Classifier Combination in Speech Recognition

László Felföldi, András Kocsor, and László Tóth

Research Group on Artificial Intelligence
of the Hungarian Academy of Sciences and University of Szeged
H-6720 Szeged, Aradi vértanúk tere 1., Hungary
{lfelfold, kocsor, tothl}@inf.u-szeged.hu
http://www.inf.u-szeged.hu/speech

Abstract. In statistical pattern recognition the principal task is to classify abstract data sets. Instead of using robust but computational expensive algorithms it is possible to combine 'weak' classifiers that can be employed in solving complex classification tasks. In this comparative study we will examine the effectiveness of the commonly used hybrid schemes - especially those used for speech recognition problems - concentrating on cases which employ different combinations of classifiers.

1 Introduction

The goal of designing pattern recognition systems is to achieve the best possible classification performance for the specified task. This objective traditionally led to the development of different classification schemes for recognition problems the user would like solved. Experiments shows that although one of the designs should yield the best performance, the sets of patterns misclassified by the different classifiers do not necessarily overlap. These observations motivated the relatively recent interest in combining classifiers. The main idea behind it is not to rely on the decision of a single classifier. Rather, all of the inducers or their subsets are used for decision-making by combining their individual opinions to produce a final decision.

A fair number of combination schemes have been proposed in the literature [3][5][6], these schemes differing from each other in their architecture, the characteristics of the combiner, and the selection of the individual classifiers. From the analytic viewpoint, there are basically two combination scenarios. In the first scenario, all the classifiers use the same representation of input patterns, while in the second scenario each classifier uses its own pattern representation, so the measurements extracted from the pattern are unique to each classifier.

In this paper we focus on classifier combinations of the first kind, especially pattern representations commonly used in speech recognition tasks. The paper is organized as follows. In the first section we give a general overview of combination schemes, concentrating on their input representation, architecture, and some of techniques for generating the independent inputs needed for combinations. Then we derive the commonly used combination rules like the Product and Sum rule. In the next section we compare the individual classifiers and the associated hybrid schemes, including the traditional Bagging and Boosting techniques. In the final section we summarize the main results and conclusions of the paper.

2 Concept of Combination Schemes

2.1 Types of Knowledge Sources

Each inducer has to be capable of assigning one of the classes to a given pattern $\mathbf{x_i}$. The output information from each is sufficient for some types of combiners (e.g., Majority Voting Rule), but other combiners might need other types of information from individual classifiers. The types of information required can be grouped into three main categories:

Abstract: Only the assigned class label ω_k is required. Classifiers which only need this information as input are voting combiners like Bagging and Boosting.

Ranking: Instead of providing just the best class ω_k associated with the given pattern $\mathbf{x_i}$, the list of classes is supplied, ranked in order of probability. This more general information type can be used as input for combiners like the Borda count rule.

Measurement or Confidence: In the most general case, each of the a posteriori probabilities $p(\omega_j|\mathbf{x_i})$ are provided. Combiners can aggregate these probabilities from different inducers and make a final decision. Examples include combiners which use the measurement information type are Prod, Sum, and Max Rules.

2.2 Combination Architectures

Various schemes for combining multiple classifiers can be grouped into three main categories according to their architecture:

Parallel: Each of the inducers are invoked independently, and their results are then combined by a combiner. The majority of combination architectures in the literature belong to this category.

Cascading: Individual classifiers are invoked in a linear sequence. The number of possible classes for a given pattern is gradually reduced as more classifiers in the sequence are invoked. For the sake of efficiency, inaccurate but cheap classifiers are considered first, followed by more accurate and expensive inducers.

Hierarchical: Individual classifiers are combined into a structure similar to that of a decision tree classifier. The tree nodes, however, may now be associated with complex classifiers requiring a large number of features. The advantage of this architecture is its high efficiency and flexibility in exploiting the discriminant power of different types of features.

