Intelligent Rapid Voice Recognition using Neural Tensor Network, SVM and Reinforcement Learning

Davis Wertheimer, Aashna Garg, James Cranston
{*daviswer, aashnagarg, jamesc4*}*@stanford.edu*

***Abstract*—We propose two machine learning improvements on the existing architecture of voice- and speaker- recognition software. Where conventional systems extract two kinds of frequency data from voice recordings and use the concatenation as input, we propose two methods to allow the input vectors to interact multiplicatively. The first is a Neural Tensor Network layer under a softmax classifier, and the second is a constrained variant of the Neural Tensor Network with reduced dimensionality. We compare these methods with the current approach, using SVMs on the concatenation of extracted data vectors. Second, we trained a shallow neural network on a Q-learning framework in order to intelligently and dynamically minimize the amount of audio required to make an accurate classification decision. While the neural network architectures failed to improve on the existing SVM model, the Q-learner did learn to dynamically minimize audio sampling while improving on the accuracy of the SVM system.**

Keywords*—function approximation, Markov Decision Process, MDP, Mel Frequency Cepstral Coefficients, Neural Network, Neural Tensor Network, MFCC, policy optimization, Q-learning, security, signal processing, Support Vector Machine, SVM, voice authentication, voice recognition.

I. Introduction

Conventional voice- and speaker- recognition systems extract two kinds of frequency data from voice recordings: MFCCs, and the delta (rate of change) of each MFCC value. Machine learning systems, typically SVMs, are then trained on the concatenation of those two vectors. While these systems do learn to distinguish between speakers based on the signs and magnitudes of the MFCC and delta values, they do not capture the intuition that each delta corresponds to a particular MFCC value. We can use this information by explicitly defining multiplicative relationships between the corresponding elements, taking their product in addition to scaling them by the appropriate learned weight parameter. This allows us to capture information on to what degree the MFCCs and their deltas are adopting similar values. We hypothesize that by introducing explicit multiplicative interaction to our machine learning architecture, via Neural Tensor Network, we can improve accuracy for the speaker recognition task.

There are two lines of reasoning to support this claim. The first is that by introducing multiplicative interaction, we are incorporating new, useful information on the relationship between MFCCs and their deltas. The second is that we avoid overfitting to the training data by constraining our system to learning only linear and multiplicative relationships between the two vectors. SVMs using the Gaussian kernel could possibly overdetermine the problem by implicitly learning higher-degree monomial relationships which are likely to be uninformative. We predict that this leads to unnecessarily convoluted decision boundaries.

In order to evaluate our hypothesis, we train two Softmax Neural Tensor Networks consisting of a softmax layer over a Neural Tensor Network layer, and compare performance to an array of one-vs-one binary SVMs performing multiclass classification. We test performance on sets of 2 to 9 speakers in order to measure how the three systems handle incremental growth in the number of classes.

The second component of our study is a reinforcement learning system which attempts to minimize the amount of time and computation involved in the speaker recognition task, without sacrificing the accuracy of the highest-performing classification system from part 1. The part 1 classifiers act as high-accuracy, high-cost systems which the higher-level decider should utilize as little as possible. This forms a Markov Decision Process with a continuous state space, where states consist of a confidence rating for each class, plus the number of audio samples already taken, and actions consist of either sampling again, or making the prediction of whichever class has the highest current confidence rating. The reinforcement learner for this MDP uses the Q-learning algorithm and a shallow neural network to approximate the expected reward for each state-action pair. The trained system dynamically decides whether to predict or continue sampling based on its current confidence judgments and the amount of time elapsed since start.

Finally, we used the classifier and reinforcement learner to build a system which can recognize peoples voices in real time, as they speak.

II. Related Work

During the recent years, there have been many studies on voice recognition using several features and techniques.Some of the previous research studies in the field of voice recognition include using SVMs along with Linear Discriminant Analysis with MFCCs which increases the accuracy and prediction rate

Andrew Ng, Associate Professor with the Department of Computer Science, Stanford email:(see http://www.andrewng.org/)

Percy Liang, Assistant Professor with the Department of Computer Science,Stanford email:(see http://cs.stanford.edu/ pliang/)

Justin Fu is our CS221 project mentor with the Department of Computer Science, Stanford e-mail: justinfu@stanford.edu.

