Information Sciences 187 (2012) 109–120

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Eigenclassifiers for combining correlated classifiers

Aydın Ulaş [a,*,1], Olcay Taner Yıldız [b], Ethem Alpaydın [a]

[a] Department of Computer Engineering, Boğaziçi University, TR-34342 Bebek, Istanbul, Turkey
[b] Department of Computer Engineering, Işık University, TR-34980 Şile, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 2 March 2010
Received in revised form 8 June 2011
Accepted 16 October 2011
Available online 10 November 2011

Keywords:
Classifier correlation
Classifier design and evaluation
Machine learning

ABSTRACT

In practice, classifiers in an ensemble are not independent. This paper is the continuation of our previous work on ensemble subset selection [A. Ulaş, M. Semerci, O.T. Yıldız, E. Alpaydın, Incremental construction of classifier and discriminant ensembles, Information Sciences, 179 (9) (2009) 1298–1318] and has two parts: first, we investigate the effect of four factors on correlation: (i) algorithms used for training, (ii) hyperparameters of the algorithms, (iii) resampled training sets, (iv) input feature subsets. Simulations using 14 classifiers on 38 data sets indicate that hyperparameters and overlapping training sets have higher effect on positive correlation than features and algorithms. Second, we propose postprocessing before fusing using principal component analysis (PCA) to form uncorrelated *eigenclassifiers* from a set of correlated experts. Combining the information from all classifiers may be better than subset selection where some base classifiers are pruned before combination, because using all allows redundancy.

1. Introduction

Let us say that we have L experts (learners, base classifiers), with their outputs d_j, $j = 1,\dots,L$, estimating some unknown parameter θ, for example, the posterior probability of a class for input x in a classification problem: $P(C_i|x)$. Let us also say that our combined estimator d to θ is the simple average:

$$d = \frac{\sum_{j=1}^{L} d_j}{L}$$

It is known that the mean squared error of d in estimating θ can be written as the sum of squared bias and variance:

$$E[(d-\theta)^2] = (E[d]-\theta)^2 + E[(d-E[d])^2] = \mathrm{Bias}^2(d) + \mathrm{Var}(d)$$

In the case of a simple average, bias is just the average bias:

$$E[d]-\theta = \frac{\sum_{j=1}^{L}(E[d_j]-\theta)}{L}$$

In stacking [46], a second layer learner is trained to combine the outputs of the given classifiers and therefore also corrects for their bias, but in voting methods, the decrease in error is mostly due to the decrease in variance. We can write the variance as

* Corresponding author. Tel.: +90 212 359 4523/4524; fax: +90 212 287 2461.
E-mail address: ulasmehm@boun.edu.tr (A. Ulaş).
[1] The author is currently working in Dipartimento di Informatica, Verona, Italy.

$$\mathrm{Var}(d) = \frac{1}{L^2}\sum_j \mathrm{Var}(d_j) + \frac{1}{L^2}\sum_i \sum_{j\neq i} \mathrm{Cov}(d_i, d_j) \tag{1}$$

If d_j are independent, then their correlation is 0 and the second, covariance term in (1) disappears. In such a case, variance decreases as L is increased. Indeed, most combination methods aim at generating uncorrelated experts, and it has been proposed [21] to use different (i) learning algorithms, (ii) hyperparameters, (iii) input features, and (iv) training sets. For example, Bagging [7] uses bootstrapping to generate slightly different training sets and takes an average for fusion, and it has been observed that the decrease in error is due to a decrease in variance. The random subspace method [16] trains different experts with different subsets of a given feature set. Different representations of the same input make different characteristics explicit and therefore accuracy may be improved by combination [1,11]. It has also been proposed to generate an ensemble of fuzzy decision trees and take their combination for better accuracy [45]. In measuring the performance of a machine learning algorithm, accuracy is not always the sole criterion. There are other measures such as the area under ROC curve (AUC) or F-measure (based on precision and recall); recently, Wang and Dong [44] applied fuzzy entropy maximization instead of training error minimization to refine fuzzy IF-THEN rules. Tong and Mitram [39] use genetic algorithms to choose between neural network activation functions and features to increase classifier performance. Wang [43] uses mutual information for feature selection in combining nearest neighbor classifiers using the fuzzy integral. Ensemble algorithms have been proposed and applied to different problems in different domains [48,47,26,52].

Normally, when we have a number of experts trained on the same data, we expect them to be positively correlated, i.e., they will be correct on the same instances and fail on the same difficult (noisy) instances. Looking at (1), we expect then the variance (and hence the error) to increase as we increase the number of experts. It is therefore critical that any positive correlation between the experts should be found and taken care of.[2]

Given a set of positively correlated experts, one line of research is in the direction of finding a minimal subset of them as we studied in our previous work [42]. That is, we are interested in both pruning the inaccurate ones and also to keep a check on complexity, we want to prune the redundant. "Diversity" measures have been proposed [23,22] and one possibility is to have an incremental, forward search where we add a classifier to an ensemble if it is diverse or adds to accuracy [9,11,35,49,42], or another possibility is to have a decremental, backward search where a classifier is removed or pruned if it is not diverse enough or if its removal does not increase error [30,27]. In this work, we propose an alternative method which combines base classifiers using principal component analysis (PCA) to get uncorrelated *eigenclassifiers*.

This current work complements and extends our previous work [42] in two directions: (i) Here, we empirically analyze the sources of correlation using real data sets and show how the base classifiers trained using these methods differ in their decisions, and (ii) instead of applying subset selection, that is, keeping some base classifiers and discarding the rest, we opt to use all the available information by constructing eigenclassifiers that combine all base classifiers. When subset selection is used, some classifiers/discriminants may be discarded and this results in simpler combinations; it may also prevent extracting features/classifiers which are costly. Unfortunately, it also has the disadvantage of discarding potentially useful information. Here, we use all the base classifiers and hence all the information they provide is incorporated in the final decision.

The rest of this paper is organized as follows: The influence of different factors on correlations between experts using fourteen classifiers and thirty-eight data sets is investigated in Section 2. In Section 3, we propose a method to extract uncorrelated eigenclassifiers from a set of correlated classifiers and show how it works in practice. We conclude in Section 4.

2. Correlation analysis on real data sets

2.1. Algorithms and data sets

We use fourteen base classifiers [42] (trained using six algorithms) which we have chosen to span as much as possible the wide spectrum of possible machine learning algorithms. All classifiers generate posterior probabilities as their output.