2.3 The Training Set

A classifier combination is especially useful if the classifiers applied are largely independent. If this is not already guaranteed by the use of different learning sets or different learning methods, various re-sampling techniques like rotation and bootstrapping may be used to artificially create such differences.

Rotation: The original learning set is divided into n disjoint subsets and uses different unions of $n-1$ subsets as training sets. This technique is commonly used in cross validation during error estimation.

Bootstrapping: A bootstrap sample can be generated by sampling instances from the training set with replacement, using a specified probability distribution. Examples include Bagging and Boosting.

Stacking: The outputs of the individual classifiers are used to train the *stacked* classifier[11]. The final decision is made based on the outputs of the stacked classifier in conjunction with the outputs of individual classifiers.

Generating noise: The generated classification model of an unstable classifier strongly depends on existing errors in the training database. Adding artificial errors is a way of generating a set of more or less independent classifiers, making it possible to fulfil the requirements of a combiner.

2.4 "Winner-Takes-All"

Providing there is a set of independent classifiers, a common way of combining them is to select one with the best behavior on a given test database. During the classification only the output of the selected classifier is computed, and only this will affect the resulting decision. This selection is an "*early*" combination scheme widely used in pattern recognition.

3 Integration of Knowledge Sources

Consider a pattern recognition problem where the pattern Z is to be assigned to one of m possible classes $(\omega_1, \ldots, \omega_m)$. Let us assume that we have R classifiers, each representing the given pattern by a different feature vector. Next, denote this feature vector (employed by the ith classifier) by $\mathbf{x_i}$. In the feature space each class ω_k is modelled by the probability density function $p(\mathbf{x_i}|\omega_\mathbf{k})$ and its a priori probability of occurrence $P(\omega_k)$.

According to Bayesian theory, for given features $\mathbf{x_i}$, $i \in \{1, \ldots, R\}$ the pattern Z should be assigned to class ω_j with the maximal value of the a posteriori probability such that

$$f(\mathbf{x_i}) = \omega_j, \quad j = \operatorname*{argmax}_k P(\omega_k|\mathbf{x_1}, \ldots, \mathbf{x_R}). \tag{1}$$

Let us rewrite the a posteriori probability using Bayes' Theorem. We have

$$P(\omega_k|\mathbf{x_1}, \ldots, \mathbf{x_R}) = \frac{p(\mathbf{x_1}, \ldots, \mathbf{x_R}|\omega_k)P(\omega_k)}{p(\mathbf{x_1}, \ldots, \mathbf{x_R})}. \tag{2}$$

In the latter the unconditional joint probability density can be expressed in terms of the conditional feature distributions, so that

$$p(\mathbf{x_1},\ldots,\mathbf{x_R}) = \sum_{j=0}^{m} p(\mathbf{x_1},\ldots,\mathbf{x_R}|\omega_j)P(\omega_j). \tag{3}$$

3.1 Product Rule

Let us assume that the probability distributions $p(\mathbf{x_i},\ldots,\mathbf{x_R}|\omega_k)$ are conditionally statistically independent. Then

$$p(\mathbf{x_1},\ldots,\mathbf{x_R}|\omega_k) = \prod_{i=0}^{R} p(\mathbf{x_i}|\omega_k), \tag{4}$$

and the decision rule

$$f(\mathbf{x_i}) = \omega_j, \quad j = \operatorname*{argmax}_k P(\omega_k) \prod_i p(\mathbf{x_i}|\omega_k), \tag{5}$$

or in terms of the a posteriori probabilities generated by the respective classifiers

$$\operatorname*{argmax}_k P^{1-R}(\omega_k) \prod_i p(\omega_k|\mathbf{x_i}). \tag{6}$$