Junjie Qin is our CS229 project mentor with the Department of Computer Science, Stanford e-mail: (see http://web.stanford.edu/ jqin/).

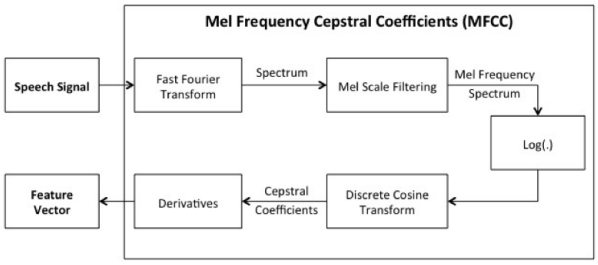

Fig. 1. MFCC block diagram [7]

over SVMs alone [1]. Yu, Li and Fang [2] propose context-dependent deep tensor neural network to reduce error rate and compares with deep neural network Hidden Markov Models. A new pattern classification method called the nearest feature line (NFL) is proposed in Li [3], where the NFL explores the information provided by multiple prototypes per class. Audio features like MFCC, ZCR, brightness and bandwidth, spectrum flux were extracted (Lu, Zhang, & Li, 2003 [4]) , and the performance using SVM, K-nearest neighbor (KNN), and Gaussian mixture model (GMM) were compared. Audio classification techniques for speech recognition for unsupervised multi-speaker change detection are proposed in Huang and Hansen [5]. Two new extended-time features: variance of the spectrum flux (VSF) and variance of the zero-crossing rate (VZCR) are used to pre-classify the audio and supply weights to the output probabilities of the GMM networks. The classification is then implemented using weighted GMM networks. A speech recognition system using Artificial Neural Network technique called Reservoir Computing (RC) was given by Abdulrahman [6]. Here they aim to improve the performance of the conventional RC approach by developing two approaches ESNSVMs (Echo State Networks with Support Vector Machines) and ESNEKMs (Echo State Networks with Extreme Kernel Machines).

III. Dataset and Features

Our data set consists of audio clips collected from 9 students on campus, 3 female, 6 male. We record each person reading aloud the first paragraph of Lewis Carrols Alice in Wonderland, followed by a 15-20 second passage randomly selected from another section of the same book. This way, we have for each speaker one sample where the word content is the same, and another where it is different. Pre-processing the data consists of slicing each audio sample into a collection of overlapping 30-ms clips, each an individual data point for that speaker/class. We then filtered out empty clips (max volume ¡ 5% of max volume for the entire recording), and calculated the Mel Frequency Cepstral Coefficient (MFCC) values for each of the remaining clips. Finally, we calculate the MFCC deltas, or the trajectories of the MFCC coefficients over time. Together, these form the inputs to the classification systems, each returning a single prediction per clip. Calculating MFCCs involves seven steps (Figure 1), each described briefly below:

1) Framing: Each audio sample, about 10-15 seconds long, is split into 30-ms time windows, with a 15-ms jump rate. Any discrete point within the original audio sample is then contained in exactly two clips.
2) Fast Fourier Transform (FFT): FFT is performed on each clip, and the real component is rescaled to Mel frequencies (which corrects for the fact that pitch is logarithmic), using the following equation:
$$M(f) = 1125 \ln(1 + f/700)$$
3) Computing Mel filterbanks: The amplitudes of the scaled Mel frequencies are passed through each of 26 triangular filters, staggered over the Mel frequency range in the same overlapping manner as the sliding windows on the original audio clip (ensuring that each amplitude is weighted equally, since adding any two consecutively filtered bands produces the original strength audio clip in the overlapping area).
4) Every amplitude value is then summed within each filtered band, yielding 26 energy density estimates over the range of produced frequencies.
5) Discrete Cosine Transform: A Discrete Cosine Transform (DCT) yields the final MFCCs, representing the energy distribution as a sum of generating frequencies in the time domain.
6) The second input vector is a delta vector on the MFCC values. We use the standard formula:
$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2}$$
which represents the rate of change of the generating forces behind the vocal intensity distribution. The more distant windows are weighted twice as heavily because the closer windows each overlap with the one under examination, sharing 50% of their generating audio signal.
7) Finally, we log-scaled both MFCC and delta vectors, since they ranged from decimal scales to the hundreds of thousands. Final input vectors range from about [-10,10].