(1)–(3) *knn*: k-nearest neighbor with $k = 1,3,5$.
(4)–(8) *mlp*: Multilayer perceptron where with D inputs and K classes, the number of hidden units is taken as D (*ml1*), K (*ml2*), $(D + K)/2$ (*ml3*), $D + K$ (*ml4*), $2(D + K)$ (*ml5*).
(9) *lnp*: Linear perceptron with softmax outputs trained by gradient-descent to minimize cross-entropy.
(10) *c45*: The most widely-used C4.5 decision tree algorithm.
(11) *mdt*: This is a multivariate tree where unlike C4.5 which uses univariate and axis-orthogonal splits uses splits that are arbitrary hyperplanes using all inputs [50].
(12)–(14) *svm*: Support vector machines (SVM) with a a linear kernel (*svl*), polynomial kernel of degree 2 (*sv2*), and a radial (Gaussian) kernel (*svr*). We use the LIBSVM 2.82 library that implements pairwise SVMs [10].

[2] Examining (1), we see that in minimizing variance, even better than the case of uncorrelated experts would be the case when we have negatively correlated experts [24]. Note however that for the case of mixture of experts, it has been shown that experts which are localized in different parts of the input space are negatively correlated but biased [17]. Comparing AdaBoost [13] with Bagging, we can say that experts trained on previous expert's errors not only decrease bias but also help in constructing negatively correlated experts.

We use a total of 38 data sets where 35 of them (*zoo*, *iris*, *tae*, *hepatitis*, *wine*, *flags*, *glass*, *heart*, *haberman*, *flare*, *ecoli*, *bupa*, *ionosphere*, *dermatology*, *horse*, *monks*, *vote*, *cylinder*, *balance*, *australian*, *credit*, *breast*, *pima*, *tictactoe*, *cmc*, *yeast*, *car*, *segment*, *thyroid*, *optdigits*, *spambase*, *pageblock*, *pendigits*, *mushroom*, *and nursery*) are from UCI [4] and 3 (*titanic*, ringnorm, and twonorm) are from Delve [31] repositories.

A given data set is first divided into two parts, with 1/3 as the test set, *test*, and 2/3 as the training set, *train-all*. This training set is then resampled using 5×2 cross-validation (cv) where 2-fold cv is done five times (with stratification) and the roles swapped at each fold to generate ten training and validation folds, tra_i, val_i, $i = 1,\ldots,10$. tra_i are used to train the base classifiers. val_i are divided into two randomly as $val\text{-}A_i$ and $val\text{-}B_i$, where $val\text{-}A_i$ are used to train the linear combiner to fuse base classifier outputs and $val\text{-}B_i$ are used for model selection (i.e. finetuning the complexity of the combiner in Section 3). We always report the accuracies on *test*, unused for training the base classifiers, combiner or model selection. This processed data of base classifier outputs is publicly available [51].

To compare the accuracies of different ensemble construction methods for statistically significant difference, we use two different methodologies. First, for each data set, we use the 5×2 cv F test [3] ($\alpha = 0.05$) which is a parametric test to compare the methods for each data set; we then use the sign test to check if the numbers of wins/losses over all 38 data sets is significant. Second, we use Friedman's test which is a nonparametric test using the rankings, and if it rejects, we use the Nemenyi test as a post hoc test to check for significant difference between methods [12,15].

In the literature, to increase diversity and reduce correlation between learners, it has been proposed to play with four factors:

1. Learning algorithms used to train the experts.
2. Hyperparameters of the learning algorithms.
3. Resampling due to folding.
4. Subset of input features.

In the next subsection, we check for the effect of these factors experimentally using several data sets.

2.2. Estimating the correlations of classifiers

We train all fourteen algorithms on a training fold, generate their posterior probabilities for the true class on the test set and calculate correlations between classifier outputs for the true class. The total correlation between two learners i and j is the sum of correlations on the individual instances

$$\text{Total Corr}(i,j) = \sum_t \text{Corr}\left(d^t_{i*}, d^t_{j*}\right) \tag{2}$$

where d^t_{i*} denotes the output of learner i for the correct class on instance t. The reason we use only the output for the true class and not all classes is to (i) be able to average correlations over data sets with different number of classes, and (ii) keep the analysis simpler. We do this ten times on the ten training folds and calculate the average test correlation for a data set. We then do this for all 38 data sets and take another average to give a general, data-independent view. This overall correlation matrix is given in Table 1. We believe that a correlation value over 0.6 indicates a strong correlation and such entries are shown in boldface.

2.2.1. Correlations due to hyperparameters

Almost all learning algorithms have hyperparameters which affect the model complexity and we check for the effect of these on correlation. Looking at the top-left corner of Table 1, we see the overall correlation matrix achieved by varying k of

Table 1
Average correlations over all data sets.

	kn1	*kn3*	*kn5*	*ml1*	*ml2*	*ml3*	*ml4*	*ml5*	*lnp*	*mdt*	*c45*	*svl*	*sv2*	*svr*
kn1	1.00	**0.71**	**0.64**	0.37	0.37	0.37	0.38	0.37	0.38	0.35	0.30	0.39	0.34	0.44
kn3	**0.71**	1.00	**0.88**	0.51	0.50	0.51	0.51	0.51	0.51	0.45	0.41	0.53	0.45	0.58
kn5	**0.64**	**0.88**	1.00	0.57	0.56	0.57	0.57	0.57	0.55	0.49	0.45	0.59	0.49	**0.64**
ml1	0.37	0.51	0.57	1.00	**0.79**	**0.81**	**0.81**	**0.79**	**0.67**	0.59	0.52	**0.75**	0.53	**0.69**
ml2	0.37	0.50	0.56	**0.79**	1.00	**0.81**	**0.79**	**0.77**	**0.66**	**0.62**	0.54	**0.76**	0.52	**0.69**
ml3	0.37	0.51	0.57	**0.81**	**0.81**	1.00	**0.81**	**0.81**	**0.67**	**0.61**	0.53	**0.75**	0.53	**0.70**
ml4	0.38	0.51	0.57	**0.81**	**0.79**	**0.81**	1.00	**0.80**	**0.67**	**0.61**	0.53	**0.75**	0.52	**0.69**
ml5	0.37	0.51	0.57	**0.79**	**0.77**	**0.81**	**0.80**	1.00	**0.67**	**0.60**	0.52	**0.75**	0.53	**0.69**
lnp	0.38	0.51	0.55	**0.67**	**0.66**	**0.67**	**0.67**	**0.67**	1.00	0.57	0.48	**0.71**	0.45	**0.63**
mdt	0.35	0.45	0.49	0.59	**0.62**	**0.61**	**0.61**	**0.60**	0.57	1.00	0.50	**0.64**	0.45	**0.60**
c45	0.30	0.41	0.45	0.52	0.54	0.53	0.53	0.52	0.48	0.50	1.00	0.54	0.43	0.54
svl	0.39	0.53	0.59	**0.75**	**0.76**	**0.75**	**0.75**	**0.75**	**0.71**	**0.64**	0.54	1.00	0.57	**0.74**
sv2	0.34	0.45	0.49	0.53	0.52	0.53	0.52	0.53	0.45	0.45	0.43	0.57	1.00	**0.65**
svr	0.44	0.58	**0.64**	**0.69**	**0.69**	**0.70**	**0.69**	**0.69**	**0.63**	**0.60**	0.54	**0.74**	**0.65**	1.00

knn. We notice that varying *k* has a small effect on removing this intragroup correlation. This indicates that if you already have *3nn* in your ensemble, adding *5nn* is not a good idea; because they are highly correlated, addition would not increase accuracy significantly. This observation also supports our previous results when we use subset selection [42]. We rarely see *knn* algorithms with different k values ending up in the same ensemble when we use subset selection or in the optimal subset. As other examples, we see a similar behavior when we vary the number of hidden units of *mlp* or the degree of polynomial kernel of *svm*, though in this latter case, we see that the classifiers are less correlated when compared with *knn* variants.

