PREDICTING AN EVENT FROM AUDIO DATA:

The goal of my project was to develop an algorithm to predict an event from audio data.

Specifically: within some margin of error, I'd like to be able to predict if a door has closed based on the sound a door makes when it closes.

PROJECT STEPS:

1) Locate data set
2) Read the data into Python
3) ‘Clean’ the data with FFT and/or other audio analysis tools
4) Perform unsupervised analysis to try to define what a cluster of similarly “shaped” data looks like with respect to frequency, time, and potentially amplitude or other variables
5) Perform supervised, categorical analysis to divide the data into a series of groups based on where in the data set the previously defined ‘shape’ fits, and what the margin of error for the fit is.
6) See how well the events actually predict the door closing

STEP 1) LOCATE THE DATA

I've found a 3-hour youtube video of somebody recording themselves repeatedly opening and closing a door:

https://www.youtube.com/watch?v=tULM0KBx7i8

(same guy that did the video of that waffle falling)

STEP 2) READ THE DATA INTO PYTHON (A)

I've decided to use a python based command-line program called youtube-dl to download the audio from the youtube video.
(The github of youttube-dl can be found here: https://github.com/rg3/youtube-dl)

I've decided to use this program for several reasons:

1) File size: the previous strategy I had involved a website that can download a youtube video, however due to the length of this video predicted file size (6 GB) is too much for my computer

2) Efficiency: the terminal will be much more efficient than using a browser to download the data

3) More Control: I have more direct control over the quality of the file that is downloaded

However:

- The audio from the youtube video is only available for download as .webm or .m4a

- In order to read the audio file into scipy I need it as a .wav file

STEP 2) READ THE DATA INTO PYTHON (B)

Good News Everyone!

youtube-dl has a post-processing algorithm that can save the file as just about any file type I need (including .wav!)

However, this functionality requires the instillation of one of two additional programs to function (ffmpeg or libav)

FFmpeg can be found here: https://www.ffmpeg.org/ with a build for just about every operating system you could need

- FFmpeg uses the Libav codec, so if you have FFmpeg installed you also have Libav.

STEP 3) 'CLEAN' THE DATA

Initially I wanted to use a comprehensive open-source C++ audio analysis library called Essentia to analyze the audio data. However, I was unable to get a working install on my computer.

(Essentia can be found here: http://essentia.upf.edu/)

Essentia includes a collection of algorithms designed around extracting all sorts of features from audio files. The library also includes Python bindings, comprehensive documentation, and even a Python tutorial.

PLAN B: SCIPY

Scipy also has a comprehensive library of functions used in audio analysis, as well as really great documentation on how to use the algorithms. However, it is not wrapped as nicely as Essentia is, and algorithms for the sort of cleaning I wanted to do on the audio data had to be written from scratch.

(Scipy can be found here: http://www.scipy.org/, but comes almost entirely pre-packaged in Pandas)

STEP 5) WHAT AM I GONNA DO?

When you read a .wav file into python using scipy.io.wavfile.read('file_name.wav'), it gives you a tuple where the 0th location is an integer value corresponding to the sampling rate (number of numbers per second of audio data in the file), and the 1st location is an numpy array containing a numerical (discrete) representation of the waveform.

In my case, the array consisted of arrays of two integers, each corresponding to one of two audio channels present in the .wav file. After splitting the channels up I ended up with two very long arrays to look at. 3 hours and 22 seconds of audio data at a sample rate of 44100 meant each audio channel was 47,7271,040 integers long.

The solution: start small

From here on out I decided to aim more towards a 'proof of concept' instead of some of my original goals. Consequently, I picked a 15 second slice of audio (from t = 40s to t = 65s) to focus on.

STEP 6) STRATEGIZE

One thing I felt was very important was trying to come up with a way to have the audio data itself predict when an event was happening.

My overall strategy was:

- Generate timestamps for the onset and end of each event of the type I was interested in
- Consolidate the audio data for those timestamps
- Run a short term Fourier transformation (stft) on consolidated audio data to generate a frequency spectrum that is characteristic of the audio event I'm interested in
- Perform an inverse Fourier transformation on the frequency spectrum generated above to get a characteristic shape for the audio of the door closing!

STEP 7) TIMESTAMPS

First I tried to use an auto-correlation to heighten the areas in a given audio channel that were repeated. This proved only marginally successful. The biggest issue was the time it took to compute. It's not efficient enough to handle arrays with millions of data points.

The solution: Modal!

Modal (Musical Onset Database And Library) is amazing. You can get it here:

https://modal.readthedocs.org/en/latest/

https://github.com/johnglover/modal

Modal comes equipped with a number of methods to detect the onset of an audio event. The two I found most useful for my project were the ComplexODF and LPEnergyODF.

THIS PROJECT WAS BROUGHT TO YOU BY

- **THE DRINK: COFFEE**
- **THE SONG: GALAXY OVERDRIVE (BY LOS STRAIGHTJACKETS)**
- **THE LETTER: P?**