SPEAKER RECOGNITION USING ADAPTIVELY BOOSTED DECISION TREE CLASSIFIER

Say Wei FOO, ***senior member IEEE***
School of Electrical and Electronic Engineering
Nanyang Technological University
Singapore 639798

Eng Guan LIM
Defence Science and Technology Agency-DIS
1 Depot Road
Singapore 109679

ABSTRACT

In this paper, a novel approach for speaker recognition is proposed. The approach makes use of adaptive boosting (AdaBoost) and C4.5 decision trees for closed set, text-dependent speaker recognition. A subset of 20 speakers, 10 male and 10 female, drawn from the YOHO speaker verification corpus is used to assess the performance of the system. Results reveal that an accuracy of 99.5% of speaker identification may be achieved.

1.INTRODUCTION

One of the two most important aspects of a speaker recognition system is the identification of discriminating features representing the specific characteristics of the voice of the speakers. Another important aspect is the choice of classifier. [1,2] These two are critical to the overall performance of any speaker recognition system.

In this paper, the application of adaptively boosted C4.5 decision trees classifier [3] to speaker recognition is presented.

2.THE SYSTEM

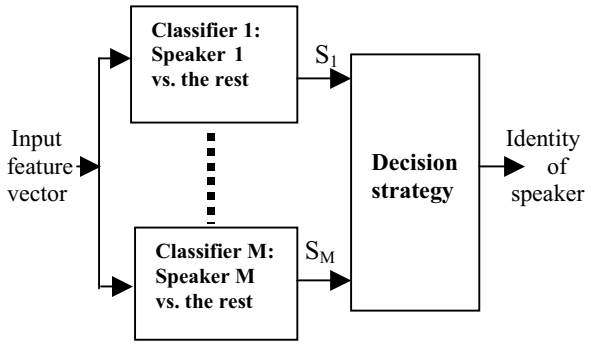

Fig.1 Block diagram of system

The block diagram of the system is given in Fig.1. At the front end, the analogue speech signal is first filtered to reduce frequency components higher than 4 kHz. The filtered signal is then sampled at 800 samples per second and digitally coded with an 8 bit μ-Law quantizer. A speech detection algorithm is applied to eliminate the silence intervals. The remaining samples are segmented into frame size of 256 samples (32ms) with an overlap of 128 (16ms) samples between adjacent frames.

Linear Predictive Cepstral Coefficients (LPCC) [4] are computed and used as features of speech segments. Coefficients of order 10 and their respective delta cepstral coefficients form a composite feature vector of 20 components. A typical text used in the experiments consists of 100 segments; the feature vectors of the segments are referred to as examples of the utterance.

For a system with *M* speakers to be recognized, the structure comprises *M* classifiers. An adaptively boosted binary classifier is created for each speaker. Each classifier is trained with the true speaker's samples against samples of the rest of the speakers. This structure of each classifier is based on "one speaker against the rest" dichotomy. The structure has the advantage of breaking down a complex multi-classification problem into *M* simpler binary classification problems that often facilitates the learning process of the classifier being used. The outputs from the M classifiers are then processed according to a desired decision strategy. The details are given in the subsections that follow.

3. ADAPTIVE BOOSTING

AdaBoost [5]-[9] belongs to the class of boosting algorithms implemented using sample re-weighting where a single learning algorithm is called each time a new weighted distribution is applied to the training samples.

Let *S* be a set of *N* training examples defined as follows.

$$S = \{x_i, y_i\}_{i=1}^{N}, \quad x_i \in X, y_i \in Y = \{0,1\} \quad (1)$$

where *X* represents the set of input vectors x_i and *Y* represents the set of desired response vector y_i. In the case of binary classification, elements of *Y* can only take on a discrete value of either 0 or 1. A value of 1 indicates that the learning algorithm regards it as a positive example and a value of 0 indicates otherwise.

The AdaBoost algorithm applies a weighted distribution to its set of training examples before calling upon the weak learning algorithm. Firstly, we define the weight vector *W(t)* as

$$W(t) = \{w_i(t)\}_{i=1}^{N} \quad (2)$$

where $w_i(t)$ is the weight given to the i^{th} example. Let the weighted distribution at the t^{th} boosting cycle be denoted by $D(t)$. This distribution can be expressed mathematically as

$$D(t) = \{p_i(t)\}_{i=1}^{N} \quad (3)$$

where

$$p_i(t) = \frac{w_i(t)}{\sum_{i=1}^{N} w_i(t)} \quad (4)$$

Other important inputs for the AdaBoost algorithm include the choice of the weak learning algorithm and the number of boosting cycles *T*.

