# Speech to Text emotion Detection.
# A CRISP-DM Study.

Kenneth Xavier Dsilva
ID: 10600644
10600644@mydbs.ie

Handup Date: 23/06/2022

## 1 Introduction.

This paper is a comprehensive study on the NLP ideology of Speech to Text conversion in real time and providing a crisp sentiment analysis on the input to determine the targets emotion at the current time. Being a part of an NLP project using unsupervised learning techniques we have employed a dynamic speech to text conversion technique using the Google API (*Speech-to-text: Automatic speech recognition - google cloud.* 2007) that i will discuss in length later on in this paper. The sentiment analysis however, is done via a supervised learning technique using datasets from three infamous data sources namely; Yelp, Google and IMDB (*UCI Machine Learning Repository: Sentiment Labelled Sentences Data Set.* 2015).

## 2 Business Understanding.

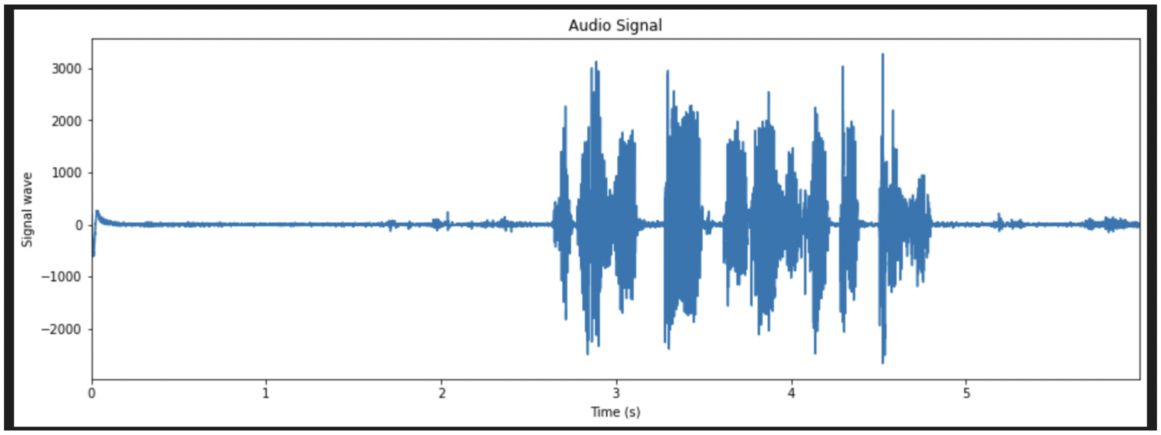
Speech to Text is a feature used and in application since the late 1999 (Shadiev et al. 2014). Its vast applications in the field of science scale from plagiarism detection to automatic language processing used by google for language conversion. Sentiment Analysis on the other hand is the most exploited tool in the branch of Natural Language Processing and combining Speech To Text with Sentiment analysis improves the effectiveness of analysing a speaker grammatically and emotionally.

A Study regarding YouTube videos showing the gender gap in the field of science was carried out by Amarasekara where he proved there existed a golfing difference between the male and female interests in the field of science by accumulating details about their comments, channels and activity. He also exposed sentiments of their subscribers and how they felt to the individual contributions (Amarasekara and Grant 2019).

Hence, to understand ones statement is important given the accent, speed and grasp of the language limitations of an individual, however, understanding the sentiment of someone is equally important. This project enables the audience to not just clearly read what is being said to them but also understand the emotion the speaker wants to deliver through their words.

## 3 Data Understanding.

The input data here is in the form of a WAV audio file generated via pyaudio from the speakers device microphone. This audio is then instantaneously saved as ”audio.wav” by pythons built in write function and is ready to be observed. We have displayed the audio by a audio wave in the time-frequency domain shown below.



Audio in Time-Frequency domain.

The audio under observation is 6 seconds long and can be controlled via a variable named ”seconds”.