2.2.2. Correlations due to algorithms

There is also correlation depending on how similar the algorithms are: We see that the perceptron variants (*lnp* and *mlp*), linear models (*lnp*, *svl*), and the *svm* variants (*svl*, *sv2*, *svr*) are correlated. We see a clear case of grouping here: The variants of the same algorithm are grouped with high intragroup correlation and we also see lower but still positive intergroup correlation which makes them more unlikely to end up in a carefully constructed ensemble [42]. For example, there is correlation between *mlp* and *svm* variants, because they both write the discriminant as a sum of multivariate basis functions (hidden units or implicitly through the kernel function). The correlation between classifier groups decrease as they are less similar in terms of the models they use, the criteria they optimize, or the method they use for optimization.

2.2.3. Correlations due to sampling

In order to figure out the correlations due to resampling, we calculate the correlation of the fourteen classifiers on the test set, trained on different training folds and average over them (over $\binom{10}{2}$ fold pairs), and average once more over 38 data sets. The boxplots of correlations for the fourteen algorithms are given in Fig. 1. We see that trees, *c45* and *mdt*, have low correlations and *mlp*, *knn* and *svm* variants have high correlations (except *1nn*); this shows that it makes sense to bag (fuse) trees but not *knn*, which is indeed actual practice. Breiman [7] mentions this when he defines the concept of stable algorithms and he says that it makes sense to bag unstable algorithms such as trees but not stable algorithms such as *knn*.[3] We have checked to see if correlation depends on training set size, but there seems to be no dependence.

2.2.4. Correlations due to shared input features

In order to figure out the correlations due to input features used, we calculate the correlation of the fourteen classifiers trained on randomly chosen half of the original features, but on the whole training set (without folding). Doing this ten times each time choosing a different subset, we average over the $\binom{10}{2}$ pairs on a data set, and average once more over 38 data sets. The boxplots of correlations for the fourteen algorithms are given in Fig. 2. We can see that there is not much difference between the algorithms and that the correlation can be anywhere between 0.0 and 1.0, mostly around 0.5. We cannot say much that is general from these results; this random subspace method [16] can be an effective method (more effective than resampling or varying the hyperparameter) for generating less correlated experts but there is no guarantee. We have checked to see if the correlation depends on the input dimensionality; but it seems not to.

2.3. Related work

Tumer and Ghosh [40] give a theoretical analysis of the error using correlated classifiers and analyze the correlation between ensembles using different diversity creation methods. Using four real world data sets, they measure the effect of the following factors on the correlation between classifiers in the ensemble: (i) cross validation, (ii) different feature subsets, and (iii) resampled data. They do not investigate the effect of different algorithms nor hyperparameters of algorithms. Brown et al. [8] discuss the relation between ambiguity decomposition and the bias/variance/covariance decomposition, presenting several methods to create diverse classifiers. Theirs is not an empirical study, but they create a taxonomy of ways to create diverse classifiers. They propose to change (i) starting point in the hypothesis space, (ii) training data, (iii) classifier architectures, and (iv) traversal of the hypothesis space. In an early work, Kuncheva [20] proposes four different methods to build a multiclassifier system: (i) using different combiners when already trained base classifiers are given, (ii) using different algorithms and starting parameters, (iii) using different feature subsets, and (iv) using different training sets.

3. Extracting eigenclassifiers for aggregate decisions

3.1. Constructing new uncorrelated eigenclassifiers

Given a set of positively correlated base classifiers, the usual approach is to keep a subset, pruning those that are correlated with those in the subset. However, unless there is perfect correlation (of 1.0), this causes a loss of information as those base classifiers which can potentially help in new cases are removed, decreasing fault tolerance. The approach we propose is

[3] *knn* can be made unstable by condensing [2] or random feature selection [5].

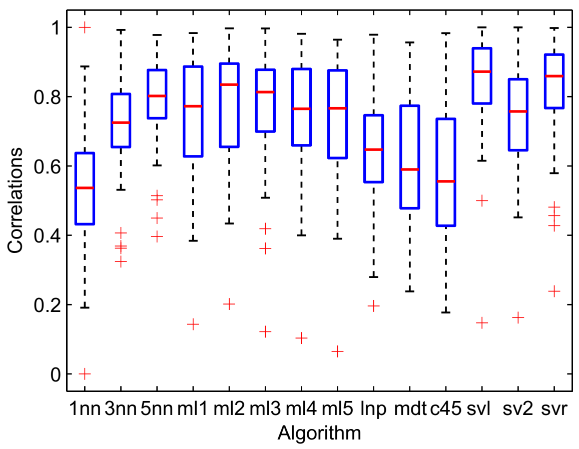

Fig. 1. The boxplot of correlations between folds of the fourteen algorithms averaged over test sets of all data sets.

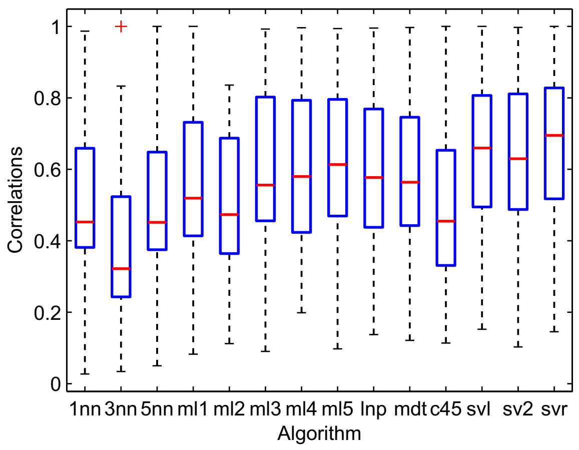

Fig. 2. The boxplot of correlations between random input feature subsets of the fourteen algorithms averaged over test sets of all data sets.

to keep all the base classifiers even if there is correlation between them (as opposed to selecting a subset [42]) but combine their predictions taking into account the fact that they are not independent.

