Implementing Speaker Recognition

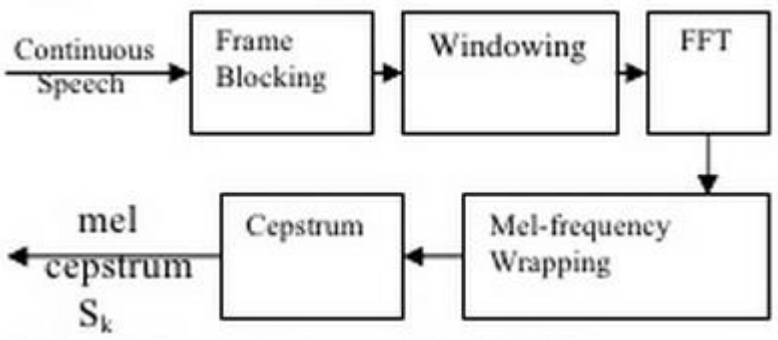
Chase Zhou

Physics 406 - 11 May 2015

Introduction

Machinery has come to replace much of human labor. They are faster, stronger, and more consistent than any human. They've exceeded human beings in most measurable ways. However, some of the most challenging problems facing modern computing is how to allow them to be more like us. For all their calculating power, it is rather difficult for them to perform some basic functions like identifying pictures and voices. Special models and algorithm must be designed for them to do so. For this project, I attempted to train a computer to identify who is the speaker of a sound.

Algorithm

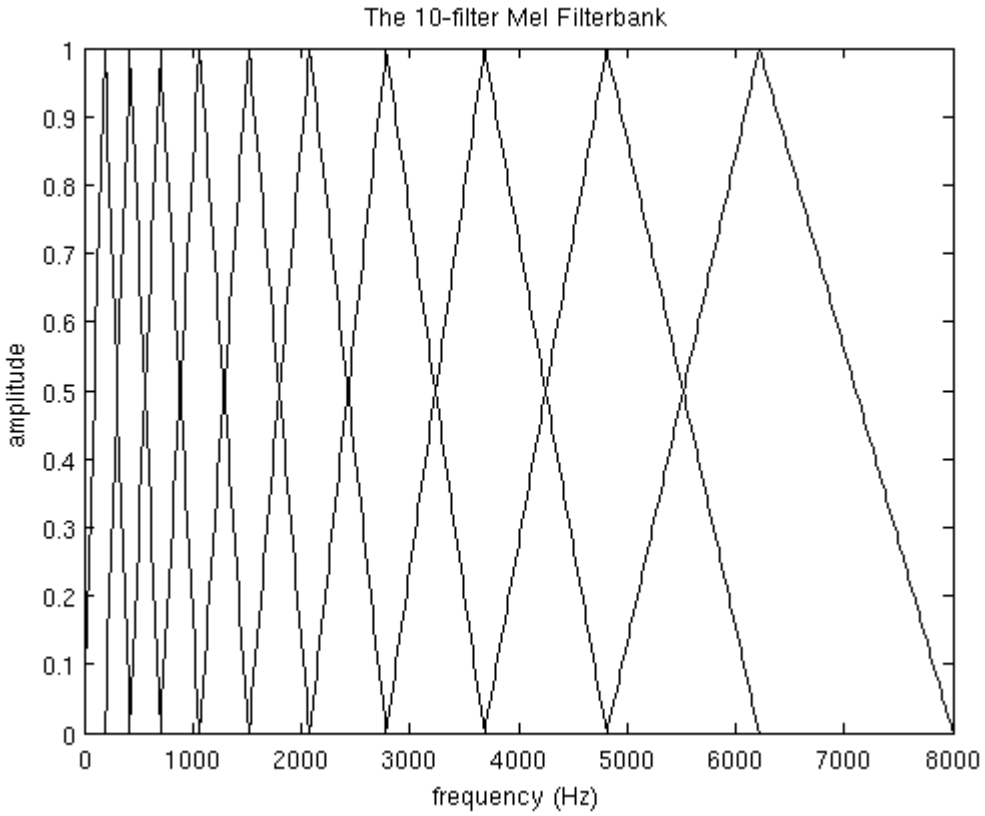


In order for the computer to recognize speech patterns, we must first transform the audio files into something the computer can learn from. The most commonly used and most effective of such transformations is to turn the sound file into a table of things called Mel-frequency cesptrum coefficients. These coefficients represent the power spectrum of sound on the Mel-scale. It will be the computer's task to figure out which speaker is which from these numbers.