3.2 Sum Rule

In some applications it may be appropriate to assume that the a posteriori probabilities computed by the respective classifiers will not dramatically deviate from those of the prior probabilities. This is a rather strong assumption but it may be readily satisfied when the available information is highly ambiguous due to high level of noise. In such a situation we may assume that the a posteriori probability can be expressed in the form

$$P(\omega_k|\mathbf{x_i}) = P(\omega_k)(1+\delta_{ki}), \tag{7}$$

where $\delta_{ki} \ll 1$. Substituting this for the a posteriori probabilities in (6), we find that

$$P^{1-R}(\omega_k) \prod_i P(\omega_k|\mathbf{x_i}) = P(\omega_k) \prod_i (1+\delta_{ki}). \tag{8}$$

If we neglect terms of second and higher order, we can approximate the right-hand side and obtain the sum decision rule

$$f(\mathbf{x_i}) = \omega_j, \quad j = \operatorname*{argmax}_k \left[(1+R)P(\omega_k) + \sum_i p(\omega_k|\mathbf{x_i}) \right]. \tag{9}$$

3.3 Max, Min Rule

The decision rules (6) and (9) constitute the basic schemes for combining classifiers. Many commonly used combination strategies can be developed from these rules after noting that

$$\prod_i P(\omega_k|\mathbf{x_i}) \leq \min_i P(\omega_k|\mathbf{x_i}) \leq \sum_i P(\omega_k|\mathbf{x_i}) \leq \max_i P(\omega_k|\mathbf{x_i}). \tag{10}$$

This inequality suggests that the product and sum combination rules may be approximated by the max and min operators, where appropriate. These approximations lead to the following:

Max Rule:

$$f(\mathbf{x_i}) = \omega_j, \quad j = \operatorname*{argmax}_k \left[(1+R)P(\omega_k) + R \max_i p(\omega_k|\mathbf{x_i})\right], \tag{11}$$

Min Rule:

$$f(\mathbf{x_i}) = \omega_j, \quad j = \operatorname*{argmax}_k \left[P^{1-R}(\omega_k) + R \max_i p(\omega_k|\mathbf{x_i})\right]. \tag{12}$$

3.4 Median Rule

Note that using the *equal prior* assumption, the sum rule can be interpreted as computing the average a posteriori probability. It is well known that a robust estimate of the mean is the median, so it might be more appropriate to use it as the basis for the combining procedure. Adopting this leads to the following rule:

$$f(\mathbf{x_i}) = \omega_j, \quad j = \operatorname*{argmax}_k \operatorname*{med}_i p(\omega_k|\mathbf{x_i}). \tag{13}$$

3.5 Voting Rule

Hardening a posteriori probabilities $P(\omega_k|\mathbf{x_i})$ will produce binary valued functions Δ_{ki} like

$$\Delta_{ki} = \begin{cases} 1 & if\ P(\omega_k|\mathbf{x_i}) = \max_j P(\omega_j|\mathbf{x_i}) \\ 0 & otherwise, \end{cases} \tag{14}$$

which results in combination decision outcomes rather than a combination of a posterori probabilities. Assuming that each a priori probabilities is equal, this leads to the following decision rule:

$$f(\mathbf{x_i}) = \omega_j, \quad j = \operatorname*{argmax}_k \sum_i \Delta_{ki}. \tag{15}$$

Note that for each class ω_k, the sum on the right hand side of (15) simply counts the votes received for this hypothesis from each individual classifier.

3.6 Borda count

Instead of hardening a posteriori probabilities it is possible to use modified probabilities ρ_{ki} based on ranking information.

$$\rho_{ki} = \frac{1}{C} \sum_{j:P(\omega_j|\mathbf{x_i}) \leq P(\omega_k|\mathbf{x_i})} 1 \tag{16}$$

where C is a normalization constant. This results in the following decision rule:

$$f(\mathbf{x_i}) = \omega_j, \quad j = \operatorname*{argmax}_k \sum_i \rho_{ki}. \tag{17}$$

