**TABLE OF CONTENTS**

# 

[1. Introduction 2](#_Toc27701798)

[1.1. Data Description: 2](#_Toc27701799)

[2. Research Gap 2](#_Toc27701800)

[3. Existing Methods 3](#_Toc27701801)

[4. Methodology Used 3](#_Toc27701802)

[4.1. Importing Data: 4](#_Toc27701803)

[4.2. Data Pre-Processing: 5](#_Toc27701804)

[4.3. Feature Engineering: 6](#_Toc27701805)

[4.4. Model Definition and Compilation: 6](#_Toc27701806)

[4.5. Model Fitting: 7](#_Toc27701807)

[5. Results 8](#_Toc27701808)

[5.1. Model Evaluation: 8](#_Toc27701809)

[5.2. Speech Recognition and Text Extraction: 9](#_Toc27701810)

[5.3. Sentiment Prediction: 10](#_Toc27701811)

[6. Discussion 10](#_Toc27701812)

[7. Conclusion and Future Scope 10](#_Toc27701813)

[8. References 11](#_Toc27701814)

[9. Code Snippet 12](#_Toc27701815)

# 

# 1. Introduction

We performed our sentiment analysis based on tweets extracted from Twitter. Tweets on Twitter are good candidates for training data for sentiment analysis due to the size restriction of 280 characters, as it forces users to focus on the message they want to communicate and can also contain slang and words used in ‘spoken’ context.

The data was obtained from Sentiment 140 and consists of 1.6 million records with two sentiments- positive and negative. We used the Long Short-Term Memory classifier in our project to learn the correct labels for binary classification. We attempted to extend the trained LSTM model to speech analytics which could be leveraged for business use cases, ranging from uncovering sentiments of consumers of a brand, to enabling product strategy or even social media content monitoring.

## 1.1. Data Description:

Data source:<http://www.sentiment140.com/>

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Variable** | **Description** | **Type** |
| 1 | Tweet ID | Unique record identifier | Number |
| 2 | Date | Date of a tweet | Day, Date and Time |
| 3 | Polarity | Tweet classification as per sentiment | Categorical – String |
| 4 | Flag | Query flag |  |
| 5 | Username | Twitter handle | String |
| 6 | Text tweet | Twitter message | String |

# 2. Research Gap

There were many challenges encountered in sentiment analysis of tweets. For example, negation allows changing a word’s meaning to its inverse meaning, to name one. It is critical to represent the process of identifying negation in the classification algorithm else it would diminish the accuracy of prediction. Here are some of the constraints and potential gaps in the realm of our study:

* Sarcasm detection
* Handling regional and vernacular terms
* Handling negation
* Context switches
* Complexities in integrating tone and emotion detection

Data pre-processing such as tokenization, stemming and lemmatization, removal of stop words are being implemented by many researchers to overcome these challenges.

We also faced issues with the very long computational time for fitting the model, as the number of parameters was high due to the large training dataset, which limited our attempts to fine-tune the model to improve model fitting performance.

# 3. Existing Methods

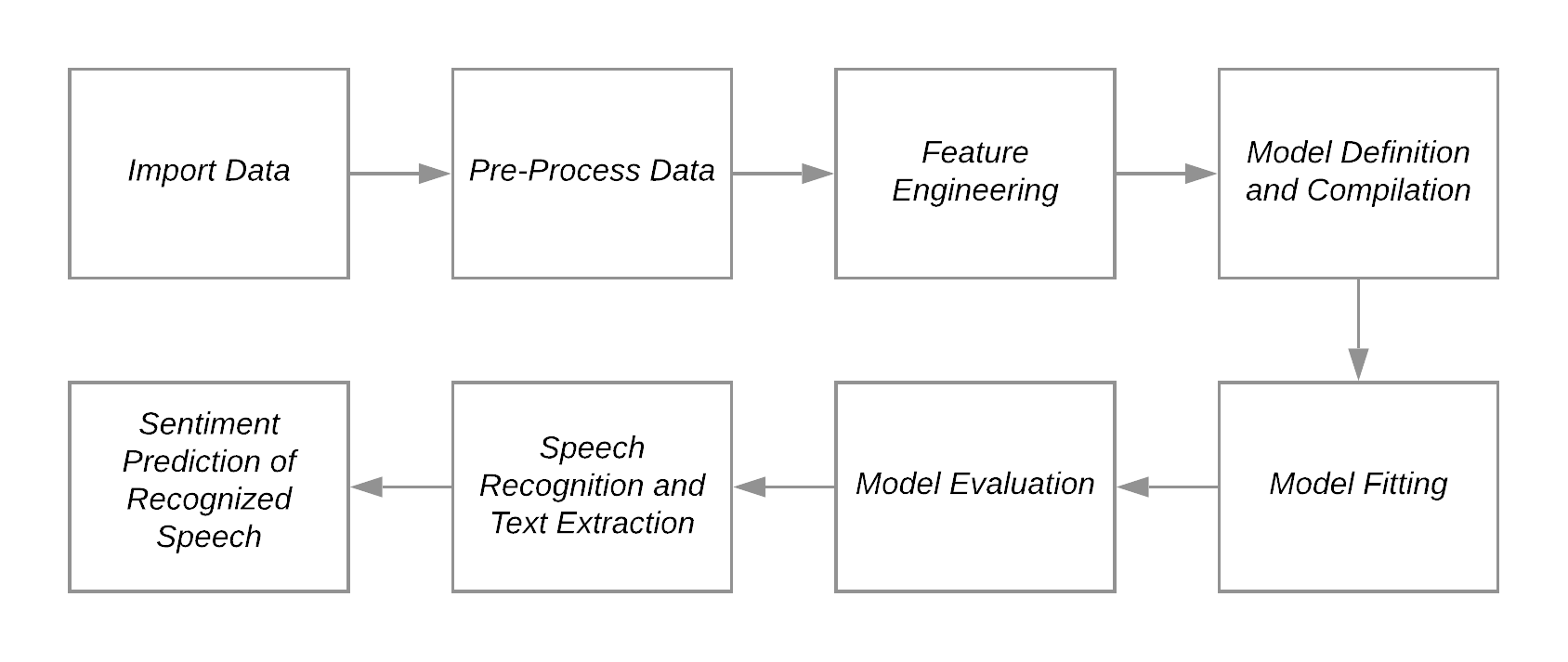
Analysis of speech sentiment classification has become a vital area of research, there are new approaches like using audio spectrograms that are being built on. The traditional method, that we have attempted to recreate is the text-based sentiment analysis approach, where speech-to-text conversion takes place, followed by text-based sentiment prediction. Some of the popular approaches/models for the same are:

* Fuzzy Neural Networks
* Naïve Bayes
* Support Vector Machine

# 4. Methodology Used

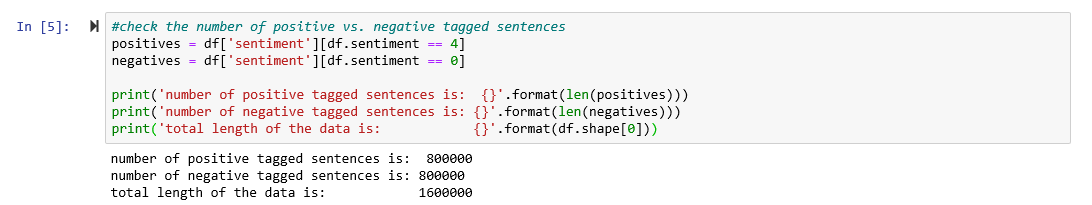
We have attempted to implement speech sentiment analysis by extending text-based sentiment analysis to speech, by incorporating speech recognition to convert speech to text. For the sentiment analysis model, we have used an LSTM (RNN) network, and the feature vector for the model was prepared using word embedding using word2vec.

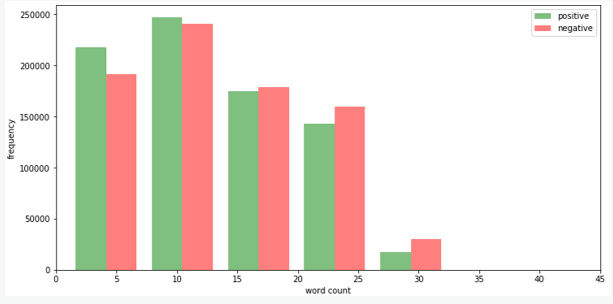
We carried out a sequence of steps to achieve our goal. The project workflow, on a high level, is as follows:



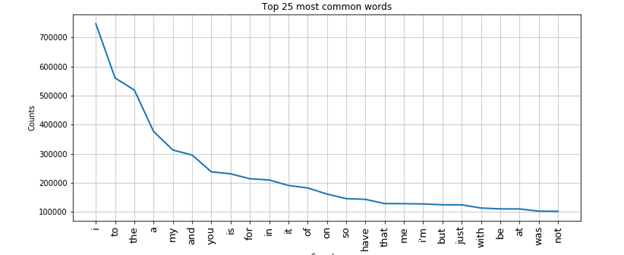
## 4.1. Importing Data:

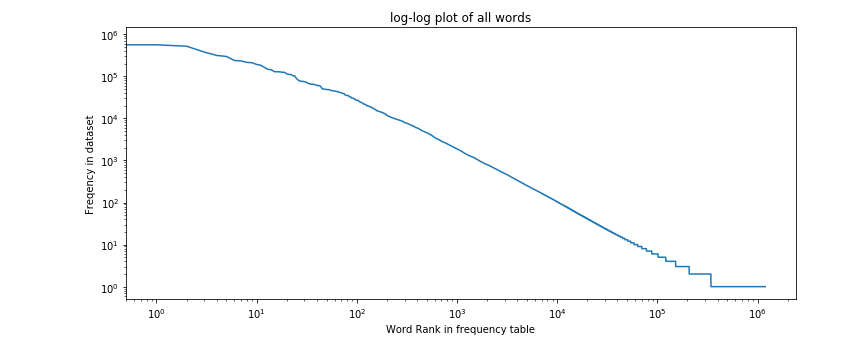
The training data was first imported from the .csv file containing data about the 1.8 million tweets using the Pandas library and stored as a dataframe. The imported dataset is a balanced one, with 800,000 positive as well as negative instances of classes.





