

DEEP LEARNING (CS 6005)

MINI PROJECT ASSIGNMENT 3

NATURAL LANGUAGE PROCESSING (NLP)

NLP methods for Automated Hate-Speech and Offensive Language
Detection

BY

NAME: SRIDHAR.S

ROLL NO: 2018103068

BATCH: P

DATE: 21.05.2021

PROBLEM STATEMENT:

Hate speech is currently of broad and current interest in the domain of social media. The anonymity and flexibility afforded by the Internet has made it easy for users to communicate in an aggressive manner. And as the amount of online hate speech is increasing, methods that automatically detect hate speech is very much required. Therefore, the goal of this project is to look at how **Natural Language Processing** applies in **detecting hate-speech and offensive language**. The task of the classifier is to assigns each tweet to one of the categories of a Twitter dataset: ***hate, offensive language, and neither.***

DATASET DESCRIPTION:

NAME: HATE SPEECH AND OFFENSIVE LANGUAGE

SOURCE: <https://www.kaggle.com/mrmorj/hate-speech-and-offensive-language-dataset>

DESCRIPTION:

Dataset using Twitter **data**, is used to research **hate-speech** detection. The text is classified as: **hate-speech, offensive language, and neither**. Due to the nature of the study, it's important to note that this **dataset** contains text that can be considered racist, sexist, homophobic, or generally **offensive**.

- 24k tweets labelled as hate speech, offensive language, or neither.
- Attributes:
 - Count - Number of CrowdFlower users who coded each tweet (min is 3, sometimes more users coded a tweet when judgments were determined to be unreliable by CF).
 - Hate_speech - Number of CF users who judged the tweet to be hate speech.
 - Offensive_language - Number of CF users who judged the tweet to be offensive.
 - Neither - Number of CF users who judged the tweet to be neither offensive nor non-offensive.
 - Class - Class label for majority of CF users.
 - 0 - hate speech,
 - 1 - offensive language,
 - 2 – neither
 - Tweet - Text tweets, including numerals and special characters. All tweets are unique and valid without any mismatched and missing values.

Class Split up:

- 1430 tweets are classified as hate speech
- 19190 tweets as offensive language
- 4163 tweets as normal language.

MODULES:

Module 1: Importing Necessary libraries, Loading Dataset & Visualizing Dataset

The necessary libraries which are used to for pre-processing of text data, building LSTM model like Keras, Sequential, Dense, Dropout, Flatten, NLTK etc are imported. The dataset is loaded and visualized.

Module 2: Data Preparation

In this module the data is processed the various steps involved in processing dataset include:

- Removing Blank spaces
- Removing Special Characters (include @, #, \$)
- Removing url
- Removing punctuations
- Removing Capitalization
- Tokenizing
 - **Tokenization** - A token is a piece of a whole, so a word is a token in a sentence is a token in a paragraph. Tokenization is the process of splitting a string into a list of tokens.
- Removal of Stopwords
 - **Stopwords** – They are words which are filtered out before or after processing of natural language data (text)
- Stemming
 - **Stemming** – It is a process where words are reduced to a root by removing inflection through dropping unnecessary characters, usually a suffix.

Module 3: Feature Extraction

In this module the features are extracted for the pre-processed data. The different techniques used for feature extraction are:

- Word Level One Hot Encoding
- TFIDF Vectorizer
- Bag of Words [COUNT Vectorizer]

Word Level One Hot Encoding:

Keras has built-in utilities for doing one-hot encoding text at the word level or character level, starting from raw text data. It takes care of a number of important features, such as

stripping special characters from strings, or only taking into the top N most common words in your dataset (a common restriction to avoid dealing with huge input vector spaces).

TFIDF Vectorizer

TFIDF Vectorizer that is an inbuilt library in python which convert a collection of raw documents to a matrix of TF-IDF (Term Frequency – Inverse Document Frequency) features. It uses two statistical methods and they are:

- **Term Frequency** – It refers to the total number of times a given term t appears in the document doc against (per) the total number of all words in the document.
- **Inverse Document Frequency** – It is a measure of how much information the word provides and it measures the weight of the given word in the entire document. It shows how common or rare a given word is across all document.

