crhung / **Voice-Emotion-Detector**

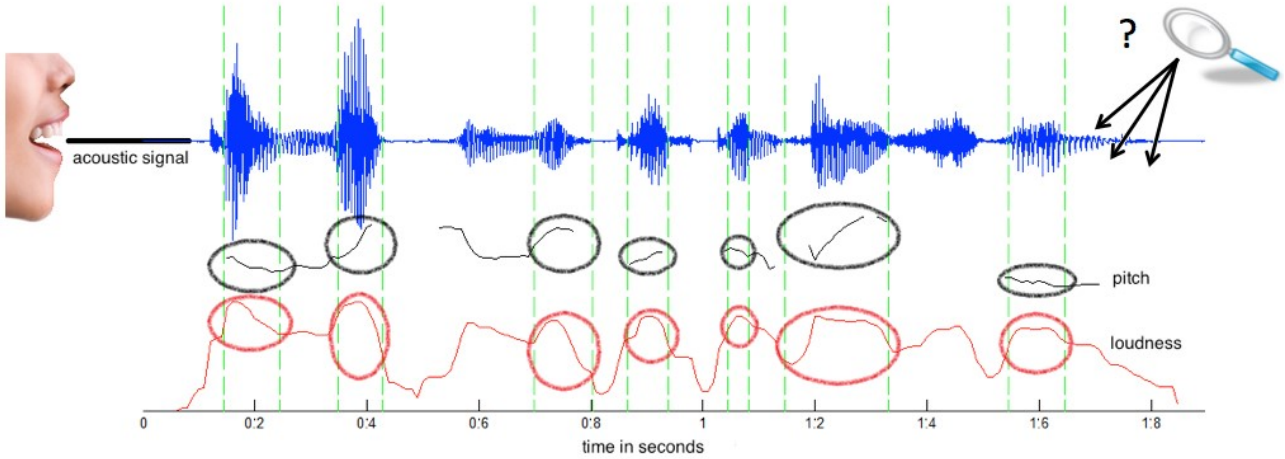
Voice Emotion Detector that detects emotion from audio speech using one dimensional CNNs (convolutional neural networks) using keras and tensorflow on Jupyter Notebook.

README.md

Speech Emotion Analyzer

- The idea behind creating this project was to build a voice emotion detector that could detect emotions from audio from speech
- It can be used in many business applications for example, this can be used by multiple industries to offer different services like marketing comapny suggesting you to buy products based on your emotions, automotive industry can detect the persons emotions and adjust the speed of autonomous cars as required to avoid any collisions etc.

Analyzing audio signals



©joomla_speech_prosody

Datasets:

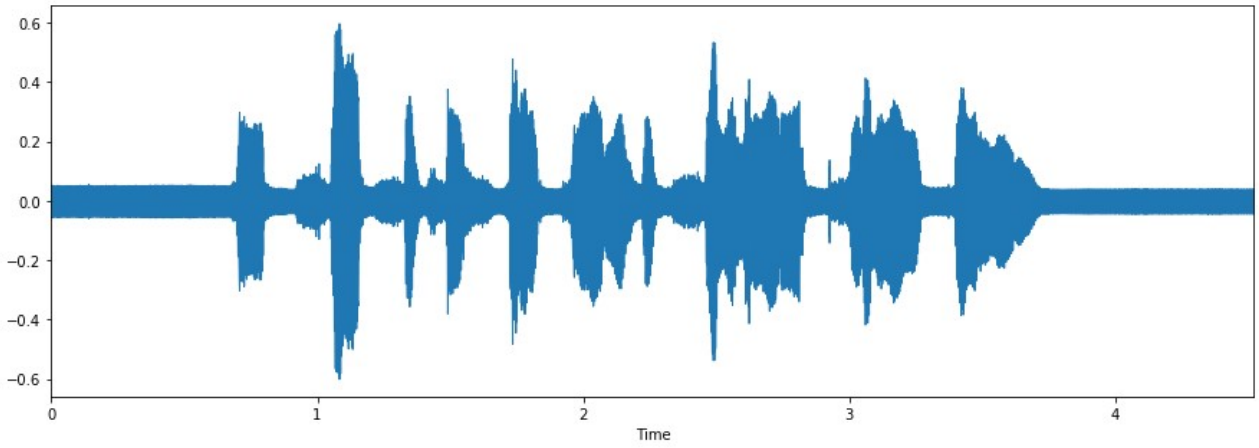
Made use of two different datasets:

1. RAVDESS. This dataset includes around 1500 audio file input from 24 different actors. 12 male and 12 female where these actors record short audios in 8 different emotions i.e 1 = neutral, 2 = calm, 3 = happy, 4 = sad, 5 = angry, 6 = fearful, 7 = disgust, 8 = surprised.
   Each audio file is named in such a way that the 7th character is consistent with the different emotions that they represent.