Another observation from the graph above, was that the majority of sentences fell between 5-25 words. Overall, there were more positive tweet expressions with 10 words or less than there were negative ones.





We also drew out the most common words from the training dataset. Words like i, and, is, usually appeared equally in tweets with both negative and positive attitudes. They added very little value to the feature space.

The log-log plot for the frequency of the words complements the previous frequency graph but incorporates all words and was plotted on a base 10 logarithmic scale to help us understand the rapidly diminishing frequency of words as their rank falls. The graph shows that the dataset exhibits Zipf’s law which states that given some [corpus](https://en.wikipedia.org/wiki/Text_corpus) of [natural language](https://en.wikipedia.org/wiki/Natural_language) utterances, the frequency of any word is [inversely proportional](https://en.wikipedia.org/wiki/Inversely_proportional) to its rank in the [frequency table](https://en.wikipedia.org/wiki/Frequency_table).

We handled these common “stopwords” later in one of our data pre-processing steps.

## 4.2. Data Pre-Processing:

We carried out the pre-processing of data by first labeling the sentiments in the dataframe. We then processed the text of the tweets to remove stopwords, hyperlinks, user mentions, and special characters and carried out stemming. We utilized the nltk library and regular expressions to carry out the processing.

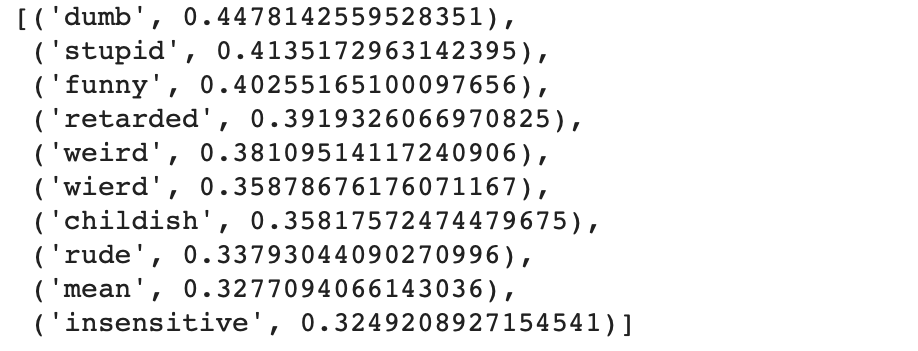
## 4.3. Feature Engineering:

We extracted the text from the dataset and then split it to convert it to a corpus of words. We then use the corpus to build the feature vector using word2vec. We input the document corpus to the word2vec model and get the output as a feature vector, which contains numerical values depending on the similarity of words in the space.

*To give a brief overview of how word2vec functions:*

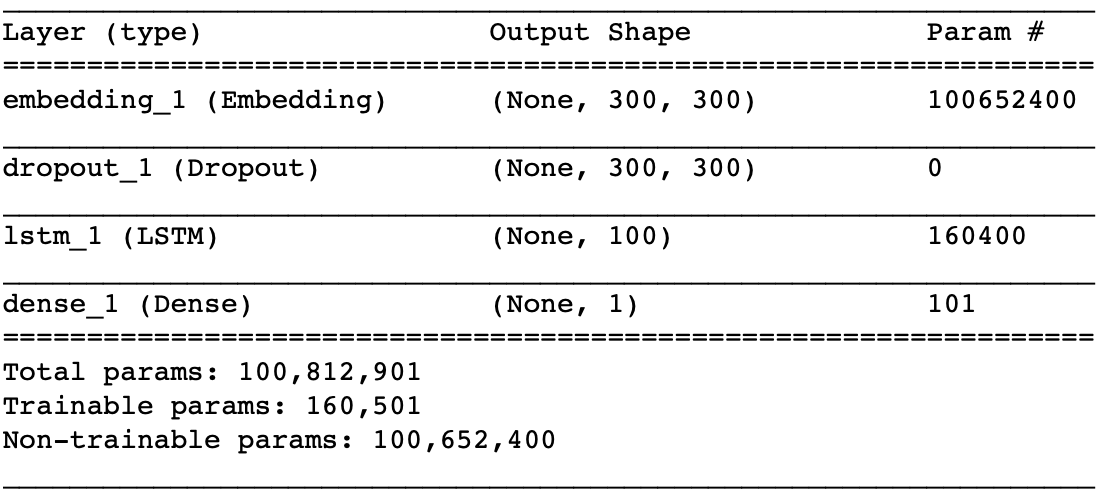
Word2vec model is a two-layer neural network that helps convert the word corpus into feature vectors by implementing context-based feature transformation. Contextually similar words are positioned together in the feature space and the measure is calculated using cosine distance of the word vectors (the closer the words appear together contextually, the lesser the cosine distance of the word vectors). Therefore, Word2vec helps convert words into a numerical machine-readable feature vector, for input for the neural network model that we will then defined.

