2. Data Wrangling

2.1. Dataset:

The dataset consists of 8732 sound excerpts that are shorter than 4 seconds each. Sound excerpts are categorized into ten classes that represent common sounds often experienced in an urban environment. The labeled classes available in the dataset are grouped into the following sounds: Air Conditioner; Car Horn; Children Playing; Dog Bark; Drilling; Engine Idling; Gun Shot; Jackhammer; Siren; Street Music.

I downloaded the data files from:
https://drive.google.com/drive/folders/0By0bAi7hOBAFUHVXd1JCN3MwTEU

The training sample dataset is 3.4GB and the test sample dataset is 2.15GB. Data files consist of .wav audio file formats where each file has a unique numeric ID. The class labels are in a separate excel file in the same folder that has two columns containing the file ID and its class annotation. The training data is completely labeled while the test data is not. As a result, I will use part of the training dataset in the testing stage. The test dataset files will not be used in this project.

2.2. Reading in the Data:

After acquiring the data, the next step is to read in the data files and store them in a processing friendly format. The data is .wav audio file format, each data sample has one (mono) or two (stereo) channels. It is important to read in the properties of the audio files for data analysis and cleaning. Audio file properties extracted were sampling rate in Hz, sampling width in bytes, total number of frames and number of audio channels. These properties were then stored in a dataprop.csv for future reference.

2.3. Loading Data Frames:

Python's wave module is the standard for loading .wav files. I utilized this module to extract audio properties in the previous section. A more numpy friendly approach to reading the data would be to use the scipy.io package
[https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.wavfile.read.html]. For example, for 16-bit Pulse Code Modulated audio, reading the file produces a numpy array of int16 datatype in the range from -32768 to 32767. This numpy array can then be made into a pandas dataframe for statistical analysis. A drawback of using the scipy.io package would be its inability to read 24-bit sampled audio data. This is quite a popular format and thus some files in our data have a 24-bit sample width. Due to this issue, the functionality of reading 24-bit data is needed in the implementation. A separate issue is the inability of the wave module to read oversampled audio, in this case files with sample frequency greater than 48kHz. Thus, all oversampled audio will have to be resampled, this will reduce the data volume available for analysis later. We used ffmpeg

[https://ffmpeg.org/] for data resampling. While resampling data, I made sure all output files have the same bit depth (16-bit PCM), this should help in the data cleaning stage.

Classification labels corresponding to training samples are provided in .csv format that could also be read in as a single numpy array. Alternatively, I will use the pandas dataframe structure and methods to handle the label data.

2.4. Data Cleaning:

After reading and loading the labels data into a single dataframe, I needed to set the index and columns appropriately as well as their data types.

The labels dataframe variable will have two columns (ID (dtype: int64); Class (dtype: object)). We'll need to change the dtype of ID column to string or pandas "object" to match the ID column of the audio dictionary that has keys as ID stored as strings and values as int16 numpy array data. Just as we used the ID to index the audio dictionary, we'll set the index of the labels dataframe to ID.

Audio data is stored as numpy array type in a python dictionary. Each element in the array column is the value of the discrete sound signal and multichannel values are in multidimensional arrays. In our case, the maximum number of channels is 2 (stereo), so at most an entry may be a 2D array.

There are no missing or null values in the labels dataframe or the audio dictionary.