Analysis of phonations for speaker identification

Akhila Ananthram

Abstract:

The goal of this project was to see if there are any key features of waveform that can be used for speaker identification in the switchboard corpus. This involved taking the discrete Fourier transform of the waveform and extracting the F0 and spectral tilt values. These were then used to create a 2D feature vector for the phone, 'aa'. These feature vectors were put in a support vector machine to create a classifier that would separate the phones based on who said it. The classifier created was 70% accurate on the training set and the test set. All the code for this project was done in Matlab and used the Voicebox Toolbox.

Extracting the phones from the corpus:

Selecting the desired phone

Originally I looked for 'ah', a low central vowel. I changed to 'aa' as 'ah' is a more common filler phone. People use 'ah' when they are unsure of what to say, and so many times when it is used the duration goes on longer than when it is used in a word. Thus, 'aa' seemed like a better choice. When I wrote the code for this part, I set it up nicely so I only had to change one line to find the new phone and do any analysis. In the string of phones, I found the starting index of every occurrence of that phone.

Eliminating outlier phones

I eliminated all that were not primary stressed, so stress would not be a factor in my feature vector. I then sorted them by speaker. However, even though I switched to 'aa', there were several tokens that were far from the mean. Because of this, I needed to eliminate phones that were too long or too short. To figure out what durations I should keep, I plotted histograms of the durations.

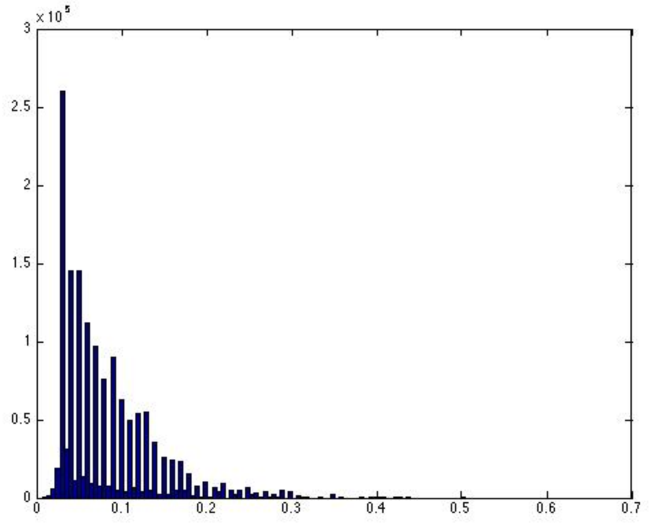

Figure 1: Histogram of durations over all speakers

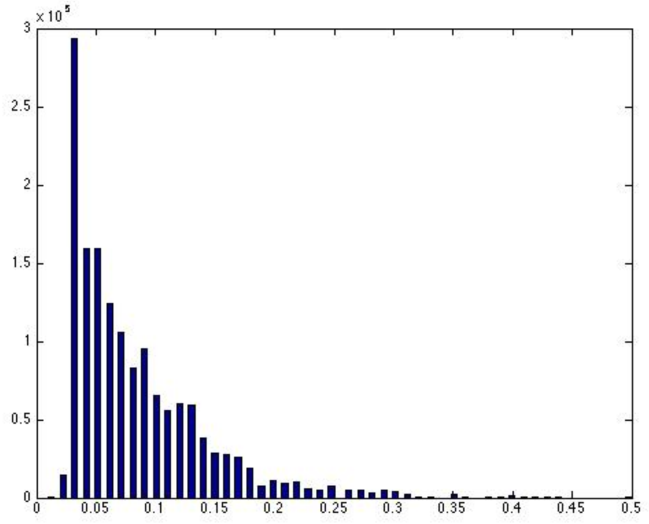

Figure 2: Histogram of rounded durations over all speakers

Figure 1 shows a histogram of the durations for every speaker. It clearly shows that the durations are skewed. To make analysis easier, I also looked at the rounded durations so the durations would fall nicely into the buckets. Because the data is so skewed, I decided to use the mean duration. However, instead of taking the mean duration over every speaker, I just used the mean duration for each individual speaker. I only kept tokens that were within some margin of the speaker's mean. After playing around with numbers, I kept the phones that were .03 s away from the mean duration.

Final sorting

For each speaker who said 'aa', I saved the conversation ID, the speaker ID, the mean duration, and the start and end times of each token. Lastly, I separated the phones by gender.

Analyzing a speaker:

For each speaker, I calculated the discrete Fourier transform of the waveform and extracted the F0 and spectral tilt values. Before actually doing the discrete Fourier transform, I trimmed the window to eliminate noise at the beginning and end of the phone. To do that, I played around with numbers to find a window that looked like just the waveform. I then calculated the discrete Fourier transform and converted it to decibels.