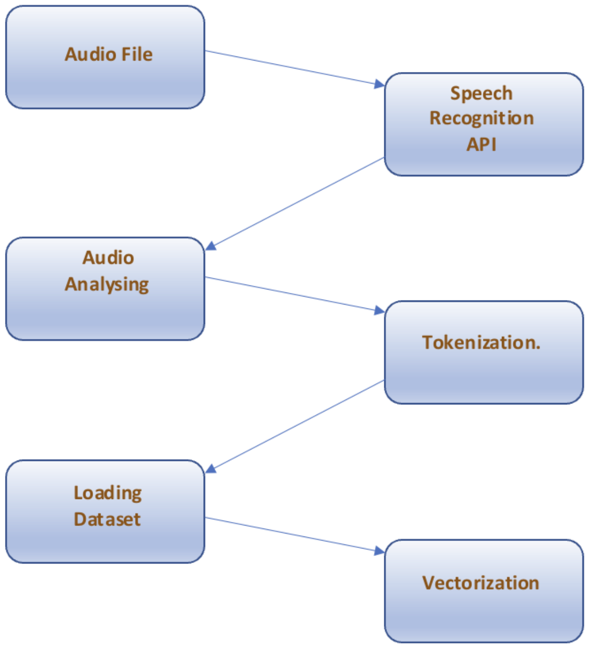
Once the audio is saved and ready to process the task of Audio to Text conversion can be commenced. This is done via the Google Speech-to-Text conversion API (*Speech-to-text: Automatic* *speech recognition - google cloud.* 2007). Here the input audio is converted into text via the Tensorflow implementation of DeepMind's WaveNet paper (Oord et al. 2016). Here A. Oord has described a Deep Neural Network approach of converting raw audio into bites and then transcribing each bite through a decoder than converts the input bites at the rate of 4800 samples/second and presenting an alphabet, which then can be stitched together to present a word and finally a sentence. However, initially this paper was aimed at the conversion of Text-to-Speech and provided a state of the art output, hence google reverse engineered the wheel and is not commercially using it to convert speech to text using the same Deep Neural Network model.

Once the text is extracted we begin the application of NLP data cleaning. As the input is a sentence extracted from a 6 second audio, the possibility of encountering symbols are minuscule, hence normalising the data to eradicate symbols and special characters is irrelevant. The data preparation for the emotion detection is presented in the following section.

## 4 Data Preparation.

This section explains the Data Preparations methods employed to arrange the raw data in an accepted format to process it correctly and generate a better more accurate and favourable output.

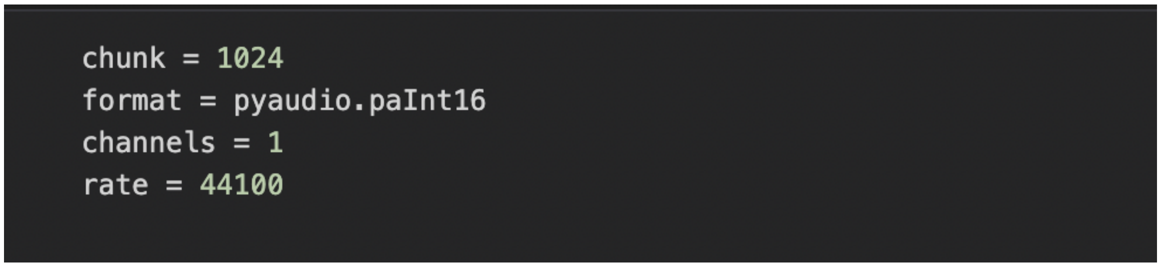
Preparing the data is mainly done in two sections. Firstly Preparing the Audio Data and then Preparing the Converted Text Data. To prepare the audio data the audio file needs to be processed and converted in the right form, sent through the API and the decoded output is then needed to be validated. Whereas, for the Decoded message, Tokenization, Loading the Dataset and Vectorizing the dataset it required to get the final emotion.



Data Preparation Steps.

### 4.1 Opening the Audio File.

This rather simple step is indeed one of the most crucial steps to undertake. Here, we open the audio file saved but before we even consider the audio file, we create some parameters that i will explain in brief. The image below shows the parameters to consider:

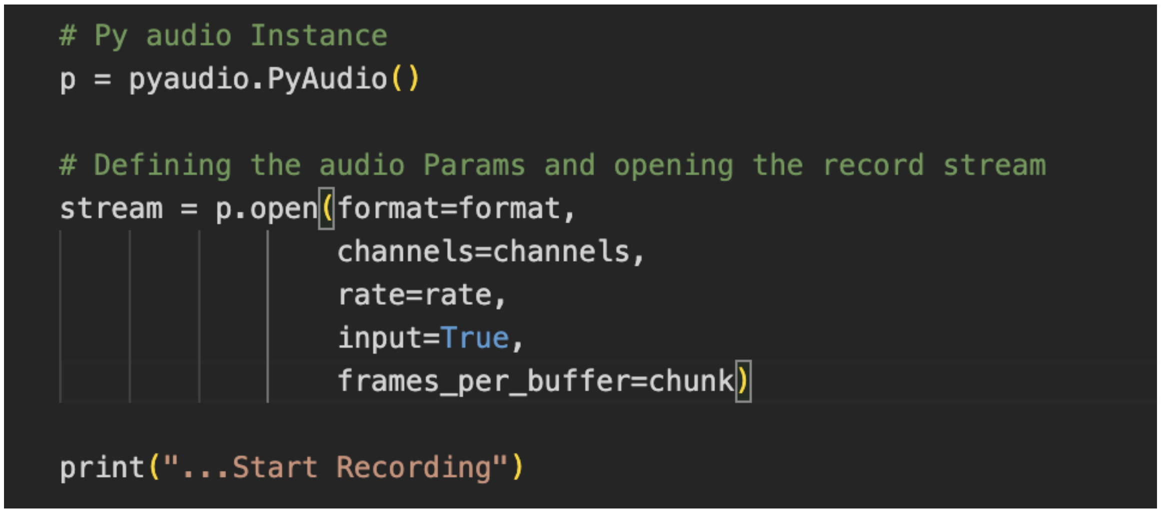


```
chunk = 1024
format = pyaudio.paInt16
channels = 1
rate = 44100
```