Bag of Words [COUNT Vectorizer]

The bag-of-words model is a way of representing text data when modeling text with machine learning and deep learning algorithms. The bag-of-words model is simple to understand and implement and has seen great success in problems such as language modeling and document classification.

Module 4: Building a Long Short-Term Memory (LSTM) model

The LSTM model is build using various layers like:

- **Embedding Layer:** It is one of the available layers in Keras. It can be used for neural network on text data. This is mainly used in Natural Language Processing related applications such as language modeling, but it can also be used with other tasks that involve neural networks. An embedded layer with **input length** as **5000** and **output dimension 20** is used.
- **LSTM Layer:** Based on available runtime hardware and constraints, this layer will choose different implementations (cuDNN-based or pure-TensorFlow) to maximize the performance. If a GPU is available and all the arguments to the layer meet the requirement of the CuDNN kernel (see below for details), the layer will use a fast cuDNN implementation. LSTM Layer with **15 units** and **dropout 50%** is added.
- **Flatten Layer:** Transforms the format of the images from a 2d-array to a 1d-array
- **Regularization function:** I used **Dropout** technique to regularize which specifies the percentage of neurons to be dropped at each iteration.

- **Dense Layer:** The **dense layer** is a fully connected **layer**, meaning all the neurons in a **layer** are connected to those in the next **layer**. Two dense layers are used one with 512 neurons and other with 3 neurons.

Activation function used:

- Dense Layer 1 - **Relu**: given a value x , returns $\max(x, 0)$.
- Output layer (Dense Layer 2) - **Softmax**: 3 neurons, probability that the tweet belongs to one of the classes.

The model built is compiled is compiled with parameters such as:

- **Optimizer:** **adam** = RMSProp + Momentum.
 - Momentum = takes into account past gradient to have a better update.
 - RMSProp = exponentially weighted average of the squares of past gradients.
- **Loss function:** I used **categorical_crossentropy** for classification, each tweet belongs to one class only

Module 5: Model Evaluation

- The model is evaluated by plotting **Accuracy and loss curves**
- The model is also evaluated by constructing **Confusion Matrix**
- Test score is obtained

Module 6: Prediction

- The model predicts the target label for the test sample

PROGRAM SNAPSHOTS:

MODULE 1:

#Importing libraries

```
In [1]: import pandas as panda
import numpy as np
import string

import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import *
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import WordNetLemmatizer

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras import regularizers
from keras import backend as K
from keras.callbacks import ModelCheckpoint

from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop, Adam

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

#Loading Dataset

```
In [2]: data = panda.read_csv("labeled_data.csv")
data
```

Out[2]:

	index	count	hate_speech	offensive_language	neither	class	tweet
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	1	3	0	3	0	1	!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	2	3	0	3	0	1	!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	3	0	2	1	1	!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...
...
24778	25291	3	0	2	1	1	you's a muthaf***in lie “@LifeAsKing: @2...
24779	25292	3	0	1	2	2	you've gone and broke the wrong heart baby, an...
24780	25294	3	0	3	0	1	young buck wanna eat!!.. dat nigguh like I ain...
24781	25295	6	0	6	0	1	youu got wild bitches tellin you lies
24782	25296	3	0	0	3	2	~~~Ruffled Ntac Eileen Dahlia - Beautiful col...

24783 rows × 7 columns

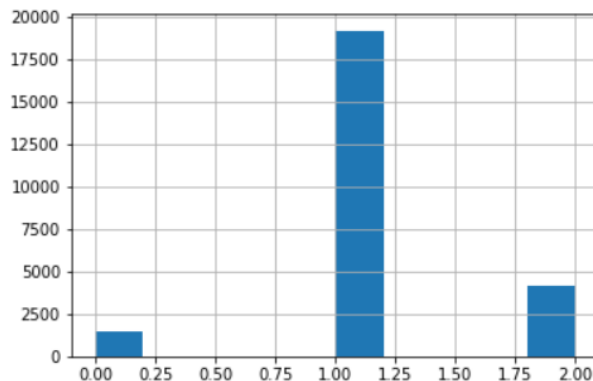
#Visualizing datatypes of attributes and target class split up