We consider the outputs of base classifiers as dimensions of a new feature space in which a new classification problem is defined, and we view the problem of combining base classifier outputs as the problem of choosing input features. With this view in mind, when we are interested in extracting the best features (that is, choose the most informative base classifier values), one way is to do *feature selection*, where we keep some of the features and discard the rest. Actually, methods where people choose a subset of base classifiers from a large ensemble of candidates [27,34,9,35,11,49,42] do exactly this.

In this paper, we advocate the other approach of combining the base classifier outputs to define new *eigenclassifiers*. This is similar to *feature extraction* in pattern recognition where we define new features that are combinations of the original features. In particular, we use PCA which defines new aggregate dimensions which we can name *eigenclassifiers*, that are linear combinations of the original features, which in our case correspond to outputs of the base classifiers.

We use PCA as follows: Given the 14 classifiers trained on the training fold tra_i, we calculate their outputs for the true class as a 14-dimensional vector on $val\text{-}A_i$ and their correlation matrix, which is a 14×14 matrix. The leading M eigenvectors of this matrix are the new eigenclassifier directions. We map the original input to this space by first calculating classifier outputs and then multiplying all the K class outputs of a classifier by the corresponding value of these eigenvectors. The output is then used to train a linear classifier in this new space, using again $val\text{-}A_i$. The input to the linear combiner (which is a linear perceptron in our case) has $M \cdot K$ dimensions where we decide on M, the number of eigenvectors (components, eigenclassifiers), based on the average accuracy on $val\text{-}B_i$, the other half of the validation fold (unused during training of the base classifiers or the linear combiner). We report and check for significant difference on the accuracies of the ten folds on the *test*.

One advantage of a linear combiner is that there is no need for scaling or any other normalization [25,18]. When there are a number of groups with different intragroup and intergroup correlations, a trained linear combiner works better than any fixed rule in decreasing error [41]. The superiority of the linear perceptron over other (linear) combiners has been shown by Raudys [32].

The eigenvectors of the correlation matrix can also be analyzed for information extraction. In Fig. 3, we show the first five eigenvectors of the correlation matrix averaged over all data sets (Table 1) (these are not the eigenvectors used; actually, for each data set, in each fold, there is a different correlation matrix and a different set of eigenvectors). The numbers in the top right corner of the figure is the proportion of variance explained by the components up until then.

In all data sets, we always see that the first eigenvector is a vector of roughly equal positive values; doing a dot product with it is almost like a simple averaging. It is known [33] that if we have a covariance matrix where all variances are equal and all correlations are also of similar magnitude, the first principal component is proportional to the mean of the input variables. In our case, the proportion of variance explained by the first eigenvector is 0.62 on the average, and this value is higher as the base classifiers become more correlated. We interpret this as follows: Even when the classifiers are correlated, the best aggregate decision is to have a simple average and this gives us more than half of the information provided by the classifiers, as measured by the proportion of variance explained. A similar conclusion has also been reached by Fumera and Roli [14], where they say that "simple averaging is the optimal linear combining rule only if the individual classifiers exhibit identical error rates and identical correlations between estimation errors." This also implies that in cases when this is not true, taking only an average corresponds to discarding the variance carrying dimensions that the other components represent.

We can extract more information by looking at the eigenvectors that follow (see Fig. 3): In the second one, we have all *knn* variants with positive weights, perceptrons with negative weights, and all others are close to 0; we interpret this as the nearest neighbor dimension. In the third, *svms* and the two trees (*c45* and *mdt*) have positive weights, perceptrons have negative weights and the others have small negative weights; this is the *svm*-tree dimension. The fourth separates trees from *svms*. Once we make a distinction between the major groups (and having explained 81% of the variance), the fifth separates *lnp* from other perceptrons, and univariate and multivariate trees.

We compare our results using PCA with four other methods:

- BEST is the accuracy of the most accurate single base classifier.
- ALL is the accuracy when all fourteen base classifiers are combined.
- OPT is the accuracy of the optimum subset, that is, the one found by exhaustive search over all 2^{14} possible subsets.
- ACC is the accuracy of the subset constructed by the ICON [42] variant which uses accuracy as the model selection criteria and forward search as the search direction. The algorithm starts with the most accurate classifier on *val-B*, and adds classifiers to the ensemble one by one, until all classifiers are used, or average *val-B* accuracy does not increase. The algorithm starts with $E^0 \leftarrow \emptyset$, then at each step t, all the base classifiers $M_j \notin E^{(t-1)}$ are combined with $E^{(t-1)}$ to form S_j^t. We select $S_{j^*}^t$ which is the ensemble with the highest accuracy. If accuracy of $S_{j^*}^t$ is higher than $E^{(t-1)}$, we set $E^t \leftarrow S_{j^*}^t$ and continue, else the algorithm stops and returns $E^{(t-1)}$. This implements the forward search; backward search starts with all the base classifiers and prunes classifiers until accuracy decreases.

Note that as with PCA, there are linear perceptrons trained to combine the outputs of the base classifiers also with BEST, OPT, ACC, and ALL, and they are also trained on *val-A* folds.

3.2. Case study: pageblock data set

Before proceeding to our complete results on all 38 data sets, we start by presenting our results on one data set, *pageblock*, in more detail, to get an initial feel.

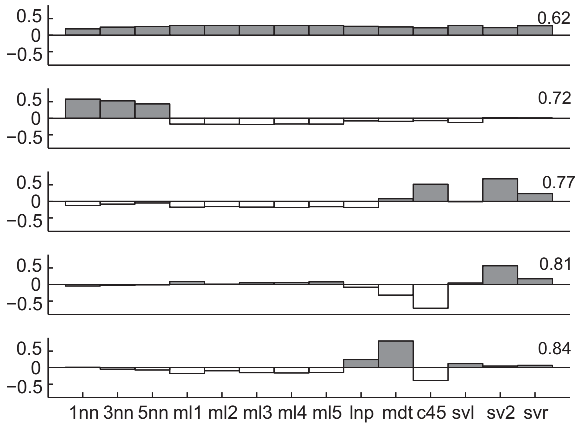

Fig. 3. The first five eigenvectors of the correlation matrix of all fourteen classifiers averaged over all data sets, shown in Table 1. The gray boxes show positive dimensions, and the white boxes show negative dimensions. The numbers on the top-right of each subfigure represents the proportion of variance explained up to and including that eigenvector.

The correlation matrix is given in Table 2 and the first five eigenvectors of the correlation matrix is given in Fig. 4. This is a data set where the base classifiers are strongly correlated and the first component explains 79% of the variance. The second separates *knn* variants from all others and the third separates *c45* from *mlp* variants. The fourth one makes a distinction between single-layer and multilayer perceptrons and *svm* with polynomial and Gaussian kernels. The fifth one separates univariate and multivariate trees.