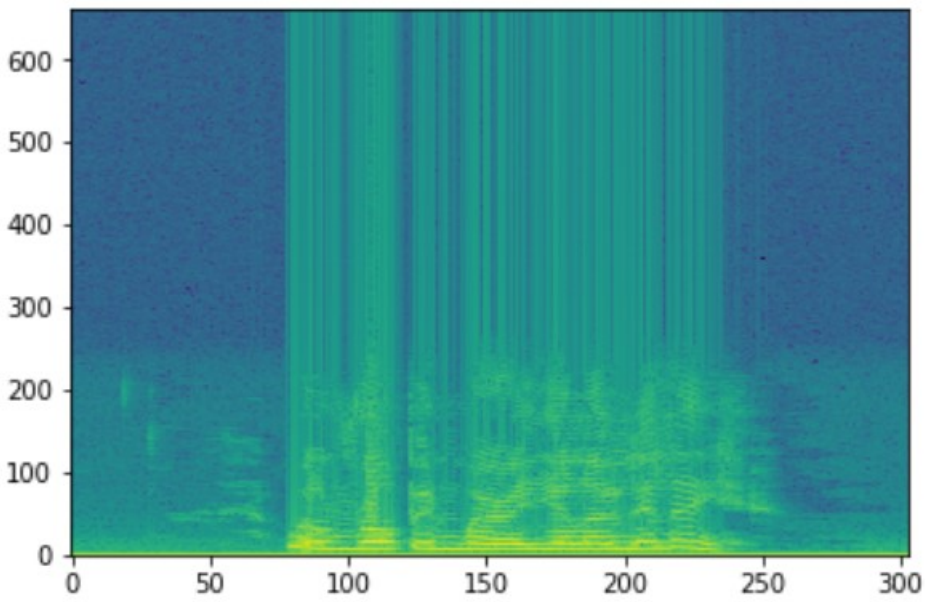
2. SAVEE. This dataset contains around 500 audio files recorded by 4 different male actors. The first two characters of the file name correspond to the different emotions that the potray.

Audio files:

Tested out the audio files by plotting out the waveform and a spectrogram to see the sample audio files.
Waveform



Spectrogram



Feature Extraction

The next step involves extracting the features from the audio files which will help our model learn between these audio files. For feature extraction we make use of the **LibROSA** library in python which is one of the libraries used for audio analysis.

```
In [14]: #Extracting the features from the audio files
         df = pd.DataFrame(columns=['feature'])
         for index,y in enumerate(mylist):
             X, sample_rate = librosa.load('RawData/'+y, res_type='kaiser_fast',duration=3, offset=0.5)
             sample_rate = np.array(sample_rate)
             mfccs = np.mean(librosa.feature.mfcc(y=X,
                                                  n_mfcc=25,),
                             axis=0)
             feature = mfccs
             #[float(i) for i in feature]
             #feature1=feature[:140]
             df.loc[index] = [-(feature/100)]
```

- The feature used to extract from are Mel-frequency cepstrum coefficients (MFCCs). This feature is used often used in voice recognition software because of the way MFCCs accurately envelope the the shape of the vocal tract.
- While extracting the features, all the audio files have been sampled starting from 0.5 seconds and then timed for 3 seconds to get equal number of features.
- The sampling rate of each file is doubled to around 44 KHz. This allows each audio samples to get more features which will help classify the audio file while keeping noise at a minimum.
- The audio files were also seperated by emotion and sex.