Pre-Set Features.

- chunk - defines the arbitary number frame in terms of a memory chunk for the recorded audio.
- format - defines output format which we will use to analysie the audio.
- channels - defines number of audio channels we will be recording the audio in.
- rate - defines what is the rate of playback as per each frame.

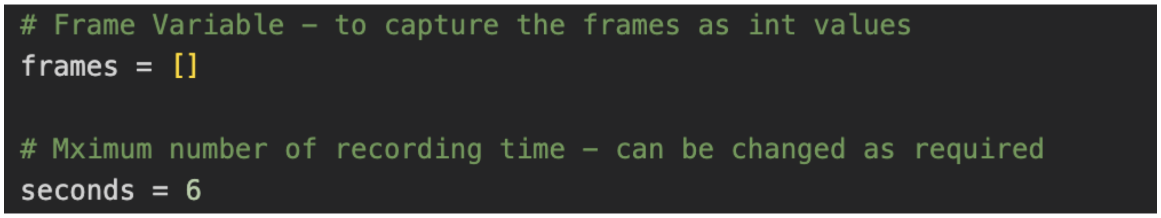
Once these Features are set we can move on to recording the audio, thanks to pyAudio feature of python we can accept the audio via the users microphone.



```
# Py audio Instance
p = pyaudio.PyAudio()

# Defining the audio Params and opening the record stream
stream = p.open(format=format,
                channels=channels,
                rate=rate,
                input=True,
                frames_per_buffer=chunk)

print("...Start Recording")
```

Using pyAudio accepting audio from user.

We can choose the length of the input audio by setting the time of the pyAudio to be available, this we can easily do by setting a time parameter to accept how many seconds the audio must be generated for during which the audio will be accepted and converted into integers in frames.



```
# Frame Variable - to capture the frames as int values
frames = []

# Mximum number of recording time - can be changed as required
seconds = 6
```

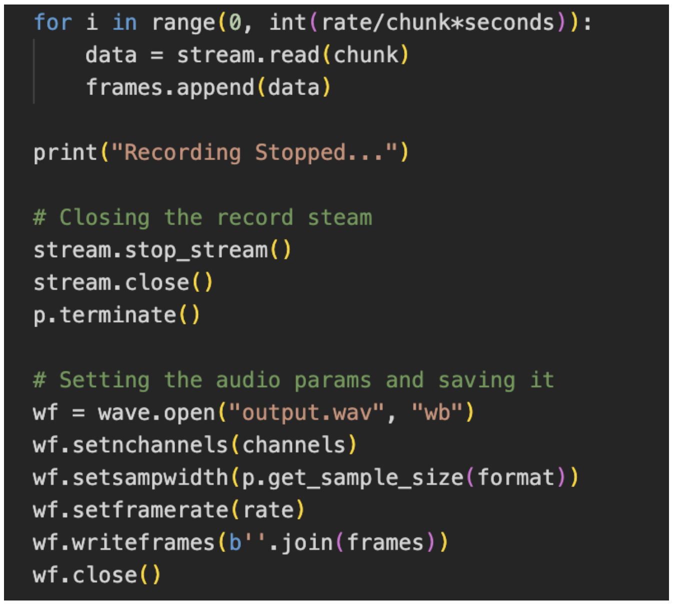
Time and Frames to accept audio.

Here the audio length will be 6 seconds long.