```
In [3]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24783 entries, 0 to 24782
Data columns (total 7 columns):
#   Column              Non-Null Count  Dtype
---  ---
0   index               24783 non-null  int64
1   count               24783 non-null  int64
2   hate_speech         24783 non-null  int64
3   offensive_language  24783 non-null  int64
4   neither             24783 non-null  int64
5   class               24783 non-null  int64
6   tweet               24783 non-null  object
dtypes: int64(6), object(1)
memory usage: 1.3+ MB
```

```
In [4]: data['class'].hist()
```

```
Out[4]: <AxesSubplot:>
```



MODULE 2:

#Defining Remove space function

```
In [5]: tweet=data.tweet
```

```
In [6]: def remove_space(tweet):
```

```
    regex_pat = re.compile(r'\s+')
    tweet_space = tweet.str.replace(regex_pat, ' ')
    newtweet=tweet.str.replace(r'\s+', ' ')
    newtweet=newtweet.str.replace(r'^\s+|\s+?$', '')
    newtweet=newtweet.str.replace(r'\d+(\.\d+)?', 'numbr')
    tweet_lower = newtweet.str.lower()

    return tweet_lower
```

```
In [7]: tweets_space= remove_space(tweet)
data["tweets_w/o_space"]=tweets_space
data.head()
```

```
Out[7]:
```

	index	count	hate_speech	offensive_language	neither	class	tweet	tweets_w/o_space
0	0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...	!!! rt @mayasolovely: as a woman you shouldn't...
1	1	3	0	3	0	1	!!!! RT @mleew17: boy dats cold...tyga dwn ba...	!!!! rt @mleewnumbr: boy dats cold...tyga dwn...
2	2	3	0	3	0	1	!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...	!!!!!! rt @urkindofbrand dawg!!!! rt @numbrs...
3	3	3	0	2	1	1	!!!!!!! RT @C_G_Anderson: @viva_based she lo...	!!!!!!! rt @c_g_anderson: @viva_based she lo...
4	4	6	0	6	0	1	!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...	!!!!!!!!!!!! rt @shenikaroberts: the shit you...

#Defining Remove url function

```
In [8]: def remove_urls(tweet):
        url_pattern = re.compile(r'https?://\S+|www\.\S+')
        return url_pattern.sub(r'', tweet)

data["tweet_w/o_url"] = data["tweets_w/o_space"].apply(lambda tweet: remove_urls(tweet))
```

#Removing Special Characters using Wordnet Lemmatizer

```
In [9]: import nltk
        nltk.download('wordnet')

[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\Sridhar\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!

Out[9]: True

In [10]: from nltk.stem import WordNetLemmatizer

data['tweet_lem'] = [''.join([WordNetLemmatizer().lemmatize(re.sub('[^A-Za-z]', ' ', tweet)) for tweet in lis])
                    for lis in data['tweet_w/o_url']]
```

#Tokenization

```
In [11]: data["tokenized_tweet"] = data["tweet_lem"].apply(lambda x: x.split())

In [12]: data.head()
```

Out[12]:

	index	count	hate_speech	offensive_language	neither	class	tweet	tweets_w/o_space	tweet_w/o_url	tweet_lem	tokenized_tweet
0	0	3	0	0	3	2	!!!! RT @mayasolovely: As a woman you shouldn't...	!!!! rt @mayasolovely: as a woman you shouldn't...	!!!! rt @mayasolovely: as a woman you shouldn't...	rt mayasolovely as a woman you shouldn t...	[rt, mayasolovely, as, a, woman, you, shouldn't...
1	1	3	0	3	0	1	!!!! RT @mleew17: boy dats cold...tyga dwn ba...	!!!! rt @mleewnumbr: boy dats cold...tyga dwn...	!!!! rt @mleewnumbr: boy dats cold...tyga dwn...	rt mleewnumbr boy dats cold tyga dwn...	[rt, mleewnumbr, boy, dats, cold, tyga, dwn, b...
2	2	3	0	3	0	1	!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...	!!!! rt @urkindofbrand dawg!!!! rt @numbrsb...	!!!! rt @urkindofbrand dawg!!!! rt @numbrsb...	rt urkindofbrand dawg rt numbrsb...	[rt, urkindofbrand, dawg, rt, numbrsbabynumbr...
3	3	3	0	2	1	1	!!!! RT @C_G_Anderson: @viva_based she lo...	!!!! rt @c_g_anderson: @viva_based she lo...	!!!! rt @c_g_anderson: @viva_based she lo...	rt c g anderson viva based she lo...	[rt, c, g, anderson, viva, based, she, look, l...
4	4	6	0	6	0	1	!!!! RT @ShenikaRoberts: The shit you...	!!!! rt @shenikaroberts: the shit you...	!!!! rt @shenikaroberts: the shit you...	rt shenikaroberts the shit you...	[rt, shenikaroberts, the, shit, you, hear, abo...

#Removing Stop Words