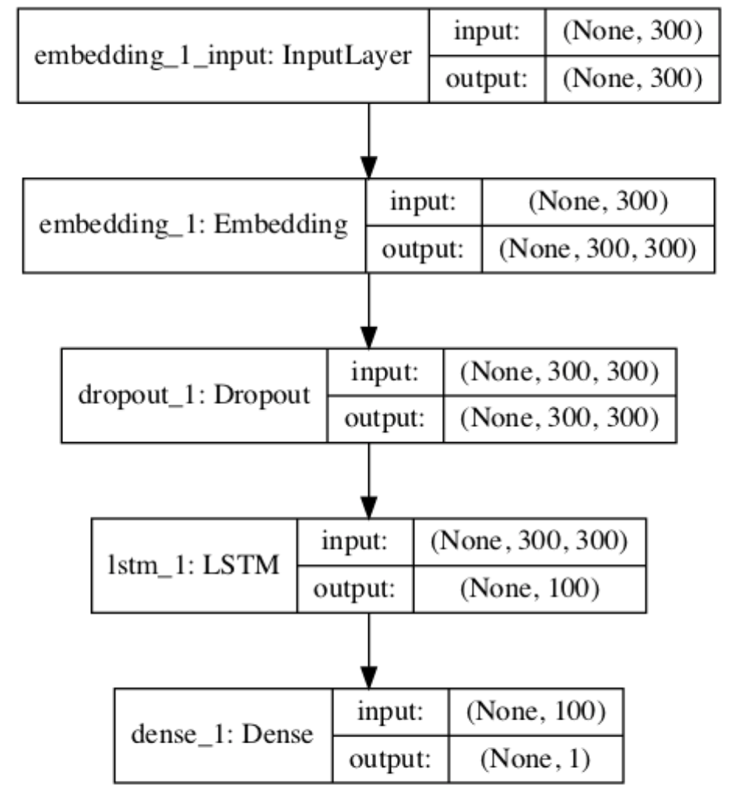
for example, In our model, we can see which words were contextually most similar to a particular word ("silly" in this case), hence closest in the word vector space:



We can observe that the word2vec model does a fairly decent job of grouping similar meaning words well, in spite of knowing nothing about the English language vocabulary.

## 4.4. Model Definition and Compilation:

We define the LSTM neural network as a three-layer network. The first layer is an embedded layer that takes in the feature vector output from the feature engineering step, after introducing some dropout, we define the recurrent LSTM layer and finally, we define the output layer with a sigmoid activation function. We compiled the model with the binary cross-entropy loss function and the Adam optimizer algorithm.

