Louis Jacobowitz
Dr. Robert Rodman
30 May 2016

Algorithms for Comparing Modulated Speech to Unaltered Speech

Sometimes, criminals disguise their voices when committing crimes, such as when making bomb threats. Most of the time, these threats are recorded and then either they are debunked or the bomb is defused, but either way the police then search for suspects. Forensic analysis of these suspects involves comparing their voices to the voices recorded in the bomb threats. However, modulated voices are difficult to recognize, so it was hypothesized that a computer program would be able to make a better comparison. I attempted to write such a program by directly comparing changes in frequency between voice samples.

My approach for doing so was based on Fourier analysis, and my code was written entirely in python. In order for this to work, other modules had to be written. First, there was a method that performed a fast Fourier transform[1] on a given sample of sound. This method used the `numpy` module's `fft.fft` function, and was called `performFFT`. It then compiled the results of the transform into a 2D python list such that each row represented a frequency and each column represented that frequency's average amplitude across the sample.

However, a fast Fourier transform has certain limitations, that being that it becomes less and less accurate as the samples on which it is performed grow longer. At the same time, different people speak at different speeds, so directly comparing two voice samples to each other at the same measure of time is an ineffective means of comparison. My solution was to write another function called `separate`. This function took as an argument a python list constructed from a wave file[2] sampled at 44100 Hz. Its main function was to split a whole wave file into several chunks, each representing a single syllable. These chunks were punctuated by a certain period during which the sample's amplitude stayed below a certain threshold – by default, about 1/20 of a second and 1/10 of the maximum amplitude.

The program I wrote moved through the files side by side, reading for each the next syllable and marking that syllable as "chosen". In order to check that the respective syllables were the same, the program checked that their lengths were within 50% of each other.[3] If they were not, the program compared the current syllable in one file to the next syllable in the next file, and vice versa, and if one of those had a valid comparison[4] then it was assumed that the skipped syllable was an errant sound such as a cough or a breath, and the matching syllables

---

[1] An efficient method for performing a Discrete Fourier Transform: a function that separates a wave, in this case a sound wave, into many different sine and cosine waves of different frequencies and respective amplitudes, such that all the sine and cosine waves added together equal the original wave.

[2] Such that that each element in the array represented the amplitude of the sound at a particular point in that file – hereafter referred to as a "sample"

[3] There do exist more effective methods of checking this, but those methods would involve checking for particular similarities in the phonemes (linguistic sounds) present in each sample, a technology that was too complicated and time-consuming to implement, and which falls under the category of voice recognition rather than sound comparison.

[4] Whichever comparison was closer to an exact match in the length of the syllable

were chosen. The number of skipped syllables and the number of total syllables were recorded. Next, the chosen syllables were directly compared to each other.

In this comparison method, the two 2D arrays produced by the two calls of `performFFT` were effectively placed on top of each other and moved vertically until the values in the overlapping range were the most similar. Then, the average percentage of difference between those values of the two was recorded. Also, the change between samples from right to left was computed and the difference between these numbers of the two syllables was also recorded. These two numbers were averaged to make a total percentage of similarity. This process was repeated for all successive syllables in each file, until the program reached the end of one of the files. Finally, the average percentage of difference of all the syllable comparisons was computed and displayed. Additionally, the percentage of syllables that were skipped by the program was also computed and displayed. These two numbers displayed, respectively, attempted to quantify the similarity in the voices of the perpetrator and suspect, and the speech patterns of the perpetrator and suspect. They were produced for every possible combination between perpetrator and suspect, and the most similar suspects to each perpetrator were thus found.

The results' correctness was not perfect: it was rare to find a particularly conclusive match in both categories, and for most perpetrators there were several suspects with extremely similar numbers to each other. Total similarity percentages on test subjects tended to hover between 85% and 90%, with the largest being 92.7%[5]. Total skipped chunk percentages had a wider variance, from around 37% to 47%, with the lowest being 35.2% – though it was observed that the percentages of skipped chunks were much higher in the comparison of the full perpetrator sound files to the full suspect sound files than in that of the first thirty seconds of each. The most similar suspect selected by the program among our test files had a better than average correctness, but not a perfect record.

I believe that the comparison algorithm used was the best way to perform this comparison. However, there are distinctly better options for improvement. Firstly, the separation and matching of syllables was likely a huge source of error – if voice recognition functionality was added to reinforce comparison of the correct corresponding syllables, the algorithm would probably be much more effective. Secondly, my program, for the sake of manageable data sizes, split the results of the Fast Fourier Transform into groups, each representing a certain quarter-octave of frequencies. If there were smaller divisions between frequencies, like eighths- or twelfths-of-octaves, accuracy might be improved. Finally, besides the algorithms themselves, another source of error was the need to transpose the suspects' audio to a higher frequency in order for it to best be compared with the modulated voice samples. If forensic investigators were able to identify the precise method of voice modulation used by the perpetrator, they would be able to make a more accurate comparison after applying that modulation to the suspects.

As a postscript, during the course of my research I found a certain program made for this exact purpose called EasyVoiceBiometrics (http://www.easyvoicebiometrics.com/index.php) which claims great success at comparing voices in a forensic context. I attempted to emulate its methods in my own process, though I was unable to test the program for myself[6].

---

[5] Between, in our sample of test files, perpetrator #5 and suspect #7.

[6] As I have the OS X operating system and EasyVoiceBiometrics is a Windows-only program, I attempted to test it using NCSU's virtual computer lab; however, when I tried this the program would not start successfully. I was unable to get it to start without a fatal error.