```
In [13]: import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\Sridhar\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

```
Out[13]: True
```

```
In [14]: stopwords = nltk.corpus.stopwords.words("english")
other_exclusions = ["#ff", "ff", "rt"]
stopwords.extend(other_exclusions)
stemmer = PorterStemmer()
```

```
In [15]: data["tweet_w/o_stop"] = data["tokenized_tweet"].apply(lambda x: [item for item in x if item not in stopwords])
data.head()
```

```
Out[15]:
```

	hate_speech	offensive_language	neither	class	tweet	tweets_w/o_space	tweet_w/o_url	tweet_lem	tokenized_tweet	tweet_w/o_stop
0		0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...	!!! rt @mayasolovely: as a woman you shouldn't...	!!! rt @mayasolovely: as a woman you shouldn't...	mayasolovely as a woman you shouldn t...	[rt, mayasolovely, as, a, woman, you, shouldn,...	[mayasolovely, woman, complain, cleaning, hous...
0		3	0	1	!!!! RT @mleew17: boy dats cold...tyga dwn ba...	!!!! rt @mleewnumbr: boy dats cold...tyga dwn...	!!!! rt @mleewnumbr: boy dats cold...tyga dwn...	rt mleewnumbr boy dats cold tyga dwn...	[rt, mleewnumbr, boy, dats, cold, tyga, dwn, b...	[mleewnumbr, boy, dats, cold, tyga, dwn, bad, ...
0		3	0	1	!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...	!!!! rt @urkindofbrand dawg!!!! rt @numbrsb...	!!!! rt @urkindofbrand dawg!!!! rt @numbrsb...	rt urkindofbrand dawg rt numbrsb...	[rt, urkindofbrand, dawg, rt, numbrsbabynumbrl...	[urkindofbrand, dawg, numbrsbabynumbrlife, eve...
0		2	1	1	!!!! RT @C_G_Anderson: @viva_based she lo...	!!!! rt @c_g_anderson: @viva_based she lo...	!!!! rt @c_g_anderson: @viva_based she lo...	rt c g anderson viva based she lo...	[rt, c, g, anderson, viva, based, she, look, l...	[c, g, anderson, viva, based, look, like, tranny]

#Stemming

```
In [16]: processed_tweet = data["tweet_w/o_stop"].apply(lambda x: [stemmer.stem(i) for i in x])
```

```
In [17]: for i in range(len(processed_tweet)):
processed_tweet[i] = ' '.join(processed_tweet[i])
tweets_p= processed_tweet
```

```
In [18]: data['processed_tweet'] = tweets_p
```

```
In [19]: data.head()
```

```
Out[19]:
```

