

## Design Doc for Police radio transcription project

### **Software**

Required software:

- Streamripper
  - Can be found here: <http://streamripper.sourceforge.net/>

Required Packages:

- requests\_html
- logging
- Re
- Pandas
- Os
- Usaddress
- pyAudioAnalysis
- pydub
- noisereduce
- scipy
- Matplotlib
- Numpy
- Wave
- Sys
- Math
- contextLib
- Soundfile
- Datetime
- Speech\_recognition

### **Work Flow**

Crawler -> scraper -> preprocess -> transcription -> parse

All data scraped from **Broadcastify.com**

### **Breakdown of modules**

Crawler:

- Written by Sungwong Chung
- The crawler is used to find urls for every broadcast on broadcastify.com
- Not finished, needs to be able to set relays for every url, currently only does one
- Inorder to select a specific stream for testing, go into crawler.py, go to the method set\_relay(self), and change the variable feed\_id to the feed you want to test
  - The feed\_id can be found by going to the url of the stream you want to scrape, and copying the number after feed/

- Example: for the stream <https://www.broadcastify.com/listen/feed/2776>, you would copy the 2776 into the feed\_id

#### Scraper:

- Written by Sungwong Chung
- Uses streamripper to rip the stream using console commands
- Currently only scrape\_one function is implemented, need to implement scrape\_all
- Can change the name of the output mp3 file by changing the last part of the string
  - Currently it is set to "test\_audio\_cc"
- Can change how long it is scraping by changing the number before the -a flag
  - Currently is set at "600" for 600 seconds

#### Preprocess:

- Written by Tucker Johnson
- Has a variety of functions for preprocessing, the main ones being:
  - convert\_mp3\_to\_wav(audio\_path)
    - Takes as input a string of a path to an mp3 audio file, will create a wav file with the same name in the same input directory
  - remove\_silence(audio\_path, out\_directory)
    - Segments a wav audio file, removing all periods of silence
    - Takes in a string path to an audio file
    - Out\_directory is a string path to where the segments will be saved to
  - noise\_reduction()
    - Should not be used in final production
    - Was trying to reduce noise using method found here:
      - Jupyter notebook: <https://timsainburg.com/noise-reduction-python.html>
      - Github: <https://github.com/timsainb/noisereduce>
    - Was found to be less effective than frequency filtering
    - Note: if used, outputted wav files need to be converted from pcm-32 to pcm-16, this can be done using the convert\_to\_pcm16(audio\_path) method
  - frequency\_filter(audio\_path, out\_path, frequency)
    - Takes in a file from the audio path, and will write a filtered wav file to the outpath
      - Note, outpath should also include outfile's name
      - Ex: "directory/filtered\_segment.wav"
    - Frequency is the cutoff for the max frequency allowed
    - After a lot of testing, best frequency found to be 400
    - Based on Moving average frequency filter which can be found here: <https://stackoverflow.com/questions/24920346/filtering-a-wav-file-using-python>
  - boost\_audio(audio\_path, boost)

- Boosts the decibels of inputted wav file
- Will replace the wav file found in the audio\_path with a boosted version
- The boost variable is the number of decibels it will boost by
  - Found to work best in the 6-10 range

#### Transcriber:

- Written by Tucker Johnson
- Takes in a speech\_recognition.AudioSegment() class as input
- Uses google speech recognition software
- Returns a string of transcribed text, or "Couldn't Transcribe" if it is at too low of confidence for the transcription
- The built in reduce\_ambient\_noise method of the speech\_recognition package does not work well, would not recommend using it

#### Parse:

- Written by Tucker Johnson
- NEEDS A LOT MORE TESTING, more of a proof of concept parser than a fully functional one
- Called using parser.parse\_lines()
- Parses out addresses using usaddress package found here: <https://github.com/datamade/usaddress>
  - Might also recommend using libpostal if you can get it to work
- If an address is within 3 lines of crime, will save a tuple of (address, crime meaning) in a class variable named street\_crimes
- Can pull street crimes using get\_street\_crimes() method
- Can pull a pandas dataframe of all codes used and the number of appearances for each one in the text file using get\_crime\_counts() method

#### What still needs to be done:

- Paralyzation of scraping, preprocessing, transcribing, and parsing
- TESTING OF PARSER
- Potentially using a different (maybe payed for?) transcription software
  - Current software naturally has 16% error rate, this is compounded by bad quality of police broadcast audio

#### What has been tested:

- Best results found when audio is segmented by noise, filtered with a frequency of 400, then boosted by 10 decibels (I've been using 10)
- Other things tried:
  - Frequencies above 400
  - Boosting at different DB levels
  - Boosting, then segmenting, then filtering, then boosting
  - Filtering, then segmenting, then boosting

- Filtering, boosting, segmenting
- Noise\_reducing, filtering, segmenting, boosting
- Filtering, noise\_reducing, segmenting, boosting
- Segmenting, filtering, noise\_reducing, boosting
- Segmenting, noise\_reducing, filtering, boosting
- Noise reduction only seems to hurt results, especially compared to filtering

**NOTE: Example.py has not been tested, might need to change a couple file paths in order to get it to work. It should be used as a general guideline for the workflow for testing purposes**

If you have any questions, contact me on linkedin:

<https://www.linkedin.com/in/tucker-johnson-398a01141/>

Or by email (although I'm less likely to see it): 15tuckerjo@gmail.com