3.7 Bagging

The Bagging (Bootstrap aggregating) algorithm[1] votes classifiers generated by different bootstrap samples (replicates). A bootstrap sample is generated by uniformly sampling m instances from the training set with replacement. T bootstrap samples $B_1, B_2, ..., B_T$ are generated and a classifier C_i is built from each bootstrap sample B_i. A final classifier C^* is built from $C_1, C_2, ..., C_T$ whose output is the class predicted most often by its sub-classifiers (majority voting).

```
Bagging algorithm
**Require:** Training Set $S$, Inducer $\mathcal{I}$
**Ensure:** Combined classifier $C^*$
  **for** $i = 1 \dots T$ **do**
    $S'$ = bootstrap sample from $S$
    $C_i = \mathcal{I}(\mathcal{S}')$
  **end for**
  $C^*(\mathbf{x}) = \operatorname*{argmax}_j \sum_{i:C_i(\mathbf{x})=\omega_j} 1$
```

For a given bootstrap sample, an instance in the training set will have a probability $1 - (1 - 1/m)^m$ of being selected at least once from the m instances randomly selected from the training set. For large m, this is about 1-1/e = 63.2%. This perturbation causes different classifiers to be built if the inducer is unstable (e.g. ANNs, decision trees) and the performance may improve if the induced classifiers are uncorrelated. However, Bagging can slightly degrade the performance of stable algorithms (e.g. kNN) since effectively smaller training sets are used for training.

3.8 Boosting

Boosting[9] was introduced by Shapire (1990) as a method for boosting the performance of a weak learning algorithm. Here we will focus on AdaBoost, sometimes called "AdaBoost.M1". Like Bagging, the AdaBoost algorithm generates a set of classifiers and it makes a decision based on their votes. However, beyond this, the two algorithms substantially differ. The AdaBoost algorithm generates the classifiers sequentially, while Bagging can generate them in parallel. AdaBoost also changes the weights of the training instances provided as input for each inducer based on classifiers that were previously built. The final decision is made using a weighted voting scheme for each classifier, whose weights depend on the performance of the training set used to build it.

```
Boosting algorithm
Require: Training Set $S$ of size $m$, Inducer $\mathcal{I}$
Ensure: Combined classifier $C^*$
  $S' = S$ with weights assigned to be $1/m$
  for $i = 1 \ldots T$ do
    $S'$ = bootstrap sample from $S$
    $C_i = \mathcal{I}(\mathcal{S}')$
    $\epsilon_i = \sum_{\mathbf{x_j} \in S' : C_i(\mathbf{x_j}) \neq \omega_j}$ weight of $\mathbf{x_j}$
    if $\epsilon_i > 1/2$ then Exit
    $\beta_i = \frac{\epsilon_i}{(1-\epsilon_i)}$
    for all $\mathbf{x_j} \in S'$ such $C_i(\mathbf{x_j}) = \omega_j$ do
      weight of $\mathbf{x_j}$ = weight of $\mathbf{x_j} \cdot \beta_i$
    end for
    normalize weights of instances to sum 1
  end for
  $C^*(\mathbf{x}) = \operatorname{argmax}_j \sum_{i:C_i(\mathbf{x})=\omega_j} \log \frac{1}{\beta_i}$
```

The AdaBoost algorithm requires a weak learning algorithm whose error is bounded by a constant strictly less than 1/2. In the case of multi-class classification this condition could be different to guarantee. Some implementations of AdaBoost makes use of boosting by re-sampling because the inducers employed are unable to support weighted instances. Using appropriate classifiers one can try re-weighting, which might work better in practice.

4 Experimental results

4.1 Data-Sets

The various hybrid techniques were compared using a relatively small corpus that consists of several speakers pronouncing Hungarian numbers. More precisely, 20 speakers were used for training and 6 for testing, and 52 utterances were recorded from each person. The ratio of male and female speakers was 50%:50% in both the training and the testing sets. The recordings were made using an inexpensive commercial microphone in a reasonably quiet environment, at a sample rate of 22050Hz. The whole corpus was manually segmented and labelled. Because the corpus contained only numbers, we had occurrences of only 32 phones, which is approximately two-thirds of the Hungarian phoneme set. Because some of these labels represented only allophonic variations of the same phoneme, some labels were fused, and so we actually worked with a set of just 28 labels. We performed tests as well with two other groupings where the labels were grouped into 11 and 5 classes, respectively, based on phonetic similarity. We had two good reasons for doing experiments with these gross phonetic classes. First, we could increase the number of training examples in each class and inspect the effects of this on the learning algorithms. Second, our speech recognition system has a first-pass stage in which the segments are classified into gross phonetic categories only.