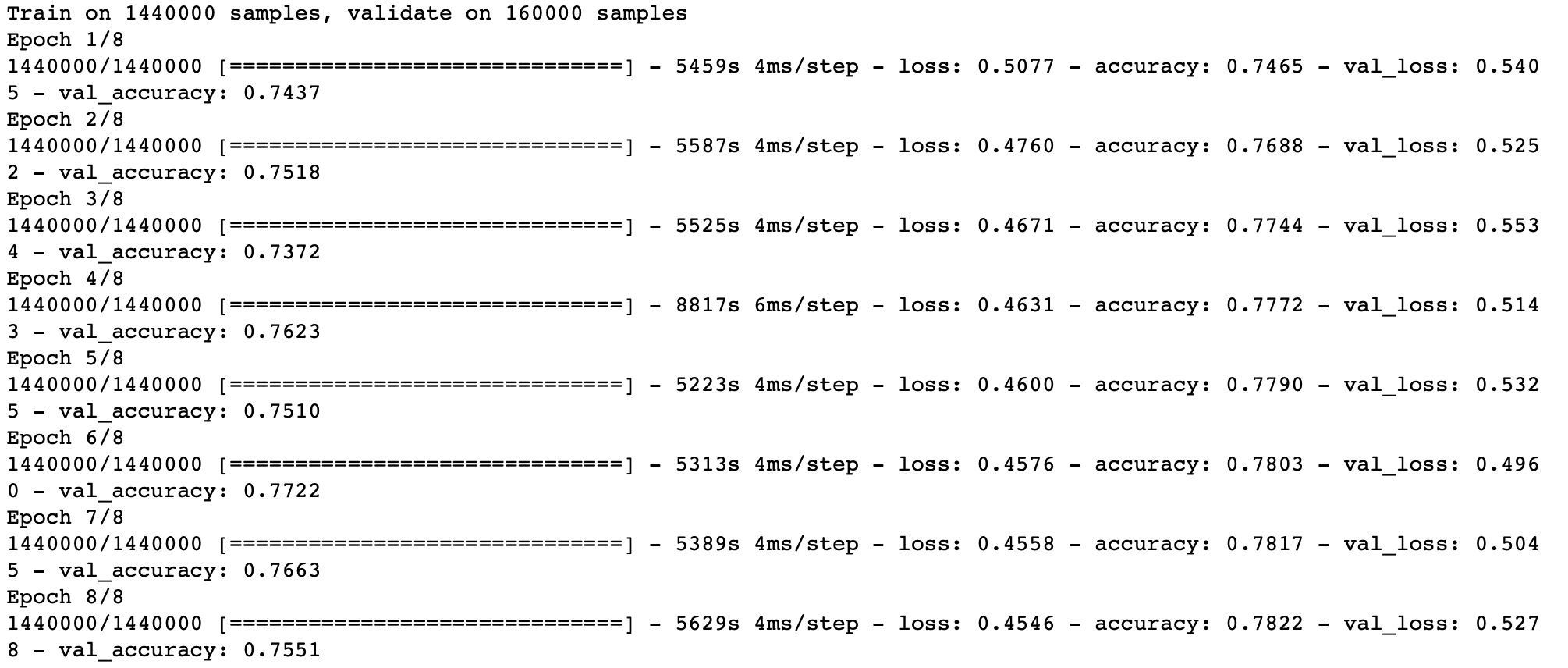
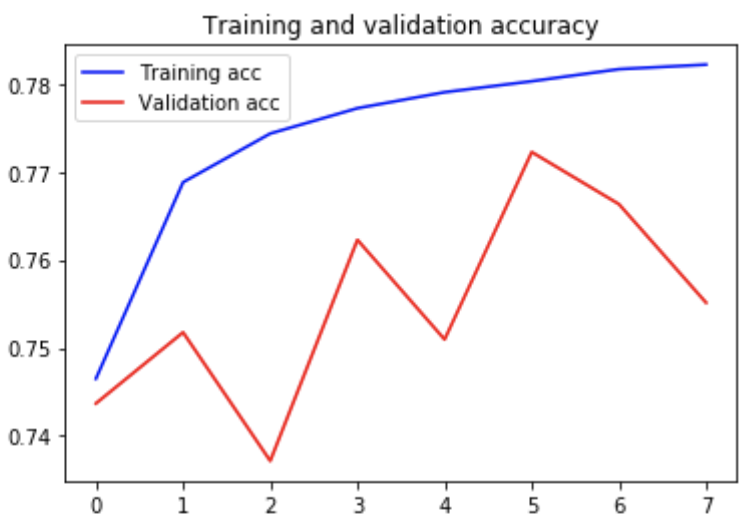
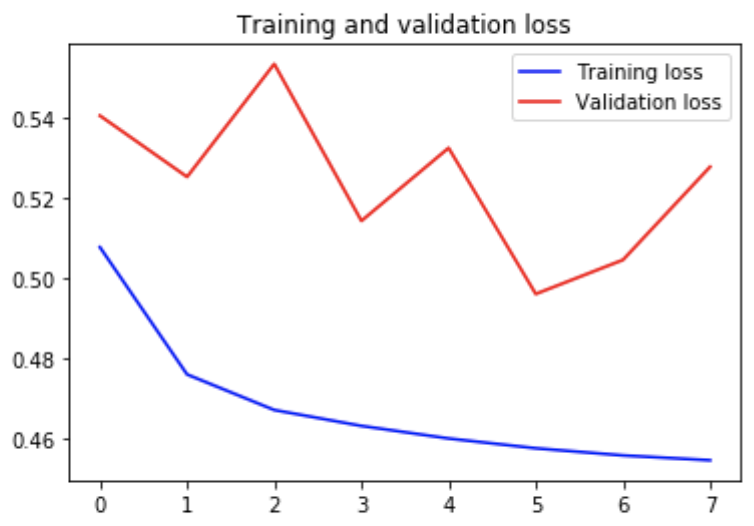


## 4.5. Model Fitting:

We fit the compiled LSTM model with the training data (along with the sentiment label) and we validated it against a validation split set (0.1) after every epoch. We fit the model over 8 epochs and used batches of

2048. We also implemented callbacks for reducing the learning rate on validation loss plateaus and early stopping based on validation accuracy over 5 epochs.

Upon fitting the model, in spite of implementing regularization techniques, we find out that the model overfits and does not perform consistently on the validation set over each epoch. This could be because we chose a small validation split set, or due to the high variability of the word occurrences in the validation data. There is scope for more tuning and one of our future goals is to try fine-tuning the model to alleviate overfitting.

