Speech Recognition

Project Kojak - Weeks 9-12 of the Metis Data Science Bootcamp

Steven Bierer December 12, 2018

Introduction

For the fifth and final project, coined "The Passion Project", I decided to delve into a problem for which I have a modest amount of domain knowledge: the processing of human speech. I've studied in the past how neural circuits in the human auditory system encode spectral and temporal cues contained in speech and other complex sounds, particularly how such cues can be optimally conveyed by cochlear implants (a medical device that provides hearing for people with profound deafness). For this project, I explored another aspect of hearing research, artificial speech recognition.

In the past several years there has been an explosion of growth in interactive speech systems. So-called "smart speakers", like the Google Home and Amazon Echo, are two prominent examples. It's been estimated that there will be over 100 million of these devices installed in homes and businesses in the United States by the year 2020. While the utility of speech systems lies in their ability to interpret vocalized commands, there is a growing need to also identify the person issuing the commands. That's known in the field as "speaker recognition". Applications of speech recognition range from documenting who is speaking in real time – such as for making a record of a meeting – to improving the interpretation of speech, which can be degraded by unusual accents or intonations. Another is voice authentication. Many systems have to limit who can use its services, and speaker recognition is one way to achieve such security.

Approach

My approach for this project was different than in others, in which I trained and tested a tried-and-true classifier or regression model. This one was more along the lines of a Monte Carlo simulation. I wanted to simulate many household devices, each of which has just a handful of regular users that employ its interactive speech capabilities. The idea was that I would create a classifier (in this case, based on nonnegative matrix factorization (nmf)), trained with labeled data, and see how well I can identify who is speaking among members of a group. The training data (i.e. the amount of audio recordings available for each person) should be small, because in reality smart speaker systems can learn voice characteristics after just a few seconds of speech, not unlike the human auditory system. Also, the classifier should reject speakers not in the training group, that is, not a member of the household.

From the audio recordings of each speaker, I transformed to a spectrogram then extracted the essential spectral features using nmf, from which I created a "dictionary" of that speaker's characteristic vocalizations. For each household group, I created a composite dictionary by concatenating all of the speaker's individual dictionaries. (In numeric terminology, each such composite dictionary has F rows where F is the number of frequency bins, and a number of columns equal to #recordings * #nmf components * #speakers in the group.) Testing of a new audio spectrogram was accomplished by solving for the nmf temporal activations, summing the energy across time of each row, then summing the energy corresponding to each speaker. If the energy of one of the speakers was clearly high enough (compared to a heuristic criterion level, as discussed below), then that in-group speaker was predicted as being the source of the audio recording. If not, no one was accepted as the speaker may have not been a member of the group.

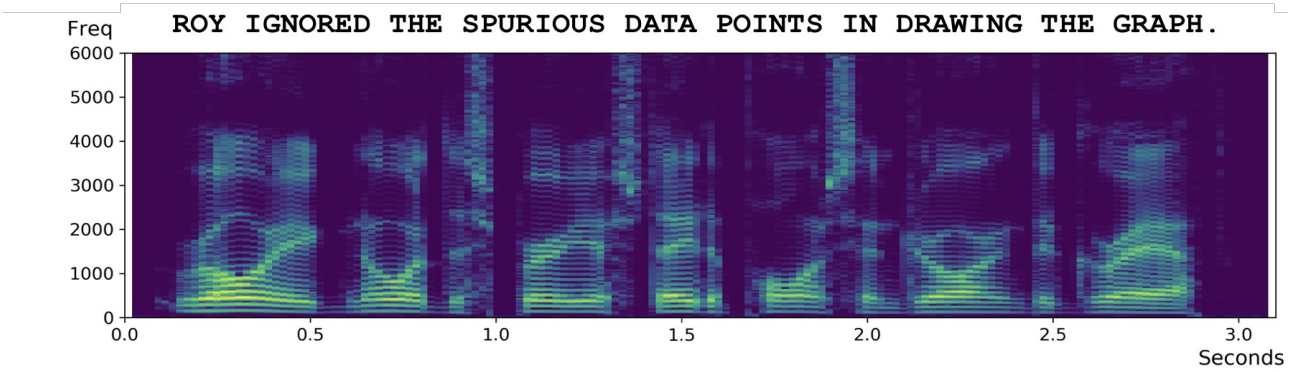
The following tools and sources were used in the analysis:

> Data source: TIMIT (Texas Instruments / MIT) Acoustic-Phonetic Continuous Speech Corpus. (https://github.com/philipperemy/timit)
>
> Tools: 1) Python and the standard libraries *numpy* and *matplotlib* for routine data manipulation and graphical display; the regular expression module, *re*, for finding string patterns; and *glob* for finding files in specified directories. 2) Python modules *pyaudio*, *librosa*, *scipy*, and *sklearn*. Oddly enough, for the first time I did NOT use the ubiquitous *pandas* module.