Hence we had three phonetic groupings, which will be denoted by *grp1*, *grp2*, and *grp3* from this point on. With the first grouping, the number of occurrences of the different labels in the training set was between 40 and 599. This value was between 120 and 1158 for the second grouping and between 716 and 2158 for the third grouping.

4.2 Features

The trials were performed on five feature sets[7] described later. Because all sets contained duration information, we do not mention it separately. *Set1* consisted of the MFCC coefficients, because these are the most commonly used features. To have the chance of studying the usefulness of the cosine transform, we also carried out tests on the filter bank energies themselves (*Set3*). By augmenting *Set1* and *Set3* with gravity center features, we acquired two new sets, *Set2* and *Set4*. We hoped that the addition of these phonetics-based features would lead to a slight improvement. Lastly, the largest set *Set5* contains all the features, that is, filter bank energies, MFCC coefficients, and gravity centers. The same trials were performed on the three phoneme groupings *grp1, grp2, grp3*.

4.3 Classifiers

Each of the classifiers used in the experiments was modified so as to make them capable of providing a posteriori probabilities for each class ω_k.

Decision Tree Learner: Our version of DTL used in the experiments is based on the C4.5 tree learning algorithm[8]. It is able to learn predefined discrete classes from labelled examples. The result of the learning process is an axis-parallel decision tree. This means that during the training, the sample space is divided into subspaces by hyper-planes which are parallel to every axis but one. In this way, we obtain many n-dimensional rectangular regions that are labelled by class labels and organized in a hierarchical way so it can be encoded into a tree. For knowledge representation, DTL uses the "divide and conquer" technique, which means that regions are split during learning whenever they are insufficiently homogenous, and left untouched when they are homogenous. Splitting is done by axis parallel hyper-planes and, thanks to this, the learning process is quite fast. Hence the greatest advantage of the method is time complexity. Unfortunately, the simple learning strategy in certain cases results in a huge number of regions that are needlessly split.

Gaussian Mixture Models: GMM[4] assumes that the class-conditional probability distribution $p(\mathbf{x_i}|\omega_k)$ can be well-approximated by a distribution of the form

$$f(\mathbf{x_i}) = \sum_{j=1}^{l} c_j \mathcal{N}(\mathbf{x_i}, \mu_\mathbf{j}, \mathbf{C_j}) \tag{18}$$

where $\mathcal{N}(\mathbf{x_i}, \mu_\mathbf{j}, \mathbf{C_j})$ denotes the multidimensional normal distribution with mean $\mu_\mathbf{j}$ and covariance matrix $\mathbf{C_j}$, l is the number of mixtures and c_j are nonnegative weighting factors that sum to 1.

As luck would have it, there is no closed formula for the optional parameters of the mixture model. Normally the expectation-maximization (EM) algorithm is utilized to find proper parameters, but it guarantees only a locally optimal solution. This iterative technique is very sensitive to initial parameter values, so we used k-mean clustering to find a good starting parameter set. Since k-mean clustering again guarantees only a local optimum, we ran it 15 times with random parameters and took the one with the highest log-likelihood to initialize the EM algorithm. In each case the covariance matrix was made diagonal because training full matrices would have required much more training data and computation time.

Support Vector Machines: SVM[10] was developed by Vapnik for binary classification. It selects a hyperplane with maximal margin to separate points with different class labels, but prior to that it applies a nonlinear transformation to map the patterns to a higher dimensional space where the classification is easier. The problem of nonlinearity is handled by kernel functions which makes Support Vector Machines a very powerful tool for machine learning.