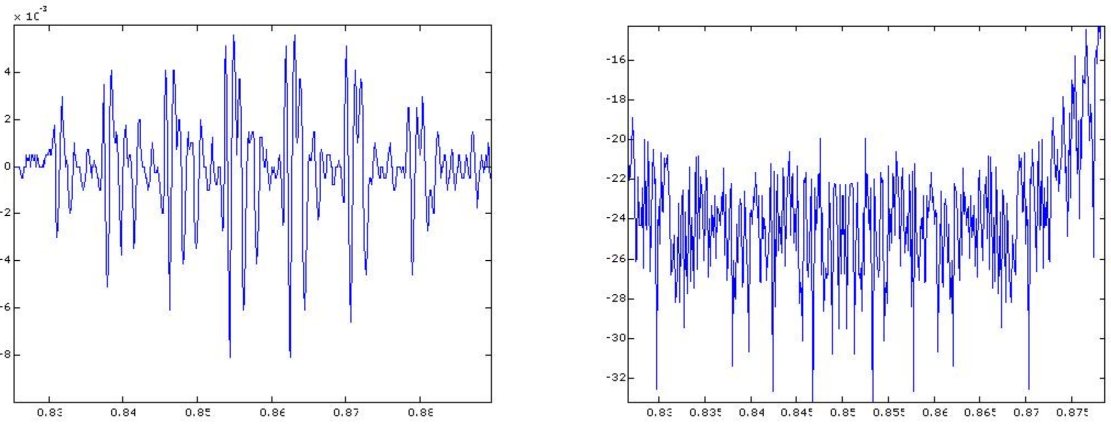

Figure 3: Example of a waveform (left) and a discrete Fourier transform (right) of aa

Extracting F0

I used Matlab' findpeaks function on the discrete Fourier transform so I could find the distance between the first two peaks. However, when I first did this, there was a lot of noise, so fake peaks appeared in my list.

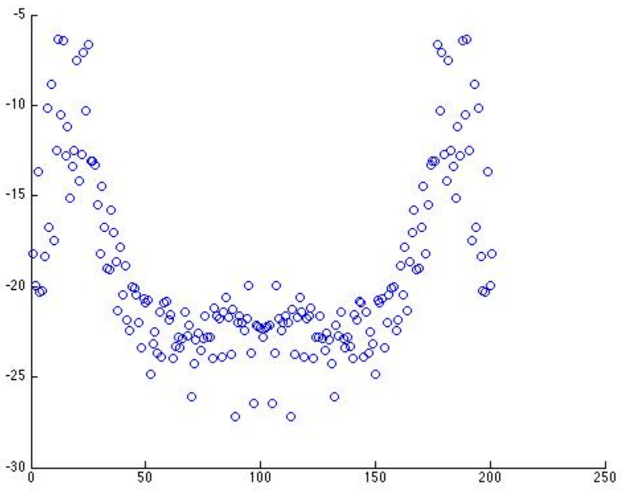

Figure 4: Peaks found using Matlab's findpeaks function

To eliminate the noise, I took the zscore of the amplitudes of the peaks.

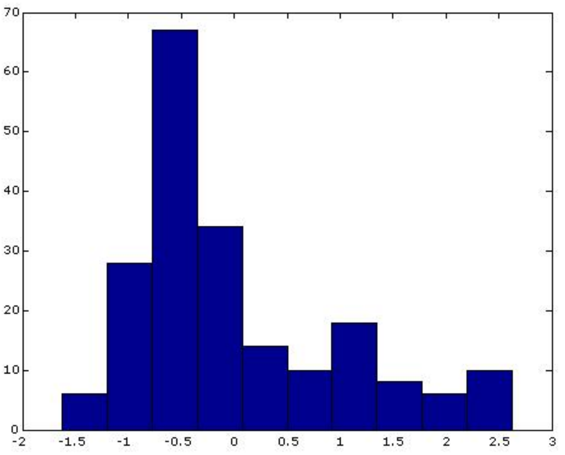

Figure 5: Histogram of zscores for one token

I only kept the peaks whose zscore was greater than 1. Then, I simply calculated F0 by getting the distance between the first two peaks. As a sanity check, I only kept tokens whose F0 value was between 75 – 500 Hz.

Extracting spectral tilt

I used the first two peaks that I found for F0 and calculated the difference between their amplitudes.

Creating the Support Vector Machine:

At first I selected two male speakers to make a classifier for and did their analysis. First I made a scatter plot of the tokens to see if there was any noticeable separator between the speakers.