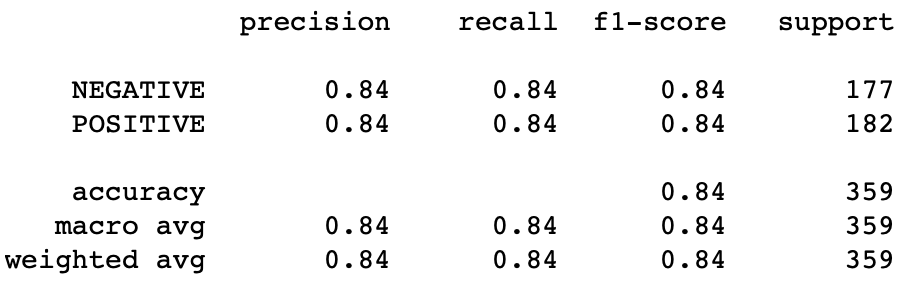


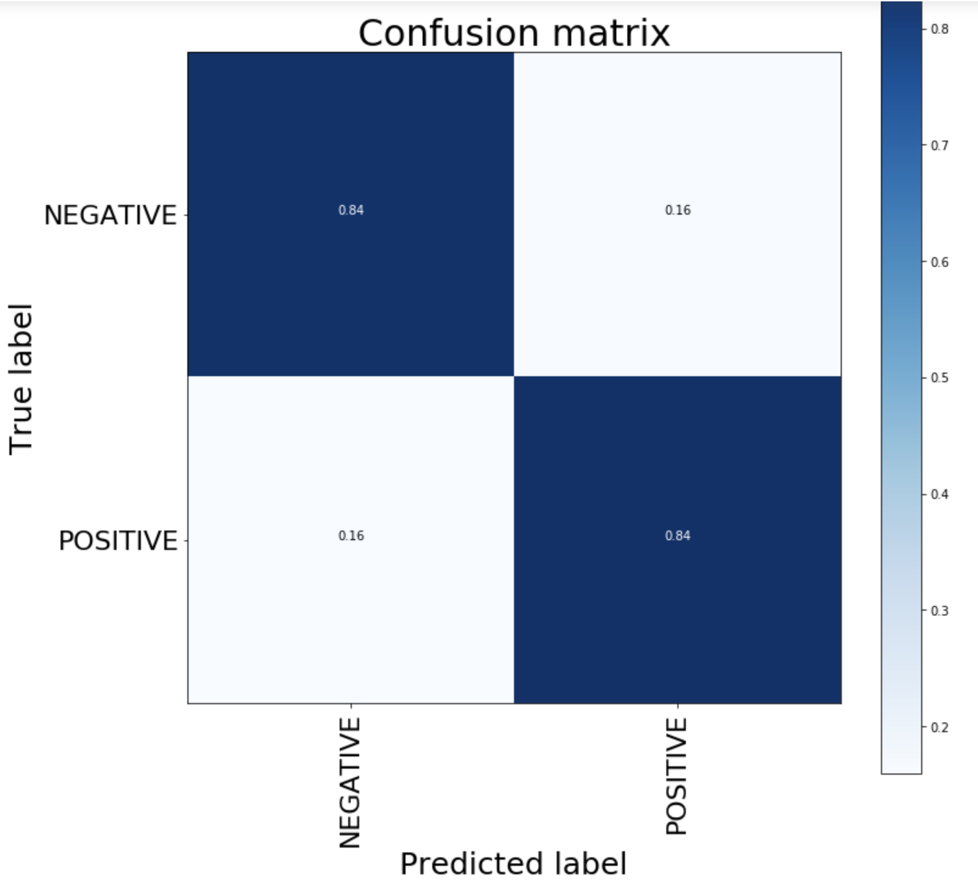
# 5. Results

## 5.1. Model Evaluation:

We then evaluate the model on untrained test data. In spite of the overfitting issue, interestingly, our model does an excellent job on the unseen data and achieves a commendable accuracy of 0.83844012.

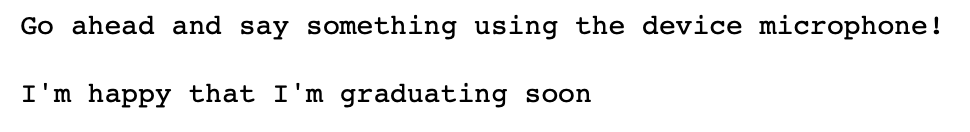
The model does an equally good job of predicting negative and positive tweets from the test dataset as manifested by the equal precision and recall values of 0.84.





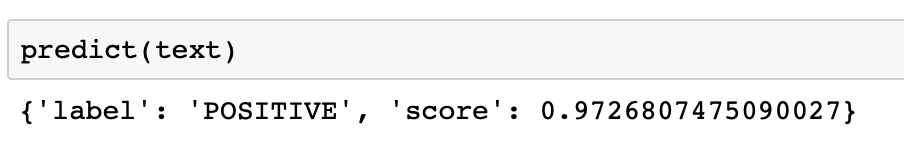
## 5.2. Speech Recognition and Text Extraction:

We implemented the speech recognition functionalities by integrating the SpeechRecognition python module into our program. This allows us to input user speech via the device microphone and helps us extract the text from speech, which we will then use for sentiment prediction.

****

## 5.3. Sentiment Prediction:

We pass the extracted text into the ’predict’ function that we define, which takes in the extracted text as a parameter. The function then tokenizes the text and uses the model that we have fitted, to predict the sentiment of the spoken text. The predicted sentiment label and the prediction score is then returned by the function.



# 6. Discussion

Text-to-speech sentiment analysis can be leveraged for a wide range of business applications. A few of these applications are:

* Brand monitoring
* Customer service
* Competitive analysis
* Social media monitoring

In more specificity, use cases in the mentioned aspects such as monitoring call center and customer support performance, gauging consumer responses, opinion analysis via interviews, developing branding and marketing strategies and detection of hate speech in online videos and podcasts have a lot of scope of sentiment analysis usage if utilized efficiently.

# 7. Conclusion and Future Scope

Overall, our model does a good job of predicting sentiment with an accuracy of almost 0.84 (0.83844) and also performs well when predicting the sentiment of spoken text. We have a few takeaways for future improvement in performance/accuracy:

* Use other approaches like using direct speech sentiment analysis where we can extract images from audio by using audio spectrograms for the spoken words and then use neural networks to carry out sentiment prediction.
* Incorporate different regularization techniques and loss functions to fine-tune the model.
* Implement different mechanisms for tokenization with emphasis on capitalization and special characters.
* Incorporate multi-class classifier to include more attitudes besides positive and negative and incorporate detailed emotions.
* Determine context by including emoticons, smileys, and emojis, which have been otherwise excluded from our original training dataset.
* Teach algorithms based on topics for compartmentalizing tweets based on the subject matter, for a more comprehensive analysis.