## 4.2 Speech Recognition API.

Once the audio file is saved and ready to process, we will now call the API to convert the audio integers into readable human-like words. For this we will use the Speech to Text API provided by

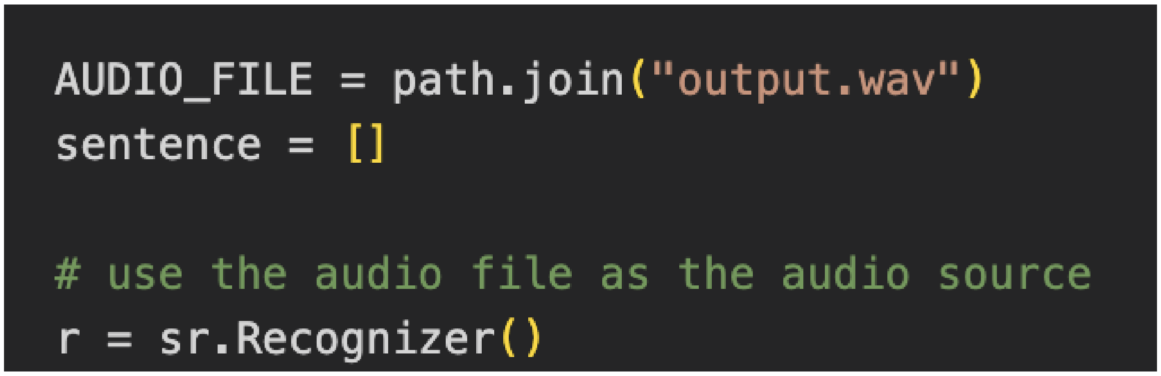
Google. Howeever, before this we need to close the audio acceptance after 6 seconds and save the file as well. This can be done with a simple python loop that keeps the microphone open for a fixed time period.



```
for i in range(0, int(rate/chunk*seconds)):
    data = stream.read(chunk)
    frames.append(data)

print("Recording Stopped...")

# Closing the record steam
stream.stop_stream()
stream.close()
p.terminate()

# Setting the audio params and saving it
wf = wave.open("output.wav", "wb")
wf.setnchannels(channels)
wf.setsampwidth(p.get_sample_size(format))
wf.setframerate(rate)
wf.writeframes(b''.join(frames))
wf.close()
```

Saving the Audio.

The saved audio can now be sent to the API via the Recognizer() function by Google API.



```
AUDIO_FILE = path.join("output.wav")
sentence = []

# use the audio file as the audio source
r = sr.Recognizer()
```

Using Google Recongnizer().

There are three main methods to use Google Speech Recongnizer (Kimura et al. 2018):

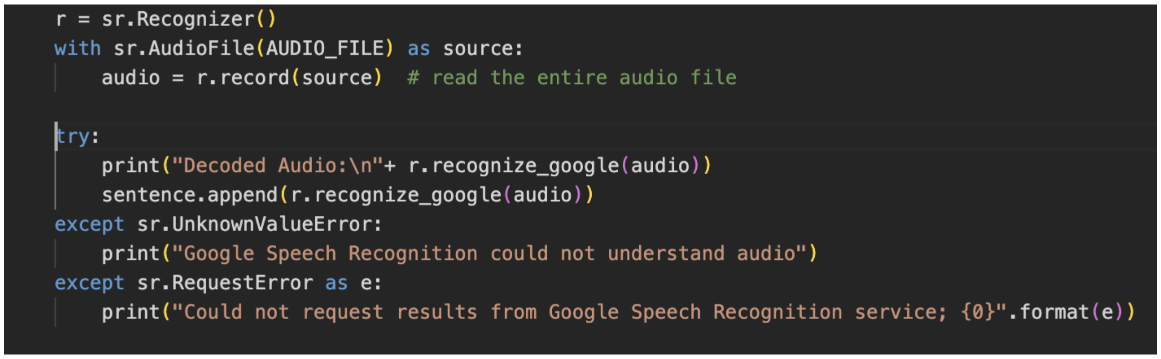
- Synchronous Recognition (REST and gRPC): propagates audio data to the API, performs recognition on the input audio data, and returns results after all audio has been processed. This has an input audio limit of 1 minute or less and provides results faster than all other methods.
- Asynchronous Recognition (REST and gRPC): transfers audio to the API and initiates a Long Running Operation. Periodic Poll is possible using this approach. The time limit here is 480 minutes.
- Streaming Recognition (gRPC): allows a bidirectional stream and propogates the audio suitable for live decoding. Here the recognition is done in real-time. It provides interm audio output with a delay of a few microseconds depending on the server speed and complexity.

## 4.3 Audio Analyzing.

Once the audio file is sent to the API the text is now awaited to be received. This is accepted in a list format. We create a list named sentence that accepts the input from the API. However, there is a great possibility that the API could not decode the audio, this could be because:

- Noise: Noise could cause interference in the audio signal causing indeterministic word decoding causing the API to fail
- Irregular Audio Rate: If the input rate isint close to the recording rate of the Audio, decoding fails.
- Connection Failure: During the decode phase any lapse in connectivity or API connectivity failure can cause the decoding to fail.
- API Key Unrecognized: Every translation or API call requires an API key. If the API key is unaccepted, outdated or broken the API call fails.