We see the error of the Pca method compared to those of Best, Opt, Acc, and All in Fig. 5, where we see that accuracy increases by including more components. On this data set, Pca, Opt, Acc, and All are significantly more accurate than Best. Pca with four eigenclassifiers is as accurate as All. Note that Opt is not the most accurate, because the optimum is chosen according to *val-B* error, but Fig. 5 shows the errors on *test*. On this data set where the first component explains 79% of the variance, average rule is as accurate as the trained linear rule.

3.3. Overall results

Comparing our Pca method with Best, All, Acc, and Opt on all 38 data sets, we get the pairwise results in Table 3. There is no significant difference between Pca, All, Acc, and Opt but all four are significantly more accurate than Best. All is not more accurate than Pca, or the other way around. Friedman's test rejects that the five methods have equal ranks. Doing Nemenyi's post hoc test, we get the results in Fig. 6. The test finds that Pca, Opt, Acc, and All belong to one group, and this group is significantly more accurate than Best. We see that Pca and All have the same accuracy, where generally a few components are enough with Pca; see Fig. 7 for the histogram of number of components used by Pca on 38 data sets. Using the top few eigenvectors of Pca, instead of all, has a smoothing effect, because it gets rid of noisy attributes.

It can be said that the major advantage of a subset selection method over Pca is that once a subset is chosen during training, afterwards during test, not all, but only those in the subset need be evaluated, whereas Pca needs to calculate all base classifier outputs before doing the dot product and calculate the eigenclassifier outputs. However the disadvantage of subset selection is that once a number of classifiers are pruned, their potentially useful contribution is also removed; Pca keeps the

Table 2
Average correlations on the *pageblock* data set.

	kn1	*kn3*	*kn5*	*ml1*	*ml2*	*ml3*	*ml4*	*ml5*	*lnp*	*mdt*	*c45*	*svl*	*sv2*	*svr*
kn1	1.00	**0.80**	**0.74**	0.59	0.57	0.57	0.57	0.59	0.55	0.53	0.50	0.56	0.52	0.59
kn3	**0.80**	1.00	**0.95**	**0.76**	**0.74**	**0.75**	**0.74**	**0.77**	**0.71**	**0.69**	**0.61**	**0.73**	**0.68**	**0.77**
kn5	**0.74**	**0.95**	1.00	**0.81**	**0.79**	**0.79**	**0.78**	**0.81**	**0.74**	**0.74**	**0.65**	**0.77**	**0.73**	**0.83**
ml1	0.59	**0.76**	**0.81**	1.00	**0.93**	**0.94**	**0.92**	**0.92**	**0.82**	**0.79**	**0.68**	**0.89**	**0.78**	**0.86**
ml2	0.57	**0.74**	**0.79**	**0.93**	1.00	**0.95**	**0.93**	**0.91**	**0.79**	**0.81**	**0.70**	**0.89**	**0.78**	**0.85**
ml3	0.57	**0.75**	**0.79**	**0.94**	**0.95**	1.00	**0.94**	**0.93**	**0.81**	**0.79**	**0.69**	**0.88**	**0.76**	**0.86**
ml4	0.57	**0.74**	**0.78**	**0.92**	**0.93**	**0.94**	1.00	**0.92**	**0.82**	**0.79**	**0.67**	**0.88**	**0.77**	**0.85**
ml5	0.59	**0.77**	**0.81**	**0.92**	**0.91**	**0.93**	**0.92**	1.00	**0.84**	**0.80**	**0.67**	**0.88**	**0.77**	**0.87**
lnp	0.55	**0.71**	**0.74**	**0.82**	**0.79**	**0.81**	**0.82**	**0.84**	1.00	**0.73**	0.59	**0.85**	**0.76**	**0.80**
mdt	0.53	**0.69**	**0.74**	**0.79**	**0.81**	**0.79**	**0.79**	**0.80**	**0.73**	1.00	**0.60**	**0.81**	**0.78**	**0.83**
c45	0.50	**0.61**	**0.65**	**0.68**	**0.70**	**0.69**	**0.67**	**0.67**	0.59	**0.60**	1.00	**0.68**	**0.63**	**0.66**
svl	0.56	**0.73**	**0.77**	**0.89**	**0.89**	**0.88**	**0.88**	**0.88**	**0.85**	**0.81**	**0.68**	1.00	**0.90**	**0.88**
sv2	0.52	**0.68**	**0.73**	**0.78**	**0.78**	**0.76**	**0.77**	**0.77**	**0.76**	**0.78**	**0.63**	**0.90**	1.00	**0.82**
svr	0.59	**0.77**	**0.83**	**0.86**	**0.85**	**0.86**	**0.85**	**0.87**	**0.80**	**0.83**	**0.66**	**0.88**	**0.82**	1.00

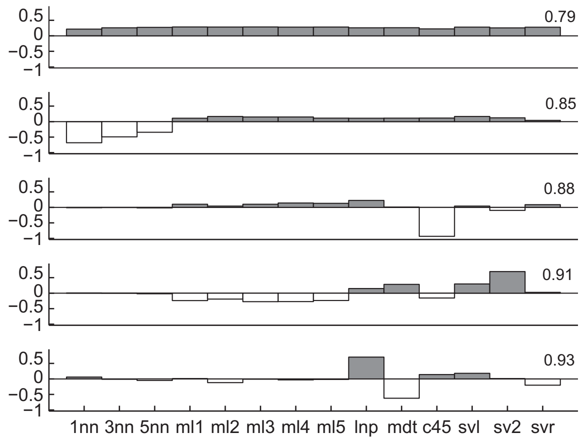

Fig. 4. First five eigenvectors of the correlation matrix on the *pageblock* data set.

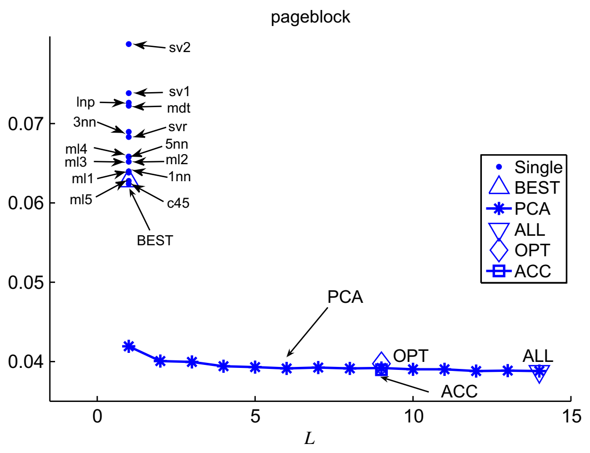

Fig. 5. Classification errors of base classifiers, PCA, OPT, ACC, and ALL on *pageblock*.

Table 3
Pairwise comparisons of BEST, PCA, OPT, ACC, and ALL.