We then held out one-fifth of the dataset to use as testing data. All of our subsequent systems, except the real-time classifier, were trained using the remaining 80%.

IV. Methods

Our four machine learning systems are described in detail below.

A. Neural Tensor Network

A Neural Tensor Network (NTN) allows two input vectors to interact in a non-linear fashion, by representing the weight matrix as a 3-dimensional tensor. Activation values use the formula:
$$g(e_1, R, e_2) = u_r^\top f\left(e_1^\top W_R^{[1:k]} e_2 + V_R \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + b_R\right)$$

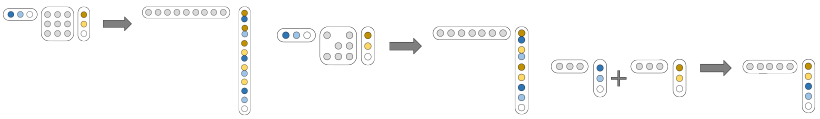

Fig. 2. Low-dimensional examples of the input supervectors used by our systems and their equivalence to traditional neural network / neural tensor network formulations. The first is the NTN supervector, the second is SNN, and the third is SVM. The left-hand side of each figure represents the traditional setup while the right-hand side gives the equivalent supervector.

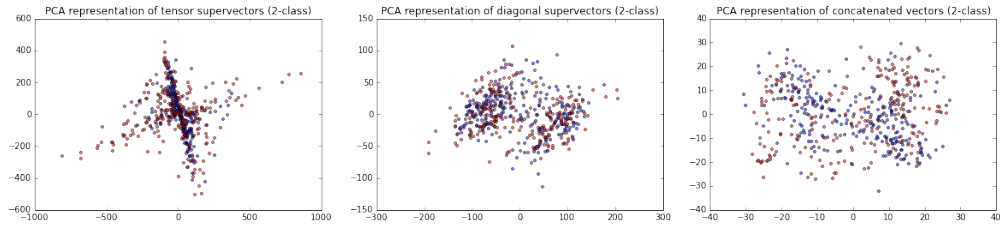

Fig. 3. PCA plots for NTN, SNN, and SVM input supervectors, respectively. 2 classes are shown here. Progressively more structure is imposed as the size of the supervector increases.

where e_1 and e_2 are same-dimensional input vectors, W_R is a square matrix slice of tensor W, V_R is a vector slice of weight matrix V, b_R is a bias term, and f is the activation function. Because tensors are difficult to represent in Python, we chose to re-represent the input vectors as a single, mathematically equivalent input supervector, which could then be fed into a standard neural network. To generate the new input vector, we append a bias term of 1 to each input vector, then calculate their outer product. This creates a square matrix containing every value of the first vector multiplied against every value of the second, plus every value in the first vector, every value in the second vector, and a bias term (due to the bias terms in both vectors). Using a standard weight matrix on these values is equivalent to taking the tensor product, and adding the standard weight product and a bias term. See Figure 2 for a low-dimensional example. We replaced the 2x26-dimensional input of the tensor network with the $(26+1)^2 = 729$-dimensional input of a standard network. We use 26 hidden layer nodes, plus a bias, since there are 26 delta/MFCC pairs in each input and extracting additional relationships is likely to be uninformative. We used mean-squared error as the objective function and hyperbolic tangent as the activation function.