Hence, due to these necessities we create a try catch block to implement this API call.



```
r = sr.Recognizer()
with sr.AudioFile(AUDIO_FILE) as source:
    audio = r.record(source)  # read the entire audio file

try:
    print("Decoded Audio:\n"+ r.recognize_google(audio))
    sentence.append(r.recognize_google(audio))
except sr.UnknownValueError:
    print("Google Speech Recognition could not understand audio")
except sr.RequestError as e:
    print("Could not request results from Google Speech Recognition service; {0}".format(e))
```

Making the API Call.

The decoding will be explained in the Modelling section ahead.

## 4.4 WordCloud Token Representation.

Now that we have the text generated from the Google API, we can begin our NLP cleaning. As per the general NLP processing techniques there are multiple steps to be undertaken before creating the model.

According to Landolt (Landolt, Wambsganss, and Söllner 2021) there are a few set of rules to prepare the data for an NLP modelling.

- Punctuation Removal.
- Stop-Word Removal
- Converting words to Lower-Case
- Tokenization.
- Stemming and Lemmatization.

But for our approach, as the input is just 6 seconds long the maximum word capaccity can be on 20 words. Hence, many of the above mentioned methods arent really required.

The steps undertaken by us to complete the pre-processing we will just Remove the Punctuation and eradicate the Stop Words and Tokenize the entire ststement to give us the important words used to predict our output.

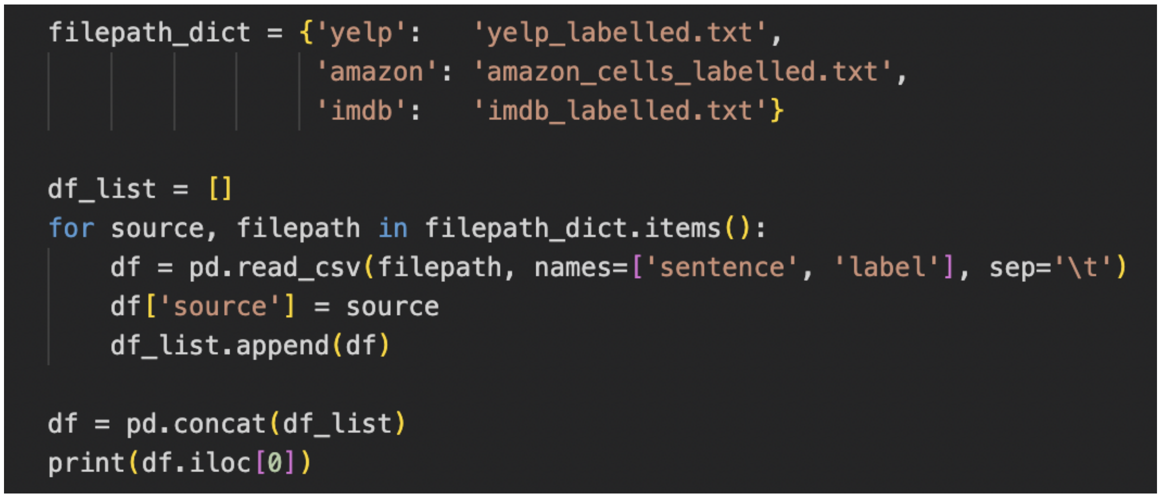
Output of WordCloud.

### 4.5 Loading the Datasets.

Now that the API has generated the sentence and we have cleaned and pre-processed the data. Now we enter the modelling preparation phase. Here we will load in three datasets

- Yelp Dataset (Asghar 2016)
- IMDB Dataset (Matthew 2020)
- Amazon Dataset (Matthew 2020)

The datasets need to be placed in the same working directory as the code, it must be in a .txt format and processed as per the image shown below.