Concepts learned and applied: Nonnegative matrix factorization; spectrograms; Monte Carlo simulations. I also made some *matplotlib* graphs outside of notebooks using the QT module, and explored the animation module of *matplotlib* which is a convenient way of making time-moving plots.

Extracting Spectral Features

The essential step for the speaker recognition model is nonnegative matrix factorization. This step is performed not on the audio signals directly, which are functions of time, but on their spectrograms, which are functions of frequency and time. After reading in an audio file, a spectrogram was made using the *scipy.signal* function by the same name that performs a series of short-term fourier transforms on the audio waveform. One such example for a male speaker is shown below.



Consider the spectrogram an F x T matrix, X, where F is the number of frequency bins and T the number of time bins. Nonnegative matrix factorization uses a constrained iterative method to decompose X into a matrix of spectral components, D (of size F x K) and temporal components, V (of size K x T). That is,

$$X = DV$$

I called the columns of D the "dictionary" and the rows of V their corresponding "activations". To create a set of components that define a speaker's voice, many such D matrixes were created from all of the audio files. Then each speaker, j, is represented by an extended dictionary, D_j.

Creating Groups of Speakers

The audio files were stored in cascading subdirectories indicting the U.S. region the speaker was from and the gender and initial of the speaker's name which served as a unique ID. I wanted the 100 household groups, each with 6 people, to have a variety of speaker diversities, based on gender and dialect. These ranged from some groups consisting of all male or all female speakers from a single dialectal region (not very diverse), to an even mixture of males and females from 6 different regions (very diverse). To make these groups, I created a set of rules in advanced and created a special subroutine to search the file subdirectories and create unique sets of speakers matching those rules.

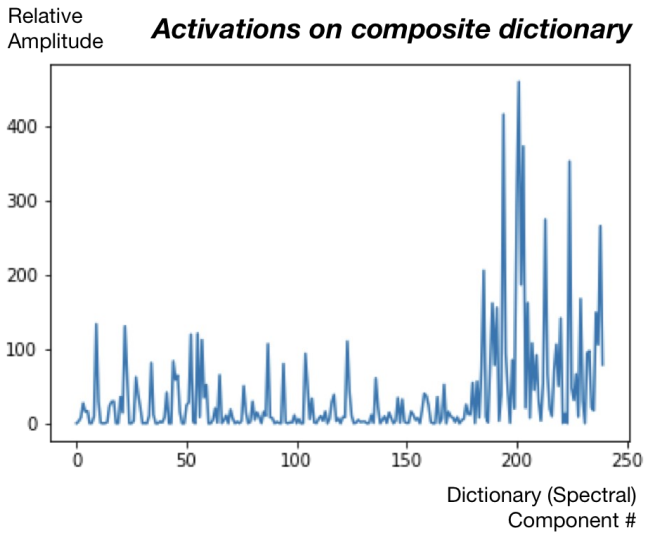
Each speaker subdirectory had10 audio files, each with a different sentence utterance. 7 of these were used for training (i.e. creating a speaker dictionary) and the remaining 3 for testing. I also created a second set of rules to gather speakers unknown to all groups. No dictionaries were created for this group, as they served as the "out-of-group" test speakers.

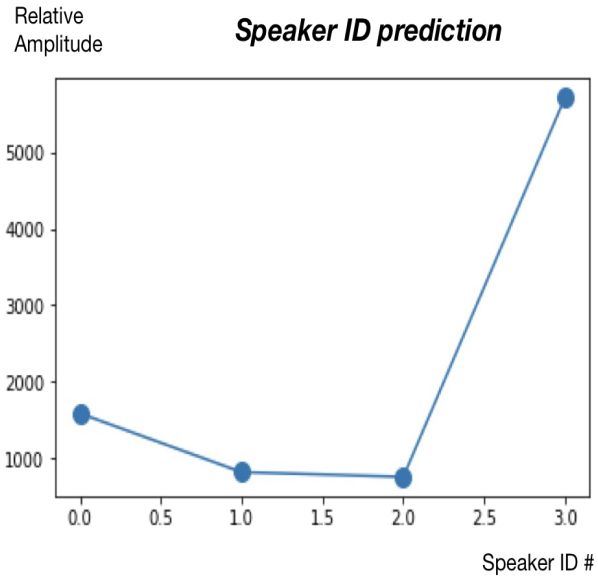
Model training

For each group of the training data set, a dictionary was created for every speaker. A composite dictionary, D_g, was made by concatenating the individual dictionaries, D_j, for all 6 speakers in a group. That is,