The extracted features looks as follows:

```
In [60]: train[255:265]
```

Out[60]:

4	5	6	7	8	9	...	121	122	123	124	125	126	127	128	129	0
0582	0.243815	0.234133	0.220812	0.222221	0.232087	...	0.248799	0.253912	0.260256	0.257698	0.258209	0.256242	0.255648	0.255648	0.255701	angry
0521	0.285065	0.291352	0.303514	0.308232	0.328804	...	0.234485	0.228035	0.216631	0.214859	0.212437	0.213037	0.218348	0.223208	0.224450	fearful
0765	0.108862	0.103840	0.101478	0.107730	0.103912	...	0.066940	0.036635	0.027208	0.036532	0.053178	0.065569	0.057186	0.039764	0.021314	angry
0141	0.074467	0.089486	0.088280	0.092139	0.093846	...	0.054423	0.053604	0.055540	0.058426	0.060729	0.068808	0.088886	0.098216	0.090357	sad
0724	0.281591	0.296421	0.285957	0.260214	0.257237	...	0.299710	0.291853	0.291916	0.299710	0.299710	0.299710	0.287766	0.252755	0.243608	happy
0779	0.330779	0.330779	0.330779	0.330779	0.330779	...	0.288739	0.287423	0.283312	0.291878	0.305482	0.321055	0.327999	0.301280	0.300456	calm
0433	0.169379	0.171645	0.179289	0.190308	0.182795	...	0.149075	0.147707	0.159900	0.184663	0.187635	0.168762	0.149145	0.130382	0.120786	neutral
0036	0.238554	0.242728	0.229463	0.228398	0.243454	...	0.223064	0.207814	0.210600	0.210909	0.202713	0.192792	0.192630	0.195298	0.187149	happy
0079	0.326079	0.305091	0.284397	0.274060	0.266039	...	0.156601	0.185422	0.202734	0.204833	0.213753	0.221158	0.222267	0.185138	0.151496	sad
0975	0.172604	0.173216	0.167372	0.168891	0.178888	...	0.205757	0.200951	0.197044	0.193599	0.208915	0.228052	0.219472	0.205900	0.201549	surprised

These are array of values with lables appended to them.

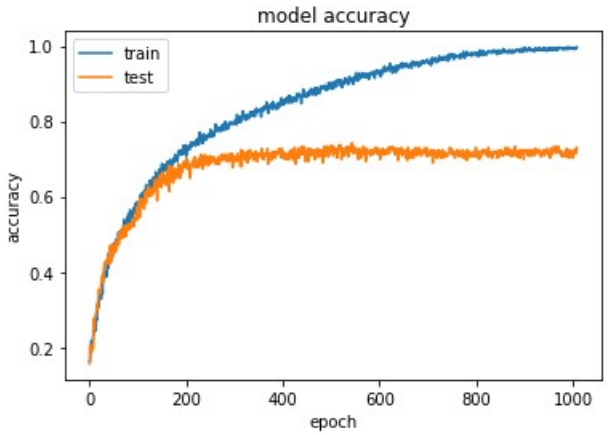
Building Models

Since the project is a classification problem, **Convolution Neural Network** seems the obivious choice. We also built **Multilayer perceptrons** and **Long Short Term Memory** models but they under-performed with very low accuracies which couldn't pass the test while predicting the right emotions.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_7 (Conv1D)            (None, 216, 128)          768
_________________________________________________________________
activation_8 (Activation)    (None, 216, 128)          0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 216, 128)          82048
_________________________________________________________________
activation_9 (Activation)    (None, 216, 128)          0
_________________________________________________________________
dropout_3 (Dropout)          (None, 216, 128)          0
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 27, 128)           0
_________________________________________________________________
conv1d_9 (Conv1D)            (None, 27, 128)           82048
_________________________________________________________________
activation_10 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_10 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_11 (Activation)   (None, 27, 128)           0
_________________________________________________________________
conv1d_11 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_12 (Activation)   (None, 27, 128)           0
_________________________________________________________________
dropout_4 (Dropout)          (None, 27, 128)           0
_________________________________________________________________
conv1d_12 (Conv1D)           (None, 27, 128)           82048
_________________________________________________________________
activation_13 (Activation)   (None, 27, 128)           0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3456)              0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                34570
_________________________________________________________________
activation_14 (Activation)   (None, 10)                0
=================================================================
Total params: 445,578
Trainable params: 445,578
Non-trainable params: 0
_________________________________________________________________
```

Building and tuning a model is a very time consuming process. The idea is to always start small without adding too many layers just for the sake of making it complex. After testing out with layers, the model which gave the max validation accuracy against test data was little more than 70%