B. Constrained Tensor Network / Softmax Neural Network

Due to the large dimensionality of the NTN we decided to reduce the size of the supervector. Since multiplicative interactions are only of interest between each MFCC and its corresponding delta, the rest can be discarded. We used a supervector equal to the MFCC and delta vectors concatenated with their elementwise product, plus a bias. This is mathematically equivalent to a tensor layer with the tensor holding only zeros except on the diagonal, the final row, and the final column (and the inputs have bias terms). Fig 2 gives a low-dimensional example. While both systems are technically tensor and softmax networks, we refer to the higher-dimensional system with the full tensor as the NTN and the lower-dimensional system as the Softmax Neural Network (SNN). The input vector for the SNN consists of $2 * 26 + 26 + 1 = 79$ features, as opposed to the NTN's 729. The hidden layer is identical to that of the NTN, for the same reasons. Activation function remained hyperbolic tangent, and we used mean-squared error as the objective.

C. Support Vector Machine (SVM)

In addition to using the NTN and SNN to make class predictions, we compare the algorithms' performances to SVMs. The SVM we used derived from the optimization algorithm used in class:

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & \alpha_i \geq 0, \ i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned}$$

Because an SVM can take only a single input vector, we concatenate the MFCC vector with the delta vector. Visualizations of the supervectors in the data set for all three models are given in Figure 3. Since SVMs can only perform binary classification, we use a series of SVMs which each predict whether or not the sample belongs to a single given class versus another, and take the result with the highest overall certainty.

D. Q-Learning and Function Approximation

Our higher-level audio-sampler and decision-maker explores a continuous state space, necessitating a function approximator to evaluate the expected reward for each state-action pair. Our state space consists of confidence ratings for each class, plus the number of samples taken so far for the particular query being processed. Because expected reward cannot be expressed in terms of a linear combination of these values (they all increase with time monotonically and roughly proportionately),

we use a simple 3-layer neural network approximator with hyperbolic tangent activation function and objective function following the form:

$$\min_{w} \sum_{(s,a,r,s')} \left(\hat{Q}_{\text{opt}}(s,a;w) - (r + \gamma \hat{V}_{\text{opt}}(s')) \right)^2$$

and update rule for each (s, a, r, s') as follows:

$$w \leftarrow w - \eta \left[\hat{Q}_{\text{opt}}(s,a;w) - (r + \gamma \hat{V}_{\text{opt}}(s')) \right] \phi(s,a)$$

Our state feature vector consists of the sorted, cumulative prediction strength for each class, plus the number of samples taken since the beginning of the authentication process, plus an action term, plus a bias term. Committing to a prediction set the action term to 1, while resampling set the action term to -1. Thus our network consisted of a size n+3 input layer, n hidden nodes, and a single output, where n is the number of classes. We use a size n hidden layer for the same reason we use 26 on the predictor: extracting more than one activation value per class is likely to be uninformative.

V. EXPERIMENTS AND RESULTS

We collected two audio clips from each speaker, one in which they read the same passage, and the other with different passages. Training the classifiers on either data set produced no discernible net difference for any of the systems, so for all subsequent analyses we trained the classification systems on the combination of both sets.

A. Classifiers

We trained our three classification systems on progressively larger subsets of our data set, first with 2 speakers, then 3, then 4, up to 9, in order to evaluate how each handles stress in the form of additional classes. Both neural networks trained for 250 passes over the training set, long enough to reach convergence on all training sets. Step size was inversely proportional to the number of classes (and number of updates per pass over the training set) and annealed at a rate inversely proportional to the number of passes already performed. While we experimented with L2 normalization and momentum, neither improved performance or learning rate for either model. We trained our binary SVM array to convergence on the same progressively increasing data sets, with normalization constant 1 and final stopping epsilon equal to .001.