Artificial Neural Networks: ANNs[2] now count among the conventional pattern recognition tools, so we will not describe them here. In the trials performed we used the most common feed-forward multi-layer perceptron network with the back-propagation learning rule. The number of neurons in the hidden layer was set at three times the number of features (this value was chosen empirically based on preliminary trials). Training was stopped when, for the last 20 iterations, the decrease in the error between two consecutive iteration step remained below a certain threshold.

k Nearest Neighbor: kNN[4] is a well known classifier used in pattern recognition. Because no rule or decision is made before the actual classification, this approach is called *lazy learning*. Typically, this kind of machine learning has a very short training time but the classification of new data takes rather a long time. The storing and processing of millions of examples can also be a serious handicap. Despite this, being a *stable* inducer it is a great tool for machine learning.

4.4 Classification without combination

In the first stage all the classifiers were tested on the same data set. As can be seen in Table 1, SVM performed the best, and ANN also had good score, but the other classifiers only produced poor results.

Table 1. Classification errors of the individual classifiers

	ANN	**SVM**	**GMM**	**kNN**	**DTL**
g1set1	**13.71**%	14.01%	21.16%	20.09%	35.58%
g1set2	**12.23**%	13.71%	24.65%	19.86%	33.75%
g1set3	11.88%	**11.76**%	26.77%	21.34%	32.80%
g1set4	12.23%	**11.82**%	25.59%	21.99%	32.09%
g1set5	11.88%	**11.41**%	24.36%	19.24%	34.22%
g2set1	10.64%	**9.93**%	18.68%	13.83%	22.10%
g2set2	10.40%	**9.69**%	21.69%	13.48%	22.46%
g2set3	8.63%	**8.22**%	18.68%	12.12%	21.75%
g2set4	10.76%	**7.92**%	18.26%	13.00%	21.45%
g2set5	9.93%	**8.16**%	20.33%	12.41%	24.76%
g3set1	7.15%	**6.03**%	11.76%	8.51%	13.38%
g3set2	6.32%	**5.56**%	11.82%	7.51%	12.71%
g3set3	**5.38**%	5.61%	10.22%	6.86%	10.87%
g3set4	5.32%	**5.14**%	10.46%	7.39%	12.00%
g3set5	5.61%	**4.96**%	10.11%	6.86%	12.35%

4.5 Selecting the Classifier Set

One can expect a different performance depending on which classifiers are combined. To obtain the optimal classifier selection, a different subset of the classifiers was selected for combining with the same method. The subsets were generated sequentially by inserting the next element from the strictly ordered list of classifiers into the previous subset. The combination rule applied in the test was the Product rule.
The above test results in Table 2 show that there is little point in using all the classifiers in combination schemes, as the optimal solution is the combination of SVM, ANN, and kNN. Including GMM and DTL only leads to a deterioration in the classification performance.

Table 2. Combination error obtained using the Product Decision Rule

	Ø	**ANN SVM**	**ANN SVM kNN**	**ANN SVM kNN GMM**	**ANN SVM kNN GMM DTL**
g1set1	13.71%	12.46%	**12.00**%	15.07%	29.91%
g1set2	12.23%	**11.70**%	11.76%	18.14%	27.42%
g1set3	11.76%	**11.05**%	11.70%	18.44%	27.96%
g1set4	11.82%	**11.05**%	12.77%	19.62%	28.25%
g1set5	11.41%	10.99%	**10.87**%	19.39%	27.66%
g2set1	9.93%	10.11%	**8.63**%	10.93%	17.43%
g2set2	9.69%	9.69%	**8.76**%	12.12%	16.61%
g2set3	8.22%	7.80%	**7.03**%	12.29%	16.19%
g2set4	**7.92**%	9.10%	8.98%	13.18%	17.14%
g2set5	**8.16**%	9.46%	8.51%	13.95%	19.33%
g3set1	6.03%	6.21%	**5.14**%	7.98%	8.04%
g3set2	5.56%	5.67%	**5.02**%	7.74%	9.04%
g3set3	5.38%	**4.96**%	5.14%	7.92%	7.21%
g3set4	5.14%	5.02%	**4.67**%	8.22%	9.40%
g3set5	**4.96**%	5.50%	5.02%	7.74%	9.10%