Prior to the start of the algorithm, the elements of the weight vector *W(t)* are initialized with a value of 1.0 such that *D(t)* is a uniform distribution. The weak learning algorithm *C* is then called upon and given the set of weighted training examples to construct a classifier with the aim of producing a decision/hypothesis that gives a low error rate with respect to the given distribution $D(t)$ over the training examples. We define the hypothesis (binary classification) provided by the classifier in boosting round *t* as

$$h_t(x) = \{0,1\} \tag{5}$$

where *x* can represent any training example.

Using the generated hypothesis, the error rate of the classifier can then be computed over the given weighted training set. This error rate is basically the sum of the distribution weights of the misclassified examples and can be calculated as follows

$$\varepsilon_t = \sum_{i=1}^{N} p_i(t) \,|\, h_t(x_i) - y_i \,| \tag{6}$$

The error rate may be reduced if the classifiers focus on achieving the correct response for examples that bear a greater weight at the expense of the not so "important" examples. Instead of minimizing ε_t, an intermediate parameter β_t defined in (7) is used.

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t} \tag{7}$$

It can be observed from (7) that

$$\beta_t = \begin{cases} [0,1] & \text{if } 0 \le \varepsilon_t \le 0.5 \\ >1 & \text{if } \varepsilon_t > 0.5 \end{cases} \tag{8}$$

The AdaBoost process terminates when $\beta_t \ge 1$ or equivalently if $\varepsilon_t \ge 0.5$.

The weight assigned to the classifier for boosting at round *t* is related to β_t as follows.

$$\alpha_t = \log\left(\frac{1}{\beta_t}\right) \tag{9}$$

From (7), it can be seen that the smaller the value of β_t hence the smaller the value of ε_t, will give a larger classifier weight. Thus the AdaBoost algorithm places more importance or weight on decisions by more accurate classifiers.

The weight vectors are then adjusted according to the following formula.

$$w_i(t+1) = w_i(t)\beta_t^{1-|h_t(x_i)-y_i|} \tag{10}$$

Note that for a correctly classified example, the weight is reduced by a factor of β_t (which is less than 1) and for an incorrectly classified example, the weight remains the same. It is also now apparent from (10) why AdaBoost terminates if ε_t=0.5 as this will lead to $\beta_t = 1.0$ and *W(t+1)=W(t)*. That is, there is no change to the weight distribution and further boosting will result in exactly the same thing happening again. If this is allowed to continue, we will end up with *T* identical classifiers since each classifier is built using identical training sets.

This procedure is repeated until either premature termination occurs or the specified number of boosting cycles is met. The latter is assumed here. The remaining task is to combine all the generated hypotheses $h_t(x)$ into a single final accurate hypothesis $h_f(x)$ that is a function of the classifier weights derived in (9). $h_f(x)$ can be expressed as

$$h_f(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \ge \frac{1}{2}\sum_{t=1}^{T}\alpha_t \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

(11) can be interpreted as a weighted majority-voting rule. A positive result or a value of 1.0 is produced if more than half of the total votes are in favor of *x*.

4. C4.5 DECISION TREE

The terms associated with decision trees include the leaf and the node. Technically speaking, the leaf is a terminating point of the decision tree and represents a particular class. The node represents an intermediate point within a tree where a decision has to be made to split the instances on hand into two or more subsets based on some criterion.

Hunt's tree construction concept [3] lays the foundation for much subsequent research work and is related to the more complex C4.5 algorithm [3], which constructs tree using a similar concept. Based on Hunt's technique, given a training set of *N* examples $\{x_1,...,x_N\}$ with *k* classes $\{C_1, C_2,..., C_k\}$, there are basically three conditions to consider at any point within the construction of a tree. If all the examples at the node belong to a particular class, then that node is declared a leaf node representing the class. If there are no examples in the node, the node is still declared a leaf node but the class has to be determined by other information. In the event of a mixture of examples belonging to different classes within the training cases, a solution have to be found to decompose the T training cases into subsets such that each set is converging towards a set containing only a single class.

In C4.5 algorithm, the method for performing the partitioning of training examples uses information-based techniques or entropy measures. Given a multidimensional feature vector, an attribute (a single component feature) is selected and sorted in descending or ascending order in terms of magnitude. The aim is to find a point of division, usually the midpoint between two values

$$m = \frac{x_i + x_{i+1}}{2} \tag{12}$$

where x_i and x_{i+1} represents two consecutive values of the attribute, that gives the maximum information gain.