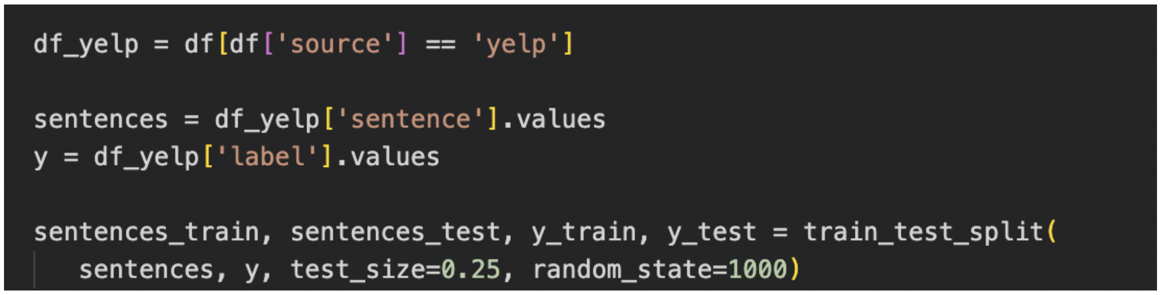


```
filepath_dict = {'yelp':   'yelp_labelled.txt',
                 'amazon': 'amazon_cells_labelled.txt',
                 'imdb':   'imdb_labelled.txt'}

df_list = []
for source, filepath in filepath_dict.items():
    df = pd.read_csv(filepath, names=['sentence', 'label'], sep='\t')
    df['source'] = source
    df_list.append(df)

df = pd.concat(df_list)
print(df.iloc[0])
```

Dataset Loading.

Now we have a working dataset loaded into a simple dictionary that can be used as per our requirement.

Finally we will load the Yelp dataset into a dataframe and split the data into the test and train sub parts. The loading of the Amazon dataset and IMDB is done in a similar way later on in the modelling phase.



```
df_yelp = df[df['source'] == 'yelp']

sentences = df_yelp['sentence'].values
y = df_yelp['label'].values

sentences_train, sentences_test, y_train, y_test = train_test_split(
    sentences, y, test_size=0.25, random_state=1000)
```

Loading Yelp Data and preparing for Modelling.

## 5 Modelling.

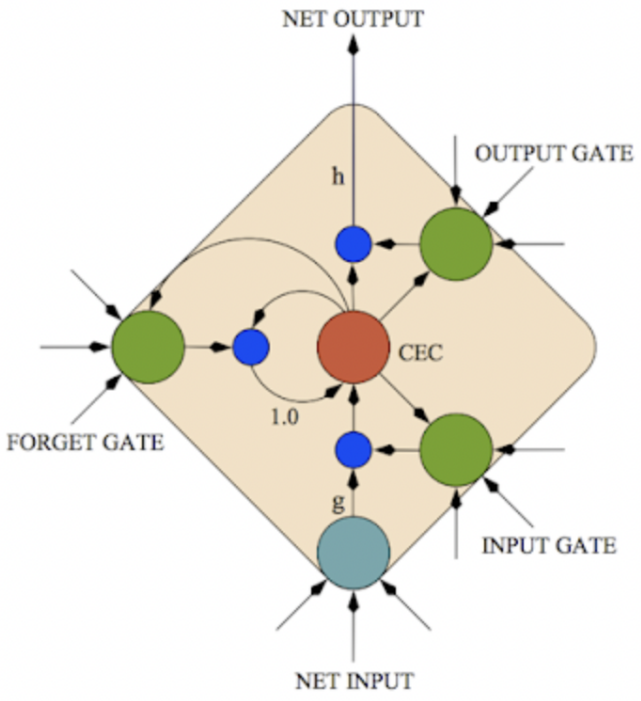
In the Modelling phase we shall discuss about two specific modelling styles carried out here:

### 5.1 Unsupervised Modelling using Google Speech-To-Text.

This modelling is done via an API namely the Google Speech to Text API. The architecture of the Google API is based on two of the most well versatile Deep Learning Techniques:

- RNN (Recurrent Neural Network).
- LSTM (Long Short Term Memory).

In 2009 when it was officially launched it adopted the GMM Gaussian Mixed Model approach however over the years it has now stabilised over LSTM with RNN.



Architecture of Google Speech-to-Text API.

For a synchronous (Short Audio below 60 Seconds) do not need a double handshake mode to be setup. A direct REST API call can be made and the output can be received. However, for Asynchronous or Streaming modes where the audio is greater than 1 minute and less than 480 minutes it would need a REST Double handshake Authorization and Authentication setup to upload and download the files.

For our purpose we have used a 6 second audio input and hence we would not require the setup for the REST API.

This model inputs the data in the form of an audio. It then breaks the input audio into frames and processes it individually under a huge corpus it can access through googles cloud services.