	BEST	PCA	ALL	OPT	ACC
BEST	0	1	1	1	1
PCA	**8**	0	0	0	3
ALL	**8**	0	0	0	2
OPT	**13**	4	2	0	2
ACC	**16**	4	2	1	0

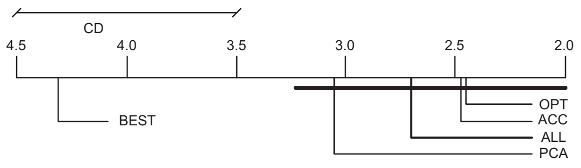

Fig. 6. Graphical representation of post hoc Nemenyi test, as proposed in [12]. The numbers on the line represent the average ranks, CD is the critical difference for statistical significance, and bold lines connect the methods which have no significant difference.

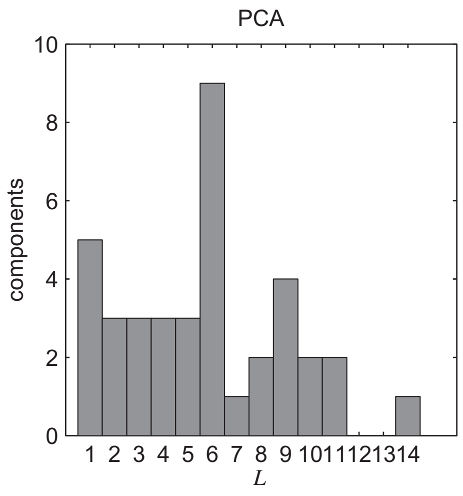

Fig. 7. Histogram of the number of components used by PCA on all 38 data sets.

whole classifiers and hence is more fault tolerant. Note that if the aim is feature *selection*, there are methods which use PCA for this purpose [19]: We can decrease the ensemble size before doing Pca; for example, given that *c45* and *mdt* are both in the tree dimension, or, members of the tree "family," we can get rid of one, increasing the weight of the other.

3.4. Comparison with AdaBoost and bagging

We compare our proposed algorithms with AdaBoost [13] and Bagging [7]:

- Ada: Standard AdaBoost implemented as proposed in [13]. We make a slight change and instead of stopping at 100% accuracy, we reset the instance weights and continue to get more diversity. We train decision tree ensembles of size 5, 10, 15, 20, 25, 30 and choose the one with the best *val-B* accuracy.
- Bag: The original Bagging algorithm proposed in [7]. We train decision tree ensembles of size 5, 10, 15, 20, 25, 30 and choose the one with the best *val-B* accuracy.

We compare accuracies of these methods in a pairwise manner on *test* in Table 4. These are the number of significant wins and losses of method in the row over the method in the column. The sum of wins and losses subtracted from 38 gives the number of ties. If the entry is bold, this means that the number of wins/losses over 38 is statistically significant using the sign test. We see that Pca is significantly more accurate than both Ada and Bag, and Ada are significantly more accurate than Bag. If we use a nonparametric test on the average ranks of the four ensemble methods on 38 data sets (Table 5), we see that Friedman's test rejects the hypothesis that the four methods have equal ranks. Doing Nemenyi's post hoc test for pairwise comparison, we get the results in Fig. 8. We see that we get similar results as the sign test, this time, the three methods Pca, Acc and Ada have comparable accuracy and are significantly more accurate than Bag.

In Table 6, we can see the average and standard deviation of number of classifiers used by the compared ensemble methods. We can see that, though Pca and Ada have comparable accuracies when we use the nonparametric tests, Pca uses fewer classifiers. This is because Ada uses the same base classifier, and to create diversity needs more classifiers, whereas Pca uses different base classifiers and profits from their diversity.

Table 4
Pairwise comparison of accuracies (wins/losses over 38) of ensemble methods using 5×2 cv *F*-test.

	Pca	Acc	Ada	Bag
Pca	0	3	**10**	**11**
Acc	4	0	9	**11**
Ada	1	2	0	**10**
Bag	0	0	0	0

Table 5
Average ranks of ensemble methods.

Pca	Acc	Ada	Bag
1.92	1.76	2.45	3.89

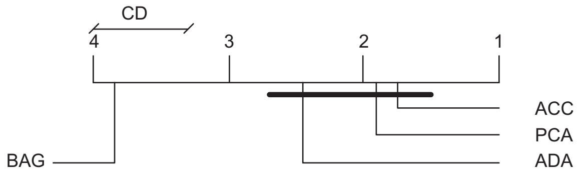

Fig. 8. Graphical representation of post hoc Nemenyi test results of compared methods with ranks given in Table 5.

Table 6
Average and standard deviation of number of base classifiers used by different ensemble methods.

Pca	Acc	Ada	Bag
5.68 ± 3.25	3.95 ± 2.62	22.10 ± 8.11	10.13 ± 8.01

3.5. Related work

3.5.1. Selecting a subset of classifiers

Instead of using all, choosing a subset may lead to higher accuracy and decreased complexity where the idea is to weed out the inaccurate or redundant (having low diversity) classifiers [54,9,35,42]. If the number of base classifiers is not high, a subset can be found by exhaustive search [36]. Otherwise heuristic methods are preferred: forward search [9,11,35,49,42], backward search [30,27], genetic algorithms [30,54,35], or optimization algorithms [37,53]. Greedy heuristic algorithms, such as forward and backward search using accuracy as the model selection criteria, are known to find ensembles having accuracies as high as the optimal subset constructed by exhaustive search [35,42]. Forward subset selection has been shown to outperform classical combination schemes such as Bagging and boosting [9].

Several studies to compare subset selection algorithms were conducted which use various search strategies, classification algorithms, model selection criteria and data sets [9,35,49,42]. It has been shown that using diversity alone is not a good indicator of ensemble accuracy, one should use accuracy or some combination of accuracy and diversity, as the model selection criterion [35,42]. Of the search strategies, forward and backward searches find ensembles which have the same accuracy, but forward search is faster, finds simpler ensembles because backward search may stop early especially when the number of base classifiers is high [35,42]. Floating search does not add to the accuracy of forward search, because most of the time the chosen subsets have a small number of base classifiers, and pruning a previously added classifier is not necessary [42], thus, forward search should be preferred as the search strategy. The chosen subset can be combined using voting or a trained combiner. The trained combiner may be needed if some of the base classifiers are inaccurate and need to be given smaller weights. Most of the time a trained combiner is not necessary when subset selection is used [42]. Moreover, one needs to have separate data to train the combiner, and this may cause overfitting if the number of classes and number of base classifiers is high.