The first step in this transformation is to divide the sound file into short frames of around 20 ms in length. This allows us to split each ~2 second sound file into ~100 individual samples. By dividing the signal into such short frames, each section is a relatively constant signal that does not change much. We then pass each frame through a windowing function to resolve the

discontinuity in the beginning and end of each frame. Many different functions can be used, but the most common and the one I used is the hamming window.

For each of these frames, we must find their cepstrum coefficients. Ordinary WAV files store sound by measuring the amplitude of the signal at a certain sampling rate. By taking the Fourier transform of this signal, we can obtain the frequency domain of the sound wave. We then pass these frequencies through a filter bank.



The Mel scale filter bank is composed of triangular band-pass filters of equal width in the Mel-Scale. The Mel-scale was developed in 1937 as a way to measure frequencies based on their perceived pitches from people. Humans actually do not perceive pitch as a linear function to frequency. Rather, it is logarithmic. The most commonly used conversion from frequency to mels is shown in figure below. Each filter represents a mel-frequency coefficient. The magnitude of the resulting signal through each filter is the value of that coefficient. The result is then an n-dimmensional vector where n is the number of mel-frequency coefficients we choose to look for.

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

After processing each frame, we will have an array of such n-dimmensional vectors. Now, the machine must learn to differentiate the speakers based on these arrays.

At this point, many machine learning techniques can be utilized to distinguish learners based on their tables of MFCCs. The one I chose to use is called vector quantization. Based on my research, it seemed to be the most effective and easiest to implement of all learning algorithms. The idea behind it is to treat each n-dimmensional vector from each frame as a point in some n-dimmensional space. We will then arrange these points into k clusters for some number k of our choosing.

I used the Linde, Buzo, Gray (LBG) algorithm to determine each cluster center. For each speaker, take the array of MFFCs. Find the center of all these points by taking the mean of all point. This point will be the first cluster-center. We then split this cluster center into two new centers. Let X be the vector representing the first cluster center. We define X_1 = X(1-e), X_2 = X(1+e) for some small e of our choosing. We then go through all the vectors again and assign each to the cluster center closest to it. Now each vector in the array is assigned to one of these two cluster centers. For each cluster center, we recalculate its position by finding the mean of each vector assigned to it. These new cluster centers are then split again into four cluster centers. This process of splitting and recalculating means is repeated until the specified number of cluster centers is found. The result is a collection of cluster centers called a “codebook”. This codebook will represent the way a speaker “sounds” and is ultimately the tool to classify which speaker is assigned to a new speech file.

After generating a codebook for each speaker, it is very easy to classify new sounds. We must first generate the MFCCs for the new sound file the same way as we generated them for training. It is important to use the same windowing function, frame length, and cepstrum coefficients in order to keep the new MFCCs compatible with the ones from the training data. For each MFCC vector of the new sound file, we calculate the distance of it to the nearest cluster center in each codebook. We then sum each of these distances for all the vectors for the

new sound source. The codebook with the smallest cumulative distance is the speaker we choose.

Process

A lot of time spent on this project was done doing research. It took quite a while to read through articles, trying to make sense of the process of speaker recognition. After puzzling together the overall process, I attempted to create a matlab program that would generate the MFCCs from wav files. However, I quickly realized that attempting to do so would take too much time and was quite risky as well. The process of generating MFCCs takes a lot of manipulation of the WAV file information. Additionally, there would be no way of knowing if my program works since the output would essentially be a random looking sequence of numbers. Ultimately, I found a matlab file online that would output MFCCs of a WAV file and decided to use that. Similarly, while researching vector quantization, I found code that would generate codebooks using the method I described above. I chose to user their code instead of writing it from scratch. With both the codebook maker and the MFCC generator, I wrote a program that took in two WAV files and generated two codebooks for them and another function that tested these codebooks with a test audio file.

I found sample files from http://minhdo.ece.illinois.edu/teaching/speaker_recognition/ which contained clean audio for training and testing. I trained and tested several pairs of such speakers and the program was able to successfully predict the speakers of all instances. However, these sound files were extremely clean with little to no background noise. I wanted to test the algorithm on more realistic audio that one might expect for everyday use. For this, I recorded three different people's voice. I had each of them say some phrase for around 1-2 seconds twice.