	nsive_language	neither	class	tweet	tweets_w/o_space	tweet_w/o_url	tweet_lem	tokenized_tweet	tweet_w/o_stop	processed_tweet
0		3	2	!!! RT @mayasolovely: As a woman you shouldn't...	!!! rt @mayasolovely: as a woman you shouldn't...	!!! rt @mayasolovely: as a woman you shouldn't...	rt mayasolovely as a woman you shouldn t...	[rt, mayasolovely, as, a, woman, you, shouldn,...	[mayasolovely, woman, complain, cleaning, hous...	mayasolov woman complain clean hous amp man al...
3		0	1	!!!! RT @mleew17: boy dats cold...tyga dwn ba...	!!!! rt @mleewnumbr: boy dats cold...tyga dwn...	!!!! rt @mleewnumbr: boy dats cold...tyga dwn...	rt mleewnumbr boy dats cold tyga dwn...	[rt, mleewnumbr, boy, dats, cold, tyga, dwn, b...	[mleewnumbr, boy, dats, cold, tyga, dwn, bad, ...	mleewnumbr boy dat cold tyga dwn bad cuffin da...
3		0	1	!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...	!!!! rt @urkindofbrand dawg!!!! rt @numbrsb...	!!!! rt @urkindofbrand dawg!!!! rt @numbrsb...	rt urkindofbrand dawg rt numbrsb...	[rt, urkindofbrand, dawg, rt, numbrsbabynumbrl...	[urkindofbrand, dawg, numbrsbabynumbrlife, eve...	urkindofbrand dawg numbrsbabynumbrlif ever fuc...
2		1	1	!!!! RT @C_G_Anderson: @viva_based she lo...	!!!! rt @c_g_anderson: @viva_based she lo...	!!!! rt @c_g_anderson: @viva_based she lo...	rt c g anderson viva based she lo...	[rt, c, g, anderson, viva, based, she, look, l...	[c, g, anderson, viva, based, look, like, tranny]	c g anderson viva base look like tranni
6		0	1	!!!! RT @ShenikaRoberts: The shit you...	!!!! rt @shenikaroberts: the shit you...	!!!! rt @shenikaroberts: the shit you...	rt shenikaroberts the shit you...	[rt, shenikaroberts, the, shit, you, hear, abo...	[shenikaroberts, shit, hear, might, true, migh...	shenikarobert shit hear might true might faker...

#Displaying tweets and processed tweets

```
In [20]: data.tweet
```

```
Out[20]: 0      !!! RT @mayasolovely: As a woman you shouldn't...
1      !!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2      !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3      !!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4      !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

      ...
24778  you's a muthaf***in lie &#8220;@LifeAsKing: @2...
24779  you've gone and broke the wrong heart baby, an...
24780  young buck wanna eat!!.. dat nigguh like I ain...
24781           youu got wild bitches tellin you lies
24782  ~~Ruffled | Ntac Eileen Dahlia - Beautiful col...
Name: tweet, Length: 24783, dtype: object
```

```
In [21]: data.processed_tweet
```

```
Out[21]: 0      mayasolov woman complain clean hous amp man al...
1      mleewnumbr boy dat cold tyga dwn bad cuffin da...
2      urkindofbrand dawg numbrsbabynumbrlif ever fuc...
3      c g anderson viva base look like tranni
4      shenikarobert shit hear might true might faker...

      ...
24778  muthaf lie numbr lifeask numbr pearl corey ema...
24779  gone broke wrong heart babi drove redneck crazi
24780  young buck wanna eat dat nigguh like aint fuck...
24781           youu got wild bitch tellin lie
24782  ruffl ntac eileen dahlia beauti color combin p...
Name: processed_tweet, Length: 24783, dtype: object
```

#One Hot Encoding of Target Label

```
In [23]: labels = data['class']
```

```
In [24]: from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse=False)

labels = encoder.fit_transform(np.array(labels).reshape(-1, 1))
```

MODULE 3

#Feature Extracted Using Word Level One Hot Encoding (F1)

```
In [25]: from nltk.tokenize.treebank import TreebankWordDetokenizer
def detokenize(text):
    return TreebankWordDetokenizer().detokenize(text)
```

```
In [26]: data1 = []

for i in range(len(tweetstop)):
    data1.append(detokenize(tweetstop[i]))
print(data1[:5])

['mayasolovely woman complain cleaning house amp man always take trash', 'mleewnumbr boy dats cold tyga dwn bad cuffin dat hoe
numbrst place', 'urkindofbrand dawg numbrsbabynumbrlife ever fuck bitch start cry confused shit', 'c g anderson viva based look
like tranny', 'shenikaroberts shit hear might true might faker bitch told ya numbr']
```

```
In [27]: data1 = np.array(data1)
```

```
In [28]: max_words = 5000
max_len = 200

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(data1)
sequences = tokenizer.texts_to_sequences(data1)
tweets_in = pad_sequences(sequences, maxlen=max_len)
print(tweets_in)

[[ 0  0  0  ...  83  76  15]
 [ 0  0  0  ...   7 605 414]
 [ 0  0  0  ... 470 900  12]
 ...
 [ 0  0  0  ...  96  95 246]
 [ 0  0  ...   3 1826 1247]
 [ 0  0  0  ...  94  17  48]]
```