3.5.2. Algorithms decorrelating base classifier outputs

All methods which consider the outputs of previously trained base learners as inputs to a new learner are variants of stacking [46,38] and so are the methods we discuss above. A subset selection algorithm has the potential to remove classifiers that may contribute to accuracy, but our post-processing approach considers all the classifiers. Merz [28,29] discuss work that are most similar to ours. Merz [28] proposes the SCANN algorithm that uses correspondence analysis on the crisp outputs of base classifiers and combines them using the nearest mean classifier. This corresponds to doing PCA on the 0/1 outputs of the base classifiers and one can show that the nearest mean classifier is also a linear classifier. In his work, Merz uses neural networks, rule lists, decision trees and nearest neighbor classifiers as base classifiers. The combination results are compared with voting, and using naive Bayes and multilayer perceptron as stackers. In theory, numeric outputs give more information and are preferred [38], but in our experiments, we have found no difference between using 0/1 or continuous outputs; this probably is because our outputs are posterior probabilities (normalized using softmax) and are close to 0/1 anyway. Note however that because we use a trained linear combiner and not a fixed rule (e.g., sum, max, etc.), there is no need that the base classifier outputs be normalized. Note also that Merz works on the full $14 \cdot K$ dimensions instead of 14.

Merz and Pazzani [29] propose the PCR* algorithm which uses PCA on the outputs of base regressors. They show that the algorithm is able to cope with the inherent correlations amongst the base regressors, which is what we try to achieve. They compare their proposed method with other regression combination algorithms in the literature. After the reduction of the output dimensionality using PCA, they also use a linear combiner for the final decision. Since theirs is a regression problem, there is only one output in their study. As future work, they write that their algorithm can be used in classification problems using a separate model for each class. The method we propose pools all class information and learns a single set of eigenclassifiers; we believe that this is better than learning a separate set of eigenclassifiers for each class because it is a simpler solution, has less parameters, pools data, and therefore has less risk of overfitting.

4. Discussion and conclusions

This paper has two parts where the first part investigates the source of correlation between learners and the second part proposes a method to remove the possibly harmful effect of such correlation.

We test the effect of four factors on the correlation between classifiers, namely, similarity in the algorithms, hyperparameters, overlapping folds, and shared input features, and see that no matter how we may vary algorithms, hyperparameters, folds, or inputs, we still get positively correlated classifiers, and postprocessing is needed to make them uncorrelated. This correlation analysis allows us to see how ensembles should be formed so that combination is useful. For example, we see that bagging trees is a good idea but bagging support vector machines is not good; with the latter, it is better to play with the kernel or inputs. With *knn*, neither resampling nor varying *k* suffices to get uncorrelated versions, one should vary some other factor, for example, input features, or one should combine *knn* with some other algorithm, a linear perceptron for example. We also see that it is better to combine different algorithms [42] or different inputs [1,11], rather than different training subsets or hyperparameters. In case we have many classifiers from different families, we should prefer to use classifiers from different families because the correlation between families is less than the correlation between variants of the same family. When learners are positively correlated, they cannot be used as they are because that will increase error. We propose to construct new uncorrelated eigenclassifiers from a set of correlated classifiers and our experimental results show that our PCA-based method is as accurate as using the whole ensemble or the optimum subset. We define eigenclassifiers, that is, linear combinations of existing base classifiers which are uncorrelated.

Using a feature extraction method such as PCA instead of a feature selection method such as subset selection may be costlier, but a subset keeps some and discards the rest and has the potential to remove information that may be useful on some instances; keeping and combining all allows redundancy and promises to be fault tolerant. For example, pattern recognition is getting increasingly popular in adversarial environments (i.e. intrusion detection, spam filtering, etc.) where there is a party who intentionally tries to fool the system [6]. We believe that our method will be useful for such applications because it combines all sources of information while also taking their correlation into account.

Compared with the most commonly used two ensemble algorithms, i.e. AdaBoost and Bagging, we have seen that the PCA-based method we propose is either more accurate, or has comparable accuracy but uses fewer number of classifiers. We have also tried using linear discriminant analysis (LDA) instead of PCA; LDA is sometimes advantageous in that it uses class information but in our case, it does not work as well as PCA because (i) the number of components should be less than K, the number of classes, which makes the approach based on LDA unsuccessful in cases where K is small [41] or (ii) LDA assumes that instances of a class form a single cluster which may not always be the case. One can also use nonlinear dimensionality reduction methods, though in such a case we might lose interpretability of the new metaclassifiers. It is also possible to combine multiple representations of the same input by training experts on different representations of the same object [1]. The analysis we do in this work can also be carried out for this case, that is, to check how much correlation there is and how experts using different representations can be fused using PCA [41].

Acknowledgment

We would like to thank Mehmet Gönen for discussions. This work has been supported by the Turkish Academy of Sciences in the framework of the Young Scientist Award Program (EA-TÜBA-GEBİP/2001-1-1), Boğaziçi University Scientific Research Project 05HA101 and Turkish Scientific Technical Research Council TÜBİTAK EEEAG 104E079.