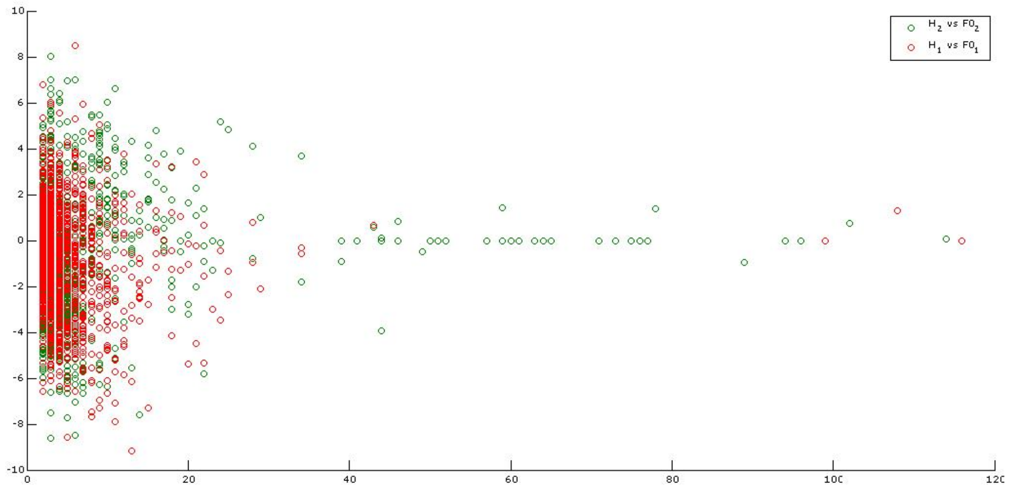

Figure 6: Scatter plot of two male speakers spectral tilts vs. F0 values

As you can see, there is no visible separator in 2D, so I decided I definitely needed the SVM.

Selecting parameters for Matlab's svm function

First I randomly selected tokens from each speaker to make the training set, ensuring that each speaker still had at least half remaining to be part of the test set. Matlab allows you to simply pass "svmtrain" the feature vectors you are training it with and which group they belong to. When I first ran it, I did it this way. However, this was not very accurate. A good classifier should classify nearly every element of the training set correctly. The default only classified about 65% correctly, making it a weak classifier. So I began to play around with the parameters. I decided to modify the kernel function, as different kernel spaces allow the data to be separated more easily. I also decided to modify the method. Below is a table of the accuracy of the classifier on the training set.

		Method		
		SMO (default)	LS	QP
Kernel Function	Linear (default)	0.6541	0.6407	0.6545
	Quadratic	0.6268	0.6368	0.6601
	Polynomial	*No convergence*	0.6794	0.6662
	Rbf	0.6719	0.6920	0.6793
	Mlp	0.4895	0.6243	*No convergence*

The values in the table are not always going to be the same if you rerun the program. That is because I randomly sampled to get the training set. After playing with the parameters, I decided to use the Gaussian Radial Basis Function (Rbf) as my kernel function and Least Squares (LS) as the method.

Gaussian Radial Basis Function (Rbf) maps the data to a kernel space using its distance from the origin or a center.

Least Squares (LS) is a form of regression analysis.

Determining training set size

Once I had my parameters, I played around with the training set size. Having too few elements in the training set leads to a classifier that misses tokens. Having too many elements leads to a classifier that is trained to notice noise as well, so it incorrectly marks noise as tokens.

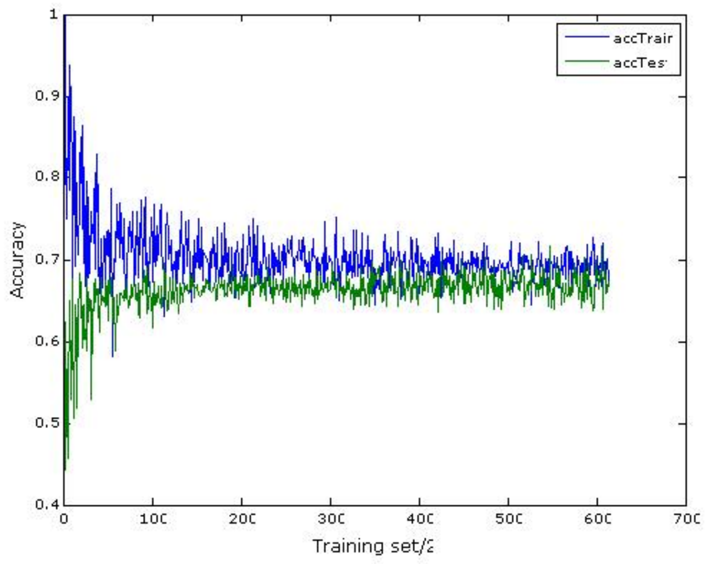

Figure 7: Number of elements in training set divided by 2 vs. accuracy of SVM

After creating classifiers using varying training set sizes, I decided that selecting 400 tokens from each speaker, making a training set of 800, would be large enough. After that point, the accuracy does not seem to change much. However, if I had done this with more tokens, we may have seen the curve for the test set eventually decline.

Results:

Accuracy of SVM classifier on training set

	Accuracy on Training Set	Accuracy of Test Set
Comparing two males	0.7013	0.6679
Comparing two females	0.7397	0.7715
Comparing a male and a female	0.7231	0.7078

Overall

It is interesting that it separated the females better. An accuracy of about 70% on the training set is decent, but not good enough to be called a strong classifier (100% on the training set). There are many parameters to the SVM and the data set that could be tweaked. The SVM classifier is able to identify a large portion of the test set correctly, which is good. The classifier may be unable to classify every feature vector of the test set because some are actually outliers that should have been eliminated.

For the future:

- Using the same phone, create SVM classifiers for other speakers in the corpus
- Identify other traits of waveforms that could be added to the feature vector to make the SVM more accurate. Examples are:
    - The context of the phone - is there a phase boundary before or after
- Adjust with other parameters of the SVM function
- Do this same program using other phones or speakers
- Identify key tokens of a speaker and weigh them more heavily in the training set
- Find a more precise method of trimming a waveform to eliminate noise on the sides