Our metrics for the classifier experiments are training accuracy, test accuracy, F1 score on the testing data, and percent improvement over random guessing. Because chance performance decreases monotonically with the number of classes, it could be the case that performance relative to the expected level of error is increasing even when the total accuracy drops. The formula for percent improvement over chance is given by $1 - \frac{1-a}{1/n}$, where a is test accuracy and n is the number of classes. This indicates the proportion of error from random guessing that vanishes when the algorithm is employed. A comparison of the results for 9-class classification is given in Table 1, and Figure 4 shows the progression of these metrics

TABLE I. COMPARISON OF NTN,SNN AND SVM BASED ON 9-CLASS EXPERIMENTS

9-class statistics	NTN	SNN	SVM
Train Accuracy	18.2	38.6	47.8
Test accuracy	14.8	35.8	44.6
F1 score	17.3	37.2	44.4
Improvement	4.2	27.8	37.7

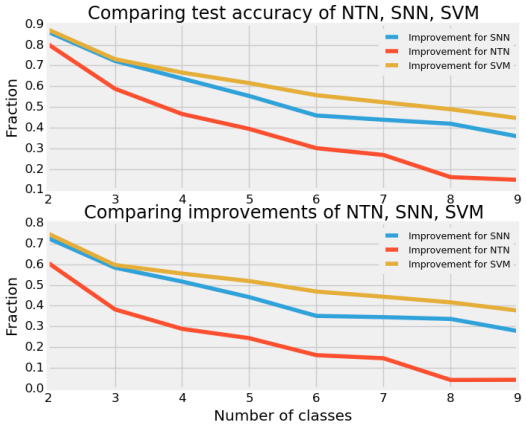

Fig. 5. Performance comparisons between NTN,SNN,SVM

over increasing numbers of classes. A comparison for all three systems is given for test accuracy and percent improvement from chance in Figure 5.

Unfortunately, neither neural network outperforms SVM. As expected from the large dimensionality, NTN overfits, evidence by the gap between training and test accuracy, but the lower-dimensional SNN variant fixes this problem. We suspect that SVM outperforms neural networks because the supervectors create highly non-convex objective functions. SGD is not equipped to handle this, and we see from the confusion matrices in Fig 5 that NTN and to a lesser extent SNN become trapped at local maxima. For 9-class classification, the NTN simply assigns all inputs to the two speakers with the most training samples. SVM does not overfit by making use of extraneous information, so we used SVM as the underlying prediction system for the minimal-sampling Q-learner.

B. Reinforcement Learner

The Q-learning neural network, like the classifiers, was trained on successively larger sets of speakers, for 1000 passes,

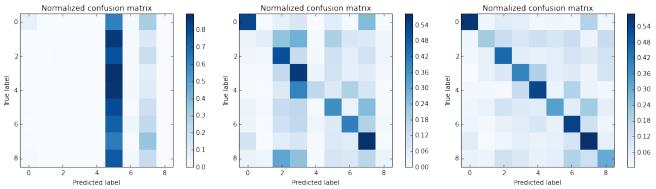

Fig. 6. Confusion matrices for 9-class NTN, SNN, and SVM, respectively

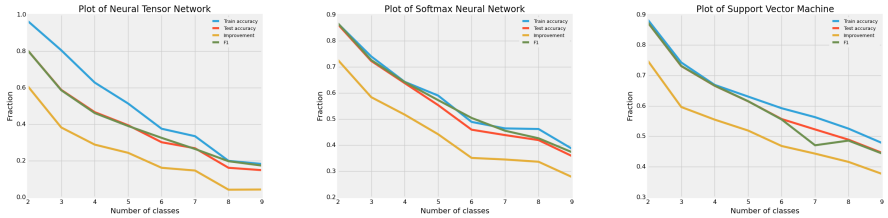

Fig. 4. Test accuracy, training accuracy, F1 score and percent improvement for NTN, SNN and SVM, respectively. Note that y-axis scaling is not preserved between figures.

where a pass consists of one epsilon-greedy walk over each speaker's training audio. Step size was inversely proportional to the number of classes, and annealing rate inversely proportional to the number of passes already performed. Epsilon annealed at a rate inversely proportional to the square root of passes performed, in order to maintain randomness in late stages. L2 normalization was employed with decay constant annealing to zero at a rate proportional to step size, producing more stable training behavior. Rewards were fixed at 1/-1 for correct/incorrect predictions, and a small, controllable penalty for resampling. Gamma was fixed at 0.995. While gamma could act as the resampling penalty parameter with the reward fixed at zero, we found that constant gamma and varying penalty produced faster convergence.