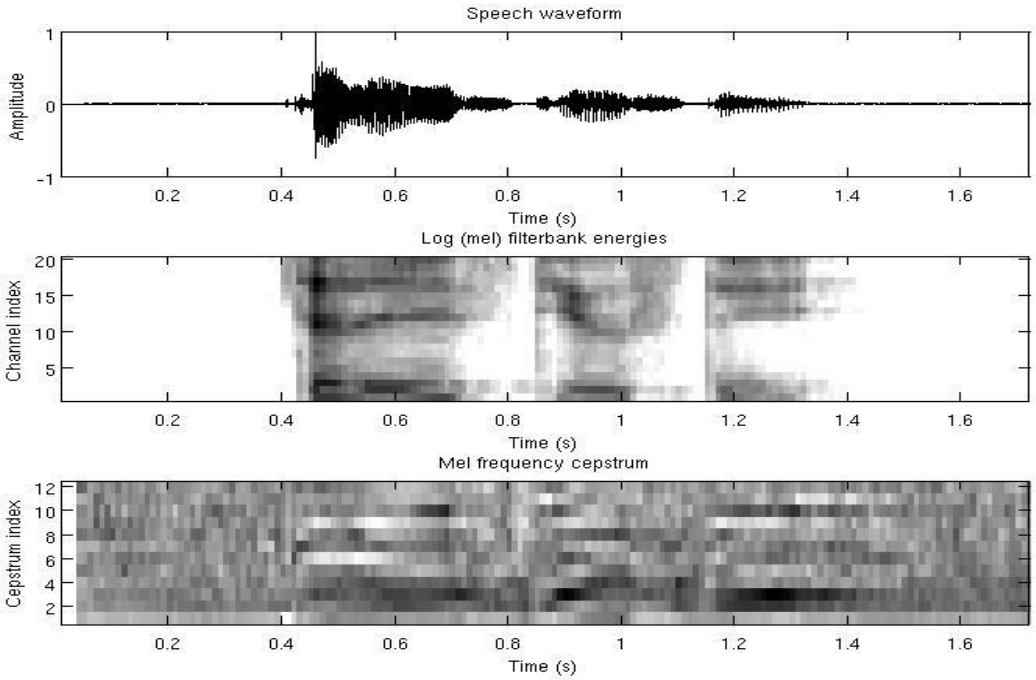


Figure 1 - Bill's Cesptrsum

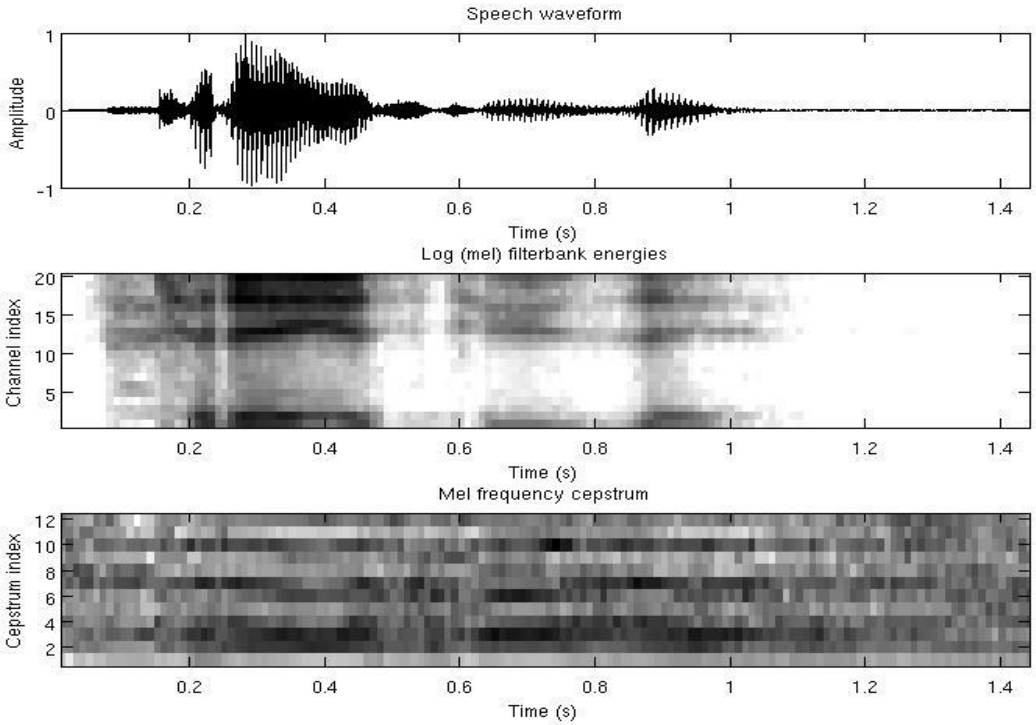


Figure 2 - Duncan's Cepstrum

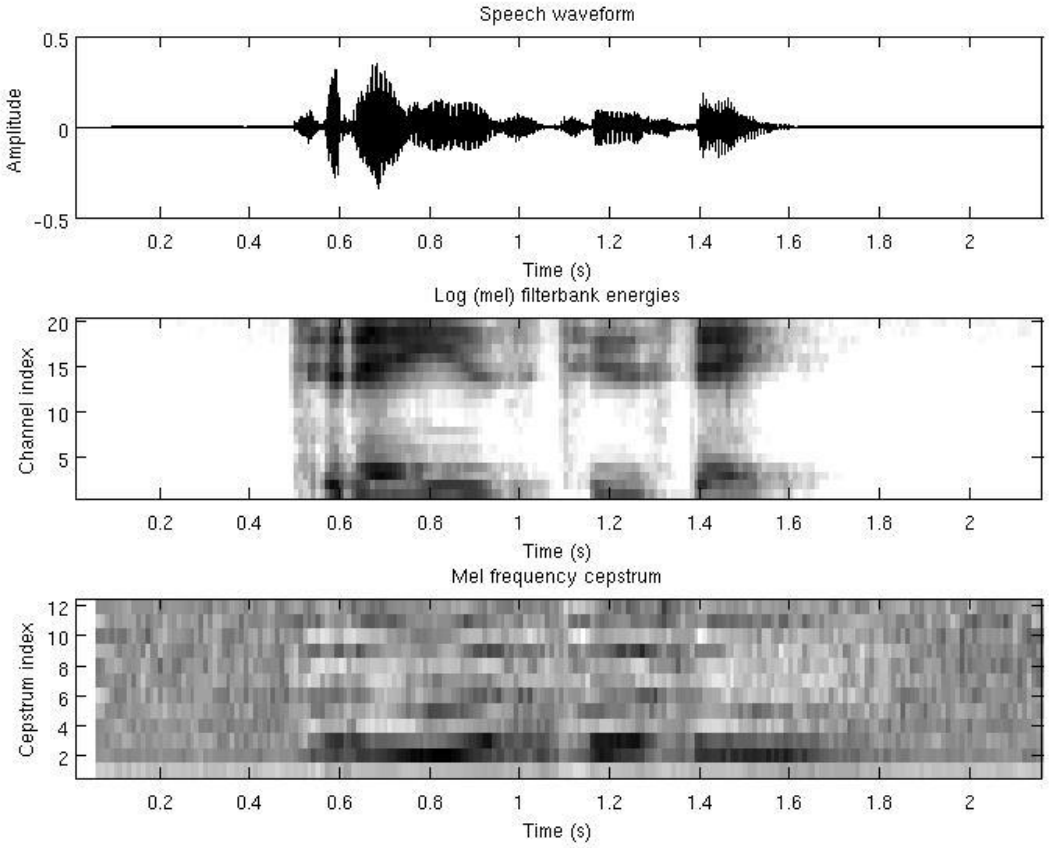


Figure 3 - Emily's Cepstrum

Here are the results of the testing. The horizontal bar represents which sound file I tested it on. The number in each result box is the difference of vector distortion of the correct and incorrect speaker normalized to the length of the test file. This number represents a quantified “sureness” of the classifier. The larger the number, the larger the difference and so the more certain we are that the classifier was correct. The first table is from the clean audio from the website. The second table is the audio I recorded.