# 8. References

Sentiment140 dataset: Go, A., Bhayani, R. and Huang, L., 2009. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1(2009), p.12.

word2vec: Distributed Representations of Words and Phrases and their Compositionality, Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, Google Inc., (2013)

SpeechRecognition: Zhang, Anthony. 2017. Speech Recognition (version 3.8).

LSTM Neural Networks: [Sepp Hochreite](https://dl.acm.org/author_page.cfm?id=81100574709&coll=DL&dl=ACM&trk=0), [Jürgen Schmidhuber](https://dl.acm.org/author_page.cfm?id=81409592380&coll=DL&dl=ACM&trk=0), Neural Computation, Volume 9 Issue 8, November 15, 1997

Sentiment detection with Keras, word embeddings and LSTM deep learning networks: Thomas Ebermann, llib blog (2018)

# 

# 

# 

# 9. Code Snippet

import pandas as pd

import numpy as np

import nltk

import matplotlib.pyplot as plt

from nltk.corpus import stopwords

from nltk.tokenize import TweetTokenizer

from collections import Counter

fpath = 'C:/Users/Meghna Das/Desktop/trainingandtestdata/training.1600000.processed.noemoticon.csv'

cols = ['sentiment','id','date','flag','user','text']

df = pd.read\_csv(fpath,header=None, names =cols, encoding = 'ISO-8859-1')

print(df.head())

positives = df['sentiment'][df.sentiment == 4]

negatives = df['sentiment'][df.sentiment == 0]

print('number of positive tagged sentences is: {}'.format(len(positives)))

print('number of negative tagged sentences is: {}'.format(len(negatives)))

def word\_count(sentence):

return len(sentence.split())

df['word count'] = df['text'].apply(word\_count)

df.head(3)

x = df['word count'][df.sentiment == 4]

y = df['word count'][df.sentiment == 0]

plt.figure(figsize=(12,6))

plt.xlim(0,45)

plt.xlabel('word count')

plt.ylabel('frequency')

g = plt.hist([x, y], color=['g','r'], alpha=0.5, label=['positive','negative'])

plt.legend(loc='upper right')

all\_words = []

for line in list(df['text']):

words = line.split()

for word in words:

all\_words.append(word.lower())

Counter(all\_words).most\_common(10)

plt.figure(figsize=(12,5))

plt.title("Top 25 most common words")

plt.xticks(fontsize=13, rotation=90)

fd = nltk.FreqDist(all\_words)

fd.plot(25,cumulative=False)

word\_counts = sorted(Counter(all\_words).values(), reverse=True)

plt.figure(figsize=(12,5))

plt.loglog(word\_counts, linestyle='-', linewidth=1.5)

plt.ylabel("Freqency in dataset")

plt.xlabel("Word Rank in frequency table")

plt.title("log-log plot of all words")

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import confusion\_matrix, classification\_report, accuracy\_score

from sklearn.manifold import TSNE

from sklearn.feature\_extraction.text import TfidfVectorizer

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Sequential

from keras.layers import Activation, Dense, Dropout, Embedding, Flatten, Conv1D, MaxPooling1D, LSTM

from keras import utils

from keras.callbacks import ReduceLROnPlateau, EarlyStopping

import nltk

from nltk.corpus import stopwords

from nltk.stem import SnowballStemmer

#nltk.download('stopwords')

datacleanRE = "@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+"

seqlen = 300

epsize = 8

BATCH\_SIZE = 1024

POSITIVE = "POSITIVE"

NEGATIVE = "NEGATIVE"

NEUTRAL = "NEUTRAL"

SENTIMENT\_THRESHOLDS = (0.4, 0.6)

decode\_map = {'0': "NEGATIVE", '2': "NEUTRAL", '4': "POSITIVE"}

def decode\_sentiment(label):

return decode\_map[str(label)]

df.sentiment = df.sentiment.apply(lambda x: decode\_sentiment(x))

print(df.head(10))

import re

stpwords = stopwords.words("english")

stemmer = SnowballStemmer("english")

def preprocess(text, stem=False):

text = re.sub(datacleanRE, ' ', str(text).lower()).strip()

tokens = []

for token in text.split():

if token not in stpwords:

if stem:

tokens.append(stemmer.stem(token))

else:

tokens.append(token)

return " ".join(tokens)

df.text = df.text.apply(lambda x: preprocess(x))

df\_train = df

test\_path= '/Users/akndiwan/Desktop/Python Proj Final/trainingandtestdata/testdata.manual.2009.06.14.csv'

df\_test= pd.read\_csv(test\_path,header=None, names =cols, encoding = 'ISO-8859-1')

df\_test = df\_test[df\_test.sentiment != 2]

df\_test.sentiment = df\_test.sentiment.apply(lambda x: decode\_sentiment(x))

print(df\_test.head())

print(len(df\_test))

df\_test.text=df\_test.text.apply(lambda x: preprocess(x))

documents = [\_text.split() for \_text in df\_train.text]

import gensim

w2v\_model = gensim.models.word2vec.Word2Vec(size=300,

window=7,

min\_count=10,

workers=8)

w2v\_model.build\_vocab(documents)

words = w2v\_model.wv.vocab.keys()

vocab\_size = len(words)

print("Vocab size", vocab\_size)

w2v\_model.train(documents, total\_examples=len(documents), epochs=32)

w2v\_model.most\_similar("silly")