The Q-learner converged consistently to a single narrow range of samples per decision for all numbers of classes, showing that it dynamically responds to confidence levels as it samples. However, because the Q-learner depends on random initialization of weights and random walks over randomly shuffled training data, setting a single value for the negative resampling reward produced a fairly wide range of models with different mean samples-per-decision. We trained six Q-learners on each training set, with penalty parameters equal to $-0.1 * \frac{1}{2^n}$ where n ranges from 0 to 5. The resulting samples-per-decision rates fluctuated too widely to extract meaningful relationships between results and parameters, other than that larger penalties lead to faster, less accurate systems, while smaller creates the opposite. However, we did log the minimum number of average samples-per-decision in models achieving greater than 99% test accuracy, and these results are given in Figure 7. This minimum learned sample rate steadily increases with the number of classes, showing that the Q-learner, despite its wide range of outputs, gracefully handles decreasing accuracy rates in the underlying classifier.

C. Real-Time Classification

Finally, we used the trained Q-learner and SVM classifier to build a system that samples audio, performs MFCC, resamples dynamically as required, and makes speaker classification predictions in real time. We tested it on ourselves, using 3-class algorithms. The system was able to make highly accurate predictions with only a short lag time (less than 1 second), though we were unable to record the exact lag times and accuracies since the system was making predictions in real time, far too quickly to log or record the results. Regardless,

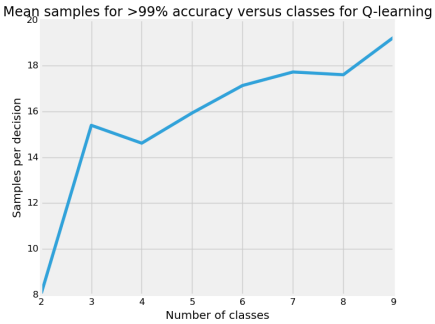

Fig. 7. Average number of samples-per-decision, among the fastest models still achieving above 99% accuracy for each number of distinct speakers.

this shows that a trained classifier under a subsequently trained sampler is a viable model for real-time speaker diarization, even for large numbers of speakers, since the minimum number of samples required for the Q-learner to achieve 99% accuracy increased only slightly with the number of classes.

VI. Conclusion And Future Work

Unfortunately, we were wrong in our prediction that explicitly defined supervectors would minimize overfitting relative to the implicitly defined supervector of a Gaussian kernel SVM. The supervectors create highly non-convex objectives under mean-squared error, with the result that SGD leads to highly local optima. SVM remains the best system for speaker classification of single clips. However, our Q-learning system shows that an intelligent data sampler can effectively balance accuracy with speed, achieving greater than 99% test accuracy for up to 9 speakers with less than .6 seconds worth of audio input per decision. Results indicate that the Q-learning system is highly robust and can be expected to maintain this level of performance for higher numbers of speakers. We were then able to construct an intelligent, real-time, highly accurate speaker diarization engine. Further research in this area would include the incorporation of non-MFCC data into the classifier input vectors, and additional experimentation with Q-learning training parameters in order to gain finer control and produce more stable behavior.

References

[1] Aamir Khan , Muhammad Farhan, Asar Ali, *Speech Recognition: Increasing efficiency of SVMs*, 2011

[2] Dong Yu, Li Deng, Frank Seide, *The Deep Tensor Neural Network With Applications to Large Vocabulary Speech Recognition*, 2012

[3] Li, S. Z. *Content-based audio classification and retrieval using the nearest feature line method. IEEE Transactions on Speech and Audio Processing* 2000

[4] Lu, L., Zhang, H.-J., Li, S. Z.*Content-based audio classification and segmentation by using support vector machines. Multimedia Systems*, 2003

[5] Huang, R., Hansen, J. H. L. *Advances in unsupervised audio classification and segmentation for the broadcast news and NGSW corpora. IEEE Transactions on Audio, Speech and Language Processing* 2006

[6] Abdulrahman Alalshekmubarak *Towards a robust Arabic speech Recognition system based on reservoir computing*, 201

[7] http://recognize-speech.com/feature-extraction/mfcc