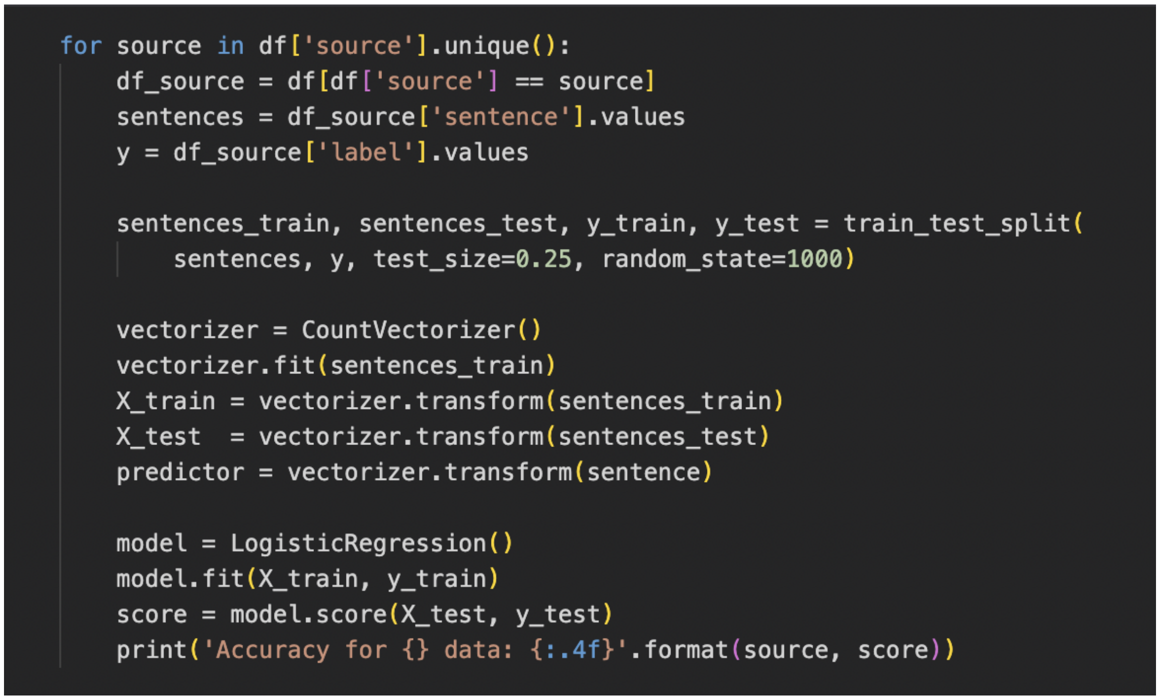
As the architecture suggests, there are three loops that each frame passes through, first the input gate where the input is fed and processed into a stream of frames.

Second the CEC is where the input frame word is matched with a word in the corpus through RNN, if the corpus and input data match it gives a feedback of '1.0' and is sent to the output gate. However, if not it stays in the loop for a predetermined time-frame and on exceeding it, its sent to the forget gate where the input is avoided.

The final stage is to stack all the outputs generated at the output gate in order of the frames received and is sent in sequence to the Net Output.

## 5.2 Supervised Modelling to Predict Emotion.

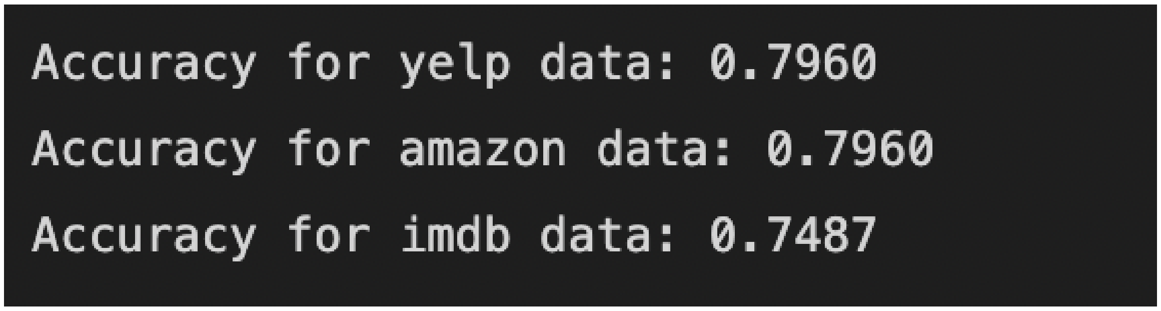
To predict the emotion based on the voice transcription provided to us by Google API we have used Logistic Regression as our mathematical model. Before we get into the modelling, we have three datasets to choose from. The best way to select a dataset is to see which fits accurately to our selected logistic regression model. Here, we vectorize the data and input the training half of the datasets into the model. We evaluate the results using the Accuracy parameter of the output of each dataset as shown below.



```python
for source in df['source'].unique():
    df_source = df[df['source'] == source]
    sentences = df_source['sentence'].values
    y = df_source['label'].values

    sentences_train, sentences_test, y_train, y_test = train_test_split(
        sentences, y, test_size=0.25, random_state=1000)

    vectorizer = CountVectorizer()
    vectorizer.fit(sentences_train)
    X_train = vectorizer.transform(sentences_train)
    X_test  = vectorizer.transform(sentences_test)
    predictor = vectorizer.transform(sentence)

    model = LogisticRegression()
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    print('Accuracy for {} data: {:.4f}'.format(source, score))
```

Vectorizing and Modelling.