4.6 Comparing combination rules

In the next stage of the testing we combined the outputs of the selected classifiers (SVM, ANN, and kNN) by applying various combination rules. Table 3 suggests that there is no definite optimal rule for combining classifiers using this database. Combiners which applied the Sum rule performed the best, but the improvement compared with the others was only marginal.

4.7 Results using Bagging

In this part the Bagging algorithm was applied to each of the classifiers. During the trials 15 bootstrap samples were generated, each of them with a size two-thirds that of the size of the original training-set.
As can be seen (Table 4), Bagging can improve classification performance, almost to the same level of the previous combination methods, but it requires more processing time.

4.8 Results using Boosting

Because Boosting is an improvement on Bagging, we expected a better performance. Testing Boosting on this data-set, however, produced roughly the same classification error values. The explanation for this is that the classifiers are too "strong", they generated very small classification error when using the training

Table 3. Classification error of hybrid combinations using ANN, SVM, and kNN

	Prod	**Sum**	**Max**	**Min**	**Borda**	**Voting**
g1set1	12.00%	**11.64**%	12.59%	12.77%	12.77%	13.23%
g1set2	11.76%	**11.41**%	12.06%	13.06%	12.06%	11.47%
g1set3	11.70%	11.35%	13.18%	12.29%	11.64%	**10.87**%
g1set4	12.77%	12.41%	14.24%	12.35%	12.59%	**11.41**%
g1set5	10.87%	**10.70**%	11.88%	11.17%	11.41%	11.17%
g2set1	8.63%	**8.45**%	9.22%	9.10%	8.87%	9.16%
g2set2	8.75%	**8.57**%	9.87%	9.16%	9.10%	9.04%
g2set3	**7.03**%	7.09%	8.04%	7.33%	7.80%	7.15%
g2set4	8.98%	7.98%	9.87%	9.34%	8.69%	**7.74**%
g2set5	8.51%	8.22%	8.98%	**7.98**%	8.81%	8.51%
g3set1	**5.14**%	**5.14**%	6.32%	5.61%	5.61%	5.56%
g3set2	5.02%	5.50%	5.73%	**4.85**%	5.08%	5.08%
g3set3	5.14%	**4.91**%	5.26%	5.20%	5.08%	**4.91**%
g3set4	4.67%	**4.61**%	5.02%	4.79%	5.02%	4.91%
g3set5	5.02%	**4.96**%	5.08%	5.14%	5.14%	5.14%

Table 4. Classification error of Bagging classifiers

	ANN	**SVM**	**GMM**	**kNN**	**DTL**
g1set1	12.71%	12.59%	19.80%	20.45%	26.36%
g1set2	11.76%	12.23%	23.05%	19.21%	22.70%
g1set3	10.99%	10.28%	22.70%	21.10%	22.87%
g1set4	11.88%	11.29%	22.94%	22.28%	21.57%
g1set5	10.70%	10.82%	21.22%	20.21%	21.39%
g2set1	11.17%	9.52%	14.83%	13.95%	16.84%
g2set2	9.75%	9.40%	18.38%	13.65%	17.32%
g2set3	8.16%	7.45%	15.96%	12.83%	16.90%
g2set4	8.69%	7.51%	16.67%	13.12%	16.31%
g2set5	9.57%	7.86%	16.67%	12.65%	16.37%
g3set1	6.80%	5.44%	11.05%	8.87%	11.23%
g3set2	6.38%	5.08%	10.64%	7.80%	10.52%
g3set3	5.79%	5.50%	10.11%	7.51%	11.82%
g3set4	5.26%	4.61%	09.57%	7.15%	10.22%
g3set5	6.09%	5.02%	9.63%	7.09%	10.40%

set. After the first step of Boosting, only the "noise" remained in the bootstrap sample, which was too difficult to separate, and the classification error on this sample generally hit the 50% limit. Here the algorithm exits, but in practice a standard Bagging (uniform bootstrapping) step can be performed instead. The result (Table 5)is very close to that for the Bagging algorithm.