However, in C4.5 the largest value in the set smaller than the midpoint is chosen rather than the midpoint itself. Suppose a specific splitting point is chosen and this Split is denoted as *S*. The information from this division can be obtained by first calculating the information info(T_i) conveyed in each subset T_i as follows

$$\text{info}(T_i) = -\sum_{j=1}^{k} \frac{freq(C_j, T_i)}{|T_i|} \times \log_2\left(\frac{freq(C_j, T_i)}{|T_i|}\right) \quad (13)$$

where $freq(C_j, T_i)$ denotes the number of cases belonging to class C_j in the subset T_i and $|T_i|$ is the total number of examples in the subset T_i. The total information $\text{info}_s(T)$ conveyed as a result of the Split *S* over the original training set T can then be expressed as a weighted sum of the information within each subset

$$\text{info}_s(T) = \sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \text{info}(T_i) \quad (14)$$

The information gained from the partitioning of the data set T using the Split *S* can then be obtained using

$$\text{gain}(S) = \text{info}(T) - \text{info}_s(T) \quad (15)$$

A normalizing parameter Split info (S) that measures the potential information generated by the Split *S* is further required to compute the gain ratio. The Split info (S) can be calculated by applying the following equation

$$\text{Split-info}(S) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} \times \log_2\left(\frac{|T_i|}{|T|}\right) \quad (16)$$

and subsequently the gain ratio can be computed as follows

$$\text{gain ratio}(S) = \frac{\text{gain}(S)}{\text{Split info}(S)} \quad (17)$$

The gain ratio for each possible division is computed for all the attributes and the split that gives the largest gain ratio is selected. This process is continued until all the subsets are classified as leaf nodes (subsequent splits does not result in further information gain) and no further splits are required.

Overfitting is a common problem among trees generated using C4.5 algorithm. It is not unusual for decision trees to achieve perfect accuracy over the training data and yet perform poorly when given an unseen data set.

Pruning is a method devised to address this issue. It is basically a bottom-up technique for simplifying a complex decision tree by removing unnecessary subtrees and replacing it with a leaf node or one of its branches. A decision has to be made at each subtree to determine if a replacement will be beneficial to its classification performance. In this project, C4.5 uses reduced error pruning for this purpose.

In reduced error pruning [3], the aim is to proceed with the subtree replacement if such an arrangement can bring about a decrease in the number of predicted errors (an estimate of the test errors). Each leaf can be described using two parameters namely *N*, the total number of training instances in the leaf and *E'*, the number of misclassified training instances. In a statistical sense, *E'/N* can be regarded as the probability of error of a particular set of sample data associated with this leaf. This information can then be used to derive an estimate for the upper limit of the population error rate using the confidence limits (with prespecified confidence level) for the binomial distribution. In C4.5, this upper limit is taken to be the predicted error rate. Given a subtree with *M* branches, we can compute the number of predicted errors as follows

$$E_p = \sum_{i=1}^{M} E_p^i \quad (18)$$

where

$$E_p^i = \begin{cases} N^i \times U_{cl}(E^i, N^i) & \text{if } i^{th} \text{ branch leads to a leaf node} \\ \sum_{j=1}^{L} E_p^j & \text{if } i^{th} \text{ branch leads to a subtree} \end{cases} \quad (19)$$

E^i is the number of misclassified training samples associated with the leaf node at the i^{th} branch, N^i is the total number of training samples covered by the leaf node and $U_{cl}(E^l, N^l)$ represents the upper limit defined earlier with confidence level *cl*. The pruning process terminates when no more beneficial subtree replacements can be found.

5.ADAPTIVELY BOOSTED C4.5 DECISION TREE

AdaBoost can be readily integrated into the C4.5 algorithm, as most of the information gain computation is dependent on the sample distribution. In the previously mentioned method in Section 2.4.2, the training examples have an unbiased weight of 1.0. With AdaBoost, each sample will now have a different weight and hence the total number of instances in any set i is now defined as

$$|T_i| = \sum_{i=1}^{n} w_i \quad (20)$$

which is the sum of the weights of all the examples in the set T_i where w_i is the weight of example i. The $freq(C_j, T_i)$ is modified to become the sum of weights of all the examples belonging to class C_j. These modifications results in changes to the information measures and a totally different tree could be constructed from the same data set.