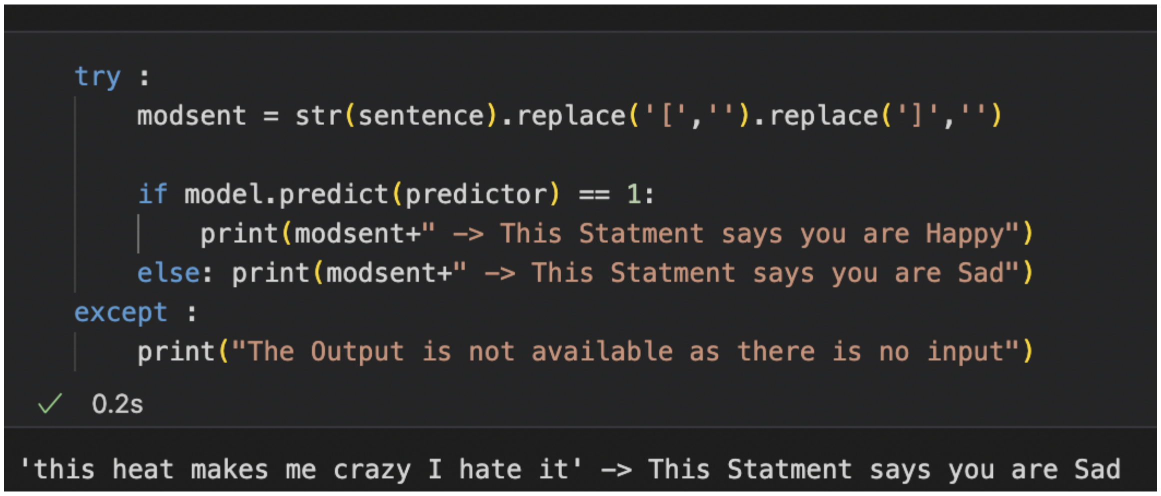
The accuracy of each dataset is shown below. Note all these datasets are run on Logistic Regression under the same parameters hence the accuracy is without any bias.



```
Accuracy for yelp data: 0.7960
Accuracy for amazon data: 0.7960
Accuracy for imdb data: 0.7487
```

Best Dataset Accuracy.

Hence, we select the Yelp Dataset and once this stage is completed we go ahead to predict the output. This is the simplest of all where we take the input we received and input it into the model to get the emotion.



```
try :
    modsent = str(sentence).replace('[','').replace(']','')

    if model.predict(predictor) == 1:
        print(modsent+" -> This Statment says you are Happy")
    else: print(modsent+" -> This Statment says you are Sad")
except :
    print("The Output is not available as there is no input")
✓ 0.2s

'this heat makes me crazy I hate it' -> This Statment says you are Sad
```

Output.

This output shows that the yelp dataset works well with the input data and our model is performing upto par. However, there are a few improvements that can be considered to make improvement in our algorithm and will be considered in our conclusion.

## 6 Deployment.

## 7 Conclusion.

## References

Amarasekara, Inoka and Will J Grant (2019). “Exploring the YouTube science communication gender gap: A sentiment analysis”. In: *Public Understanding of Science* 28.1, pp. 68–84.

Asghar, Nabiha (2016). “Yelp dataset challenge: Review rating prediction”. In: *arXiv preprint arXiv:1605.05362*.

Kimura, Takashi et al. (2018). “Comparison of speech recognition performance between Kaldi and Google cloud speech API”. In: *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. Springer, pp. 109–115.

Landolt, Severin, Thiemo Wambsganss, and Matthias Söllner (2021). “A taxonomy for deep learning in natural language processing”. In: Hawaii International Conference on System Sciences.

Matthew (2020). *Sentiment analysis and classification of Amazon, imdb, and yelp reviews.* `https://monstott.github.io/sentiment_analysis_and_classification_of_amazon_imdb_and_yelp_reviews`. Accessed:2022-07-27.

Oord, Aaron van den et al. (2016). “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499*.

Shadiev, Rustam et al. (2014). “Review of speech-to-text recognition technology for enhancing learning”. In: *Journal of Educational Technology & Society* 17.4, pp. 65–84.

*Speech-to-text: Automatic speech recognition - google cloud.* (2007). `https://cloud.google.com/speech-to-text`. Accessed:2020-07-21.

*UCI Machine Learning Repository: Sentiment Labelled Sentences Data Set.* (2015). `https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences`. Accessed:2020-07-21.