Table 5. Classification error of Boosting classifiers

	ANN	**SVM**	**GMM**	**kNN**	**DTL**
g1set1	12.51%	12.44	18.55%	20.32%	25.12%
g1set2	11.56%	11.97	22.05%	19.42%	22.25%
g1set3	10.79%	9.67	20.12%	20.87%	21.98%
g1set4	11.88%	10.43	20.05%	22.16%	21.60%
g1set5	10.66%	10.34	21.22%	19.87%	20.47%
g2set1	11.17%	9.72	14.87%	13.95%	16.75%
g2set2	9.75%	9.31	17.12%	13.87%	15.88%
g2set3	8.42%	7.14	15.27%	12.83%	16.36%
g2set4	8.64%	7.62	14.98%	13.07%	16.59%
g2set5	9.77%	7.36	15.14%	12.68%	15.72%
g3set1	6.76%	5.32	11.23%	8.97%	11.17%
g3set2	6.41%	4.87	9.96%	7.72%	10.69%
g3set3	5.63%	5.50	10.03%	7.31%	11.47%
g3set4	5.28%	4.82	9.98%	7.22%	9.93%
g3set5	6.02%	4.91	9.61%	6.98%	10.05%

q

5 Conclusion

We reviewed the various combination schemes available using speech recognition oriented data-sets. Making classifier hybrids improved the discrimination performance, the best results being obtained by aggregating the output of SVM, ANN, and kNN. Experimental results show that the performance of the combiners applying different decision rule was not significantly different, but the sum rule outperformed the others. Comparing the traditional Bagging and Boosting techniques, we found that they have nearly the same classification improvement, but their applicability is limited because they are too cpu intensive. The findings suggest that it is worth applying combination techniques in phoneme-level speech recognition systems because they will hopefully produce better scores, hence improved results. In the future we plan to investigate advanced combination methods, focusing on word (phoneme sequence) recognition where another dimension of possible knowledge source aggregation arises.

References

1. L. Breiman, *Bagging Predictors,* Machine Learnings, Vol. 24, No. 2, pp. 123-140, 1996.
2. C. M. Bishop, *Neural Networks for Pattern Recognition,* Oxford University Press, 1995.
3. T.G. Dietterich, *Machine-Learning Research: Four Current Directions,* The AI Magazine, Vol. 18, No. 4, pp. 97-136, 1998.
4. R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification,* John Wiley & Sons Inc., 2001.
5. A. K. Jain, *Statistical Pattern Recognition: A Review,* IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 22. No. 1, January 2000.
6. J. Kittler, M. Hatef, R.P.W. Duin, J. Matas *On Combining Classifiers,* IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 20. No. 3, March 1998.
7. Kocsor et al *A Comparative Study of Several Feature Transformation and Learning Methods for Phoneme Classification,* International Journal of Speech Technology, Vol. 3. pp. 201-215, 2000.
8. J.R. Quinlan, *C4.5: Programs for Machine Learning,* Morgan Kaufmann, 1993.
9. R.E. Shapire, *The Strength of Weak Learnability,* Machine Learnings, Vol. 5, pp. 197-227, 1990.
10. V. N. Vapnik, *Statistical Learning Theory,* John Wiley & Sons Inc., 1998.
11. David H. Wolpert, *Stacked Generalization,* Neural Networks, Vol. 5, pp. 241-259, 1992.