6. DECISION STATEGY

The system is tested for both speaker identification and verification. For speaker identification tests, a winner-takes-all approach is adopted by evaluating the score of the test utterance against each speaker tree model and subsequently selecting the speaker that corresponds to the highest score. Mathematically, the rule may be expressed as:

$$\hat{S} = \arg\max_{1 \le j \le M} W_j \tag{21}$$

The score W_j for the j^{th} speaker tree model is calculated using

$$W_j = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{22}$$

where N is the number of feature vectors in the test utterance and y_i is the output corresponding to the *ith* feature vector.

The other alternative is to use a competitive-based approach where the speaker's voice is matched against not only the claimed model but other models as well. A decision is then made based on how well the output of the claimed model fare against the other models.

A measure of how well the claimed model does is expressed in terms of the ratio between the score produced by the claimed model and the score produced by the next most competitive model.

$$R = \frac{S_v}{S_c} \tag{23}$$

where S_v is the score of the claimed model and S_c is the score of the next most competitive model. The ratio R is then matched against a predefined threshold T_r (e.g. a value of 1.0) that can be speaker independent or speaker specific. For the system proposed in this paper, a speaker independent threshold has been adopted. The decision logic is defined as

$$V = \begin{cases} 1 & \text{if } R \ge T_r \\ 0 & \text{if } R < T_r \end{cases} \tag{24}$$

7.EXPERIMENTS AND RESULTS

The verification and identification performance of the proposed system is assessed using data from the YOHO speaker verification corpus designed specifically for speaker verification purposes by the US government.

As a preliminary study, tests are carried out on a subset of 20 speakers, 10 males and 10 females, from the database.

A binary classifier is used to represent each speaker giving a total of 20 binary classifiers per system. Pruning at 25% confidence level is performed after each tree is constructed. The verification and identification performance of the unboosted C4.5 classifier system with and without the application of competitive-based scoring are summarized in Table I.

Table I: Accuracy of speaker recognition

	Non-competitive scoring		**Competitive-based scoring**	
	True speaker rejection	Imposter acceptance	True speaker rejection	Imposter acceptance
C4.5	5.0%	0.82%	3.0%	0.32%
Boosted C4.5	2.25%	0.76%	0.5%	0.05%

Based on the results indicated in Table I, impressive results are achieved with boosting. Without competitive-based scoring, a reduction of 2.75% in true speaker rejection rate and 0.06% in impostor acceptance rate are obtained. The improvement is more dramatic when competitive-based scoring is incorporated into the system. With competitive-based scoring, a reduction of 2.5% and 0.27% is achieved over the unboosted system. The identification accuracy (100% minus true speaker rejection rate) is also increased by 2.5% to a high 99.5% (which is equivalent to two identification mistakes over the 400 test samples used).

8.CONCLUSION

A new method of speaker recognition using adaptively boosted C4.5 Decision Tree classifiers is proposed. It can be concluded that the application of AdaBoost algorithm leads to significant improvements in recognition accuracy over the base classifiers. With the adaptively boosted C4.5 Decision Tree classifiers, very high degree of accuracy can be achieved in speaker identification and speaker verification.

9. REFERENCES

[1] J.P. Campbell. Speaker Recognition: A Tutorial. In IEEE Proceedings, 1997.

[2] J.M. Naik. Speaker Verification: A Tutorial. In IEEE Communications Magazine, 1990.

[3] J.R. Quinlan. C4.5 Programs For Machine Learning. Morgan Kaufmann Publishers. 1993.

[4] Liu. Li, H. Jialong, G. Palm. On the Use of Residual Cepstrum in Speech Recognition. In IEEE Proceedings International Conference on Acoustics, Speech and Signal Processing, vol. 1, pp. 5-8, 1996.

[5] Y. Freud. Boosting a Weak Learning Algorithm by Majority. In Information and Computation 121(2), pp. 156-285.

[6] R.E. Schapire. A Brief Introduction to Boosting. In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, pp. 1401-1405, 1999.

[7] Y. Freud, R.E. Schapire. Experiments with a New Boosting Algorithm. In Proceedings of the Thirteenth International Conference on Machine Learning, pp. 148-156, 1996.

[8] T.G. Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting and Randomization. In IEEE Transactions on Information Theory, pp. 21-27, 1998.

[9] J.R. Quinlan. Bagging, Boosting and C4.5. In Proceedings of the Thirteenth International Conference on Artificial Intelligence. pp. 725-730, 1996.