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(df\_train.text)

vocab\_size = len(tokenizer.word\_index) + 1

print("Total words", vocab\_size)

x\_train = pad\_sequences(tokenizer.texts\_to\_sequences(df\_train.text), maxlen=seqlen)

x\_test = pad\_sequences(tokenizer.texts\_to\_sequences(df\_test.text), maxlen=seqlen)

encoder = LabelEncoder()

encoder.fit(df\_train.sentiment.tolist())

y\_train = encoder.transform(df\_train.sentiment.tolist())

y\_test = encoder.transform(df\_test.sentiment.tolist())

y\_train = y\_train.reshape(-1,1)

y\_test = y\_test.reshape(-1,1)

print("x\_train", x\_train.shape)

print("y\_train", y\_train.shape)

print("\n")

print("x\_test", x\_test.shape)

print("y\_test", y\_test.shape)

W2V\_SIZE = 300

W2V\_WINDOW = 7

W2V\_EPOCH = 32

W2V\_MIN\_COUNT = 10

embedding\_matrix = np.zeros((vocab\_size, W2V\_SIZE))

for word, i in tokenizer.word\_index.items():

if word in w2v\_model.wv:

embedding\_matrix[i] = w2v\_model.wv[word]

print(embedding\_matrix.shape)

embedding\_layer = Embedding(vocab\_size, W2V\_SIZE, weights=[embedding\_matrix], input\_length=seqlen, trainable=False)

from keras.utils.vis\_utils import plot\_model

model = Sequential()

model.add(embedding\_layer)

model.add(Dropout(0.3))

model.add(LSTM(100, dropout=0.3, recurrent\_dropout=0.3))

model.add(Dense(1, activation='sigmoid'))

model.summary()

model.compile(loss='binary\_crossentropy', optimizer="adam", metrics=['accuracy'])

#!conda install graphviz

from keras.utils.vis\_utils import plot\_model

plot\_model(model, to\_file='model\_plot.png', show\_shapes=True, show\_layer\_names=True)

callbacks = [ ReduceLROnPlateau(monitor='val\_loss', patience=5, cooldown=0),

EarlyStopping(monitor='val\_acc', min\_delta=1e-4, patience=5)]

epsize = 8

BATCH\_SIZE = 2048

history = model.fit(x\_train, y\_train, batch\_size=BATCH\_SIZE, epochs=epsize, validation\_split=0.1, verbose=1, callbacks=callbacks)

score = model.evaluate(x\_test, y\_test, batch\_size=BATCH\_SIZE)

print()

print("Accuracy:",score[1])

print("Loss:",score[0])

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training acc')

plt.plot(epochs, val\_acc, 'r', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label='Training loss')

plt.plot(epochs, val\_loss, 'r', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

history.history

def decode\_sentiment(score, include\_neutral=True):

if include\_neutral:

label = NEUTRAL

if score <= SENTIMENT\_THRESHOLDS[0]:

label = NEGATIVE

elif score >= SENTIMENT\_THRESHOLDS[1]:

label = POSITIVE

return label

else:

return NEGATIVE if score < 0.5 else POSITIVE

def predict(text, include\_neutral=True):

x\_test = pad\_sequences(tokenizer.texts\_to\_sequences([text]), maxlen=seqlen)

score = model.predict([x\_test])[0]

label = decode\_sentiment(score, include\_neutral=include\_neutral)

return {"label": label, "score": float(score)}

predict("Machine learning is love!")

ypred1d = []

ytest1d = list(df\_test.sentiment)

scores = model.predict(x\_test, verbose=1, batch\_size=8000)

ypred1d = [decode\_sentiment(score, include\_neutral=False) for score in scores]

def plot\_confusion\_matrix(cm, classes,

title='Confusion matrix',

cmap=plt.cm.Blues):

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title, fontsize=30)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=90, fontsize=22)

plt.yticks(tick\_marks, classes, fontsize=22)

fm = '.2f'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fm),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label', fontsize=25)

plt.xlabel('Predicted label', fontsize=25)

import itertools

cnf\_matrix = confusion\_matrix(ytest1d, ypred1d)

plt.figure(figsize=(12,12))

plot\_confusion\_matrix(cnf\_matrix, classes=df\_train.sentiment.unique(), title="Confusion matrix")

plt.show()

print(classification\_report(ytest1d, ypred1d))

accuracy\_score(ytest1d, ypred1d)

import speech\_recognition as sr

r = sr.Recognizer()

with sr.Microphone() as source:

print("Go ahead and say something using the device microphone! \n")

audio = r.listen(source)

text = r.recognize\_google(audio)

print(text)

predict(text)