and New York: Chapman and Hall.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*, *121*(2), 256–285. Also appeared in COLT90.

Freund, Y., & Schapire, R.E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. *Proceedings of the Second European Conference on Computational Learning Theory* (pp. 23–37). Springer-Verlag.

Freund, Y., & Schapire, R.E. (1996). Experiments with a new boosting algorithm. *Proceedings of the 13th International Conference on Machine Learning*. Morgan Kaufmann.

Greenacre, M.J. (1984). *Theory and application of correspondence analysis*. London: Academic Press.

Ho, K., Hull, J.J., & Srihari, S.N. (1994). Decision combination in multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-16*(1), 66–75.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J., & Hinton, G.E. (1991). Adaptive mixtures of local experts. *Neural Computation*, *3*(1), 79–87.

Jordan, M.I., & Jacobs, R.A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*, 181–214.

Krogh, A., & Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems* (Vol. 7, pp. 231–238). MIT Press.

Maclin, R., & Shavlik, J.W. (1995). Combining the predictions of multiple classifiers: Using competitive learning to initialize neural networks. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*.

Margineantu, D.D., & Dietterich, T.G. (1997). Pruning adaptive boosting. *Proceedings of the 14th International Conference on Machine Learning*. Morgan Kaufmann.

Meir, R. (1995). Bias, variance and the combination of least squares estimators. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems* (Vol. 7, pp. 295–302). MIT Press.

Merz, C.J. (1995). Dynamical selection of learning algorithms. In D. Fisher & H. Lenz (Eds.), *Learning from data: Artificial intelligence and statistics* (Vol. 2). Springer Verlag.

Merz, C. (1998). *Classification and regression by combining models*. Ph.D. thesis, University of California, Irvine.

Merz, C., & Murphy, P. (1996). UCI repository of machine learning databases. http://www.ics.uci.edu/mlearn/MLRepository.html.

Murthy, S., Kasif, S., Salzberg, S., & Beigel, R. (1993). OC1: Randomized induction of oblique decision trees. *Proceedings of AAAI-93*. AAAI Pres.

Opitz, D.W., & Shavlik, J.W. (1996). Generating accurate and diverse members of a neural-network ensemble. In D.S. Touretzky, M.C. Mozer, & M.E. Hasselmo (Eds.), *Advances in neural information processing systems* (Vol. 8, pp. 535–541). MIT Press.

Perrone, M.P. (1994). Putting it all together: Methods for combining neural networks. In J.D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in neural information processing systems* (Vol. 6, pp. 1188–1189). Morgan Kaufmann Publishers.

Perrone, M.P., & Cooper, L.N. (1993). When networks disagree: Ensemble methods for hybrid neural networks. In R.J. Mammone (Ed.), *Artificial neural networks for speech and vision* (pp. 126–142). London: Chapman & Hall.

Press, W.H. (1992). *Numerical recipes in C: The art of scientific computing* (pp. 59–70). Cambridge University Press.

Quinlan, R. (1993). *C4.5 programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Quinlan, J.R. (1996). Bagging, boosting, and C4.5. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart, J.L. McClelland, & the PDP research group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1: Foundations). MIT Press.

Salzberg, S. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, *1*(3).

Schapire, R.E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197–227.

Shannon, W., & Banks, D. (1997). A distance metric for classification trees. *Preliminary Papers of the Sixth International Workshop on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, Fort Lauderdale, FL.

Tresp, V., & Taniguchi, M. (1995). Combining estimators using non-constant weighting functions. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems* (Vol. 7, pp. 419–426). MIT Press.

Wolpert, D.H. (1992). Stacked generalization. *Neural Networks*, *5*, 241–259.

Received October 3, 1997
Accepted April 21, 1998