References

[1] F. Alimoğlu, E. Alpaydın, Combining multiple representations and classifiers for pen-based handwritten digit recognition, in: Proceedings of the International Conference on Document Analysis and Recognition, ICDAR'97, 1997.
[2] E. Alpaydın, Voting over multiple condensed nearest neighbors, Artificial Intelligence Review 11 (1–5) (1997) 115–132.
[3] E. Alpaydın, Combined 5 $\times$ 2 cv F test for comparing supervised classification learning algorithms, Neural Computation 11 (8) (1999) 1885–1892.
[4] A. Asuncion, D.J. Newman, UCI machine learning repository, <http://www.ics.uci.edu/~mlearn/MLRepository.html>, 2007.
[5] S.D. Bay, Combining nearest neighbor classifiers through multiple feature subsets, in: Proceedings of the International Conference on Machine Learning, ICML'98, 1998.
[6] B. Biggio, G. Fumera, F. Roli, Multiple classifier systems for robust classifier design in adversarial environments, International Journal of Machine Learning and Cybernetics 1 (2010) 27–41.
[7] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140.
[8] G. Brown, J. Wyatt, R. Harris, X. Yao, Diversity creation methods: a survey and categorisation, Information Fusion 6 (1) (2005) 5–20.
[9] R. Caruana, A. Niculescu-Mizil, G. Crew, A. Ksikes, Ensemble selection from libraries of models, in: Proceedings of the International Conference on Machine Learning, ICML'04, 2004.
[10] C.C. Chang C.J. Lin LIBSVM: a library for support vector machines, <http://www.csie.ntu.edu.tw/~cjlin/libsvm>, 2001.
[11] C. Demir, E. Alpaydın, Cost-conscious classifier ensembles, Pattern Recognition Letters 26 (14) (2005) 2206–2214.
[12] J. Demsar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.
[13] Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, in: Proceedings of the International Conference on Machine Learning, ICML'96, 1996.
[14] G. Fumera, F. Roli, A theoretical and experimental analysis of linear combiners for multiple classifier systems, IEEE Transactions on Pattern Analysis Machine Intelligence 27 (6) (2005) 942–956.
[15] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Information Sciences 180 (10) (2010) 2044–2064.
[16] T.K. Ho, The random subspace method for constructing decision forests, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (8) (1998) 832–844.
[17] R.A. Jacobs, Bias/variance analysis of mixtures-of-experts architectures, Neural Computation 9 (2) (1997) 369–383.
[18] A. Jain, K. Nandakumar, A. Ross, Score normalization in multimodal biometric systems, Pattern Recognition 38 (2005) 2270–2285.
[19] I.T. Jolliffe, Discarding variables in a principal component analysis. II: Real data, Applied Statistics 22 (1) (1973) 21–31.
[20] L.I. Kuncheva, Combining classifiers: soft computing solutions, in: S.K. Pal (Ed.), Pattern Recognition: From Classical to Modern Approaches, World Scientific, 2001.
[21] L.I. Kuncheva, Combining pattern classifiers: methods and algorithms, Wiley-Interscience, 2004.
[22] L.I. Kuncheva, Special issue on diversity in multiple classifier systems, Information Fusion 6 (1) (2005) 1–115.
[23] L.I. Kuncheva, C.J. Whitaker, Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy, Machine Learning 51 (2) (2003) 181–207.
[24] L.I. Kuncheva, C.J. Whitaker, C.A. Ship, R.P. Duin, Is independence good for combining classifiers? in: Proceedings of the 15th International Conference on Pattern Recognition, ICPR'00, 2000.
[25] C.-L. Liu, Classifier combination based on confidence transformation, Pattern Recognition 38 (2005) 11–28.
[26] R. Mallipeddi, S. Mallipeddi, P. Suganthan, Ensemble strategies with adaptive evolutionary programming, Information Sciences 180 (9) (2010) 1571–1581.
[27] D.D. Margineantu, T.G. Dietterich, Pruning adaptive boosting, in: Proceedings of the International Conference on Machine Learning, ICML'97, 1997.
[28] C.J. Merz, Using correspondence analysis to combine classifiers, Machine Learning 36 (1–2) (1999) 33–58.
[29] C.J. Merz, M.J. Pazzani, A principal components approach to combining regression estimates, Machine Learning 36 (1–2) (1999) 9–32.
[30] D. Partridge, W.B. Yates, Engineering multiversion neural-net systems, Neural Computation 8 (4) (1996) 869–893.

[31] C.E. Rasmussen, R.M. Neal, G. Hinton, D. van Camp, M. Revow, Z. Ghahramani, R. Kustra, R. Tibshirani, Delve data for evaluating learning in valid experiments, <http://www.cs.toronto.edu/∼delve/>, 1995–1996.

[32] S. Raudys, Trainable fusion rules. I: large sample size case, Neural Networks 19 (2006) 1506–1516.

[33] A.C. Rencher, Interpretation of canonical discriminant functions, canonical variates, and principal components, The American Statistician 46 (3) (1992) 217–225.

[34] F. Roli, G. Giacinto, G. Vernazza, Methods for designing multiple classifier systems, in: Proceedings of the International Workshop on Multiple Classifier Systems, MCS'01, 2001.

[35] D. Ruta, B. Gabrys, Classifier selection for majority voting, Information Fusion 6 (1) (2005) 63–81.

[36] A.J.C. Sharkey, N.E. Sharkey, U. Gerecke, G.O. Chandroth, The "test and select" approach to ensemble combination, in: Proceedings of the International Workshop on Multiple Classifier Systems, MCS'00, vol. 1857, 2000.

[37] C. Tamon, J. Xiang, On the boosting pruning problem, in: Proceedings of the European Conference on Machine Learning, ECML'00, 2000.

[38] K.M. Ting, I.H. Witten, Issues in stacked generalization, Journal of Artificial Intelligence Research 10 (1999) 271–289.

[39] D.L. Tong, R. Mintram, Genetic algorithm-neural network (gann): a study of neural network activation functions and depth of genetic algorithm search applied to feature selection, International Journal of Machine Learning and Cybernetics 1 (2010) 75–87.

[40] K. Tumer, J. Ghosh, Error correlation and error reduction in ensemble classifiers, Connection Science 8 (3) (1996) 385–404.

[41] A. Ulaş, Incremental construction of cost-conscious ensembles using multiple learners and representations in machine learning, Ph.D. thesis, Boğaziçi University. <http://www.cmpe.boun.edu.tr/∼ulas/phdthesis.pdf>, 2009.

[42] A. Ulaş, M. Semerci, O.T. Yıldız, E. Alpaydın, Incremental construction of classifier and discriminant ensembles, Information Sciences 179 (9) (2009) 1298–1318.

[43] L.J. Wang, An improved multiple fuzzy nnc system based on mutual information and fuzzy integral, International Journal of Machine Learning and Cybernetics 2 (2011) 25–36.

[44] X.-Z. Wang, C.-R. Dong, Improving generalization of fuzzy if-then rules by maximizing fuzzy entropy, IEEE Transactions on Fuzzy Systems 17 (2009) 556–567.

[45] X.-Z. Wang, J.-H. Zhai, S.-X. Lu, Induction of multiple fuzzy decision trees based on rough set technique, Information Sciences 178 (2008) 3188–3202.

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

[47] R. Xia, C. Zong, S. Li, Ensemble of feature sets and classification algorithms for sentiment classification, Information Sciences 181 (6) (2011) 1138–1152.

[48] J. Xiao, C. He, X. Jiang, D. Liu, A dynamic classifier ensemble selection approach for noise data, Information Sciences 180 (18) (2010) 3402–3421.

[49] Y. Yang, G.I. Webb, J. Cerquides, K.B. Korb, J. Boughton, K.M. Ting, To select or to weigh: a comparative study of linear combination schemes for superparent-one-dependence estimators, IEEE Transactions on Knowledge and Data Engineering 19 (12) (2007) 1652–1665.

[50] O.T. Yıldız, E. Alpaydın, Linear discriminant trees, in: Proceedings of the International Conference on Machine Learning, ICML'00, 2000.

[51] O.T. Yıldız, A. Ulaş, M. Semerci, E. Alpaydın, AYSU: machine learning data sets for model combination, <http://www.cmpe.boun.edu.tr/∼ulas/aysu>, 2007.

[52] E. Yu, P. Suganthan, Ensemble of niching algorithms, Information Sciences 180 (15) (2010) 2815–2833.

[53] Y. Zhang, S. Burer, W.N. Street, Ensemble pruning via semi-definite programming, Journal of Machine Learning Research 7 (2006) 1315–1338.

[54] Z.-H. Zhou, J. Wu, W. Tang, Ensembling neural networks: many could be better than all, Artificial Intelligence 137 (2002) 239–263.