train\test	S1Test	S2Test	S3Test	S4Test
S1/S2	Correct 697.4417	Correct 667.6604	X	X
S1/S3	Correct 466.6963	X	Correct 122.9370	X
S1/S4	Correct 204.5936	X	X	Correct 258.7675
S2/S3	X	Correct 743.2253	Correct 336.1526	X
S2/S4	X	Correct 640.6071	X	Correct 546.5009
S3/S4	X	X	Correct 221.2987	Correct 446.1940

train\test	Emily 1	Emily 2	Duncan1	Duncan2	Bill 1	Bill 2
Emily 1, Bill1	X	Incorrect 20.3377	X	X	X	Correct 145.5533
Emily 2, Bill 2	Correct 97.2494	X	X	X	Correct 127.5489	X
Emily 1, Duncan 1	X	Correct 55.2156	X	Correct 67.1011	X	X
Emily 2, Duncan 2	Correct 213.3416	X	Correct 237.7819	X	X	X
Bill 1, Duncan 1	X	X	X	Incorrect 127.2159	X	Correct 251.8071
Bill2, Duncan 2	X	X	Correct 126.4453	X	Incorrect 58.3555	X

The clean audio was able to achieve a 100% success rate while my recorded audio achieved a 75% success rate. This discrepancy can most likely be attributed to background noise and the fact that a good portion of the sound file was silence. Modifying the sound files to contain only the voices should greatly increase the model's accuracy.

To train and test these or your own sound files, download the files and run this in MATLAB:

TrainAndTest('Speaker1Train'wav,'Speaker2Train.wav','Test.wav');

Where the arguments are the names of audio files you want to use for training and testing.

This will output which speaker the computer believes is the speaker of the input wav file.

FILES:
https://drive.google.com/folderview?id=0BwxRkJZ9bJhyfmpQRGVsS2pUclpGYlVKU3NGdFFid254N2N1bFRCOUlLbS05TkFiSWpFZEU&usp=sharing

Future

There are many ways to further test and build on this program. One of the original plans was to test instrument recognition instead of speaker recognition. However, people were more accessible to me than the large variety of instruments needed for such a project. Theoretically, the same process could be applied to musical instruments. By training the program with two instruments playing the same note, it should be able to recognize which instrument is being played if it's playing the same note. I also want to extend the code to test for more than two speakers. It should be pretty easy to implement. I just need to patch the code to handle training and testing a variable number of speakers. Finally, I want to get the accuracy of my classifier higher. I would like to find a way to clean up the audio in some kind of pre-processing before handing it off to the trainer/tester. I would also like to train the classifier on more audio files as that should make the codebooks more representative of the speaker.

Bibliography

MFCC generator: http://www.mathworks.com/matlabcentral/fileexchange/32849-htk-mfcc-matlab/content/mfcc/mfcc.m

Vector quantization: http://www.mathworks.com/matlabcentral/fileexchange/10943-vector-quantization-k-means/content/qsplit.m

"Mel Frequency Cepstral Coefficient (MFCC) Tutorial." *Practical Cryptography*. N.p., n.d. Web. 15 May 2015. <http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>.

Martinez, J.; Perez, H.; Escamilla, E.; Suzuki, M.M., "Speaker recognition using Mel frequency Cepstral Coefficients (MFCC) and Vector quantization (VQ) techniques," *Electrical Communications and Computers (CONIELECOMP), 2012 22nd International Conference on* , vol., no., pp.248,251, 27-29 Feb. 2012

Hasan, Rashidul, Mustafa Jamil, and Golam Rabbani. *Proceedings of ICECE 2004: Venue: Pan Pacific Sonargaon Hotel, Dhaka, Bangladesh, Date: December 28 - 30, 2004*. Dhaka, Bangladesh: n.p., 2004. *SPEAKER IDENTIFICATION USING MEL FREQUENCY CEPSTRAL COEFFICIENTS*. Web. <http://www.buet.ac.bd/icece/pub2004/P141.pdf>.

Do, Minh. "Digital Signal Processing Mini-Project:." *DSP Mini-Project: Speaker Recognition*. N.p., n.d. Web. 15 May 2015. <http://minhdo.ece.illinois.edu/teaching/speaker_recognition/>.

Soong, F.; Rosenberg, A.; Rabiner, L.; Juang, B.H., "A vector quantization approach to speaker recognition," *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '85.* , vol.10, no., pp.387,390, Apr 1985

"Mel Scale." *Wikipedia*. Wikimedia Foundation, n.d. Web. 15 May 2015. <http://en.wikipedia.org/wiki/Mel_scale>.