```
In [110]: #sigmoid
          plt.plot(cnnhistory.history['acc'])
          plt.plot(cnnhistory.history['val_acc'])
          plt.title('model accuracy')
          plt.ylabel('accuracy')
          plt.xlabel('epoch')
          plt.legend(['train', 'test'], loc='upper left')
          plt.show()
```



Predictions

After tuning the model, tested it out by predicting the emotions for the test data. For a model with the given accuracy these are a sample of the actual vs predicted values.

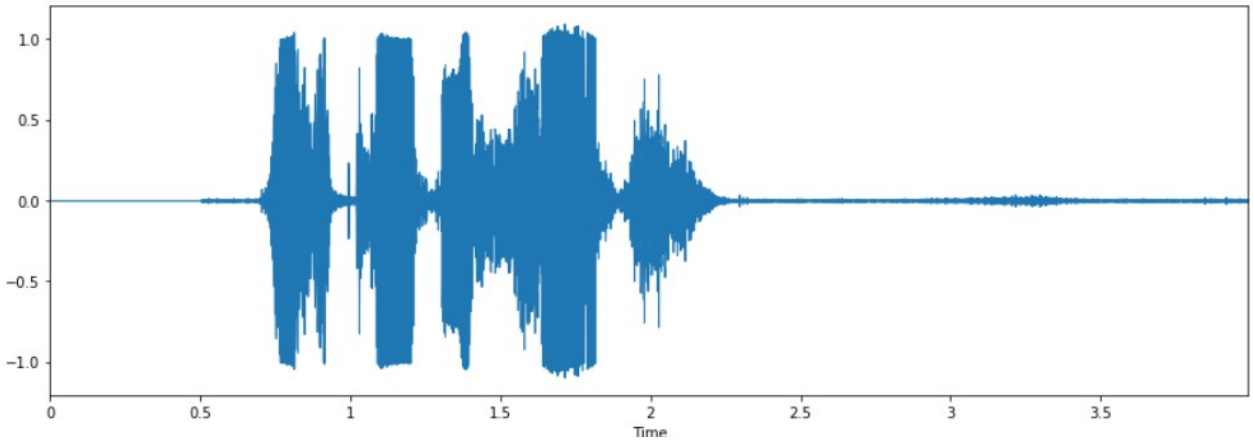
```
In [75]: finaldf[58:68]
```

Out[75]:

	actualvalues	predictedvalues
58	male_fearful	male_happy
59	male_fearful	male_fearful
60	male_fearful	male_fearful
61	male_fearful	male_fearful
62	male_sad	male_sad
63	male_fearful	male_fearful
64	male_happy	male_happy
65	female_angry	female_angry
66	female_angry	female_fearful
67	male_angry	male_angry

Testing out with live voices.

In order to test out our model on voices that were completely different than what we have in our training and test data, we recorded our own voices with dfferent emotions and predicted the outcomes using the audiorecorder.ipynb. You can see the results below: The audio contained a male voice which said "**This coffee sucks**" in a angry tone.




```
twodim= np.expand_dims(livedf2, axis=2)
```

```
livepreds = loaded_model.predict(twodim,
                         batch_size=32,
                         verbose=1)
```

```
1/1 [==============================] - 0s
```

```
livepreds
```

```
array([[  9.24052530e-22,   0.00000000e+00,   3.62402176e-26,
          1.30680162e-36,   4.47264152e-28,   1.00000000e+00,
          1.80208343e-30,   2.76873961e-27,   3.62227194e-23,
          1.67396652e-11]], dtype=float32)
```

```
livepreds1=livepreds.argmax(axis=1)
```

```
livepredictions = (lb.inverse_transform((liveabc)))
livepredictions
```

```
array(['male_angry'], dtype=object)
```

As you can see that the model has predicted the male voice and emotion accurately.

Conclusion

Building the model was a challenging task as it involved lot of trail and error methods, tuning etc. The model is very well trained to distinguish between male and female voices and it distinguishes with 100% accuracy. The model is still being tuned to detect emotions with more than 70% accuracy which is achievable by increasing the size of the dataset.