$$D_g = [D_1 \ D_2 \ \dots D_6]$$

For making speaker predictions using the test audio files, the nmf fitted model for each group was forced to be D_g, by manipulation of the *sklearn* Python module's attributes. Then, the output activations, V_g, contained the temporal sequence with which each dictionary element was "active" during the test utterance. Summing these over time yielded an energy signal V_{energy}. An example of an energy signal is shown below for a training audio file. Note that the x-axis has 240 features in this case, corresponding to 6 speakers x 6 components x 8 training files. (In the final model, however, 10 nmf components were used and only 7 training files.) The recording was made for the last speaker (id=3), so as expected most of the energy is on the right. Summing up the features based on speaker id leads to the next plot, which shows a much higher amplitude than the other three speakers (ids 0, 1, 2).

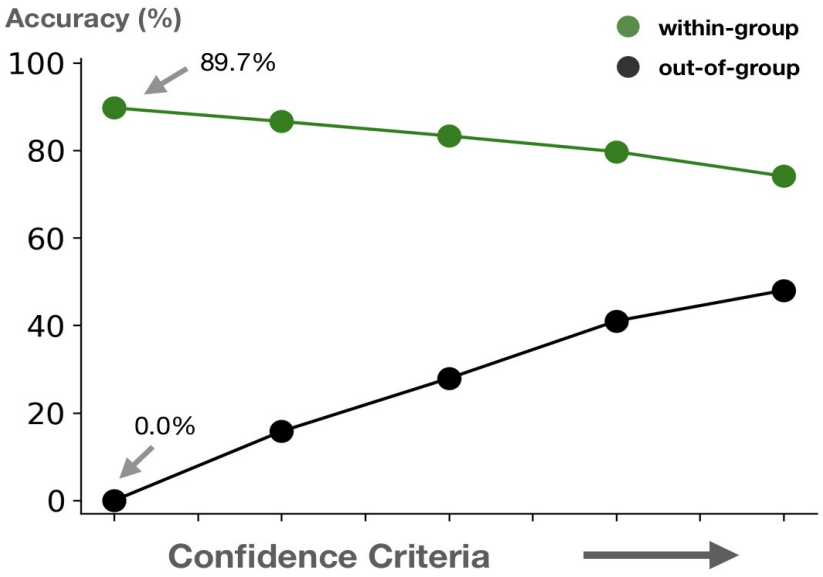




Hyperparameters such as regularization scalars, the number of nmf components, and the limitation of frequency range were assessed over a subpopulation of speaker groups using the test data. From this procedure, I found that there was some benefit of imposing a special sparsity constraint such that at any given time point, for the matrix V_g, it was preferred that only dictionaries from one speaker are active. No existing common Python module exists for this elaboration of the nmf routine, so I had to write my own iteration update algorithm.

Prediction results

The simple energy summation of the test audio sparsity-nmf activations proved to be a very robust way of predicting a speaker from among a small group of known users. An extra layer of classification, such as Naive Bayes or Support Vector Machine, was not necessary. If speakers outside of the group weren't a consideration, the system performed with 89.7% accuracy, with errors generally occurring between group members of the same gender. This value is shown with a green arrow in the figure below. Of course, in that case an out-of-group speaker would never be correctly identified as such (black arrow). As a criterion level was adjust to correct for unknown speakers, the in-group accuracy declined to about 76% while the out-of-group accuracy climbed substantially to 41%.



Conclusions

Designing a simple speaker recognition system, one that could work on limited amount of training data and make its judgment for an incoming audio sigmal in just a few seconds, was overall successful. The system had an added benefit that for a modest cost in identifying someone from a 6-member group of known users, it could determine if a speaker was not a member of that group. Surpisingly, group diversity didn't impact accuracy very much, either for within-group or out-of-group speakers.

Given time, I would have liked to test the system with noisier audio files, which is closer to real-world environments. Of course, most modern smart speaker systems have multiple microphones that helps eliminate background noise via beam-forming. I also started working on a Python application that takes in audio input via a computer's microphone and can train and test in near real-time.