NLP Final Project Report on Speech Recognition

Yanshan Guo and Lucy Lu

February 25, 2016

1 Introduction

Our project is a simple speech recognition program that recognizes certain phones from given sound inputs. Right now, the program is able to recognize the following phones: ah, ao, eh, ey, f, ih, iy, k, n, ow, r, s, t, th, uw, w, z(CMU phonetics Dictionary). This project involves three main components:
1). converting sound waves to their spectral representations
2). Training an acoustic model
3). Using the trained acoustic model to predict what phone an input sound file represents.
In the following sections, we will explain how we implement each component and discuss the result at the end.

2 Feature Extraction:MFCC Vector

Sound waves are converted to their spectral representations: MFCC vectors. This representation provides a 39-dimensional vector of features for every 30 ms of the input wave file. For this assignment, we will use the first 13 cepstrum coefficients of the MFCC vector to represent an observation. The window type of our MFCC calculation is Hamming window. For implementation, we used the Python package scikits.toolbox to convert sound waves into their spectral representations.

3 Building HMM Model

3.1 HMM Structure and Transition Matrix

In our HMM model, each phone is represented by three HMM states: a beginning, a middle, and an end state, each representing a stage within a phone sound. The starting state of the HMM only goes to the beginning states of phones and only end states of phones go to the ending state of the HMM. For non-begin and non-end states, they have 0.5 probability of going back to itself or proceeding to the phone in next stage.

4 Training the Acoustic Model

4.1 Computing Acoustic Likelihood for Emission Matrix

To deal with the issue that sound is represented as real-valued vectors, a probability density function (pdf) is computed to construct the acoustic model. In our program, we use the multivariate Gaussian probability density function to compute the acoustic likelihood. For each state j, $P(j|o_t) = f(o_t|\mu_j, \sigma_j)$, where f is the multivariate Gaussian probability density function for state j with mean μ_j and variance σ_j. In Multivariate Gaussian, the mean is represented as a vector and the variance represented as a covariance matrix. It is worth noting that in order to compute the covariance matrix faster, one important assumption that features in different dimensions do not co-vary is made. Thus, in our program, we stored the covariance matrix as a one-dimensional vector with its entries being the diagonal entries of the covariance matrix.

In implementation, we used the log value of $f(o_t|\mu_j, \sigma_j)$ to avoid the underflow problem. Since the prediction of phones is done by comparison, using the log value will not affect the result.

4.2 Training Data

For this project, we did not have enough time to get a large training data set. We manually get phonetics from webs by recording the sound in computer and then edited them in Audacity. We ended up using 3 sound files for training each state. They are named as aa1,aa2,aa3 (in the example of aa) in the train folder.

5 Baum-Welch Embedded Training

To train the mean and covariance in multivariate Gaussian, we used the Baum-Welch algorithm. Notice that for the purpose of this specific project, we only need to train the emission matrix. Since the transition matrix could be predetermined, we do not train transition matrix within the Baum-Welch training algorithm. The Baum-Welch training algorithm at each iteration updates the mean and covariance values of each state j. We initialize the mean and covariance value of each state to be the global mean and global covariance of all input vectors(MFCC vectors representing sounds). Denoting transition matrix to be A, and emission matrix to be B. The process iterates as the following:
Initialize A and B matrices,
E-step:
Initialize an empty dictionary G for storing the observation o_t and $\gamma_j(o_t)$.

-For each observation, run forward and backward algorithms to build α and β matrices.

-For each state j and time step t, store ($\gamma_j(o_t)$, o_t) as in dictionary $G[state]$.

M-step
For each state j

-μ_j is updated by $\frac{\sum_1^T o_t \gamma_j(o_t)}{\sum_1^T \gamma_j(o_t)}$

-σ_j is updated by $\frac{\sum_1^T \gamma_j(o_t(o_t-\mu)(o_t-\mu)^T)}{\sum_1^T \gamma_j(o_t)}$
Iterate until means and covariances converge

Notice that for speech recognition, we need to train the emission matrix for each phone separately and then append them together in the final emission matrix. We got stuck on this for a long time and were very confused about why every state eventually ended up with same Gaussian functions. After we figured out the problem,we modified our program to train each phone separately and this new method yielded much better result and also it actually makes much more sense.

6 Decoding

6.1 Viterbi Algorithm

In order to return the most likely state sequence, Viterbi algorithm is used. The algorithm is similar to the one we learnt in class and we simply implement it from scratch with some modifications to deal with underflow.

6.2 Prediction

After we get the mostly likely state sequence, we predict the phone of the given sound wave input to be the phone that appears the most in the entire sequence.

6.3 Testing Data Set

Due to the time constraint, we only have 1 test file for each phone. They are wav files ending with 4 in the train folder.

7 Result

Figure.1 is the result we get for predicting phones. Correct predictions are highlighted in blue. We get a 50 percent accuracy for this project.

8 Conclusion

We are pretty content about the result we get from this program. However, we can not explain why phone iy is predicted to be the correct phone so many times. We are suspecting this has something to do with the length of the sound input, yet we do not have any proof and idea of improvement for this problem.

Phone	Prediction
ah	ah
ao	iy
eh	eh
ey	iy
f	iy
ih	eh
iy	iy
k	k
n	n
ow	iy
r	iy
s	iy
t	t
th	th
uw	iy
w	w
z	iy

Figure 1: Prediction Result

9 Packages

One thing to notice is that we used many math packages in this program and it does require some work to install all of them. Numpy and Scipy can be downloaded from their official websites and installed easily. For Scikits.talkbox.features, we have to install the set up tools for python first and then use it to install the talkbox module. We include all of these packages in our code zip folder.
scipy.stats
numpy
scikits.talkbox.features
scipy.io

10 Further Improvement

We can improve our final project is many ways. To increase the accuracy of the prediction, we can implement the Gaussian Mixture Model. Also, we could increase the size of both the training and the testing data set. Another aspect to work on is to extend the program so that it can recognizes digits and words. This will entail building a more complex transition

and emission matrices.