#Feature Extracted Using TFIDF Technique (F2)

```
In [35]: tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 2),max_df=0.75, min_df=5, max_features=1000)
tfidf = tfidf_vectorizer.fit_transform(data['processed_tweet'] ).toarray()
tfidf
```

```
Out[35]: array([[0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 ...,
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.],
 [0., 0., 0., ..., 0., 0., 0.]])
```

```
In [36]: tfidf.shape
```

```
Out[36]: (24783, 1000)
```

#Feature Extracted Using Bag of Words Technique [COUNT Vectorizer] (F3)

```
In [42]: bow_vectorizer = CountVectorizer( max_features=100, stop_words='english', ngram_range=(2,2))
bagofwords=bow_vectorizer.fit_transform(data['processed_tweet'] ).toarray()
bagofwords
```

```
Out[42]: array([[0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 ...,
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0],
 [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

```
In [43]: bagofwords.shape
```

```
Out[43]: (24783, 100)
```

#Combining all the Three Features [F1 + F2 + F3]

```
In [48]: modelling_features = np.concatenate([tfidf,bagofwords,tweets_in],axis=1)
modelling_features.shape
```

```
Out[48]: (24783, 1300)
```

MODULE 4:

#Building a LSTM Model and Saving it

```
In [30]: model = Sequential()
model.add(layers.Embedding(max_words, 20))
model.add(layers.LSTM(15,dropout=0.5))
model.add(layers.Flatten())
model.add(layers.Dropout(0.25))

model.add(layers.Dense(512,activation='relu'))
model.add(layers.Dense(3,activation='softmax'))

model.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['accuracy'])

In [31]: checkpoint = ModelCheckpoint("best_model.hdf5", monitor='val_accuracy', verbose=1,save_best_only=True, mode='auto',
period=1,save_weights_only=False)

WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify the frequency in number of batches seen.
```

The above LSTM Model is trained for four different features namely:

- Word Level One Hot Encoding (F1)
- TFIDF Vectorizer (F2)
- Bag of Words [COUNT Vectorizer] (F3)
- Combining all three features (F1 + F2 + F3)

TRAIL 1:

#Training the Model using Word level One Hot Encoding (F1) Features

#Initializing X_train, X_test, y_train & y_test for F1

```
In [29]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(tweets_in,labels, random_state=0)
print (len(X_train),len(X_test),len(y_train),len(y_test))
```

```
18587 6196 18587 6196
```

```
In [32]: history1 = model.fit(X_train, y_train, steps_per_epoch=20, epochs=10, validation_data=(X_test, y_test), callbacks=[checkpoint])

Epoch 1/10
20/20 [=====] - ETA: 0s - loss: 0.9616 - accuracy: 0.7525
Epoch 00001: val_accuracy improved from -inf to 0.77469, saving model to best_model.hdf5
20/20 [=====] - 9s 429ms/step - loss: 0.9616 - accuracy: 0.7525 - val_loss: 0.7381 - val_accuracy: 0.7747
Epoch 2/10
20/20 [=====] - ETA: 0s - loss: 0.6738 - accuracy: 0.7742
Epoch 00002: val_accuracy did not improve from 0.77469
20/20 [=====] - 9s 460ms/step - loss: 0.6738 - accuracy: 0.7742 - val_loss: 0.6357 - val_accuracy: 0.7747
Epoch 3/10
20/20 [=====] - ETA: 0s - loss: 0.6209 - accuracy: 0.7742
Epoch 00003: val_accuracy did not improve from 0.77469
20/20 [=====] - 7s 367ms/step - loss: 0.6209 - accuracy: 0.7742 - val_loss: 0.5986 - val_accuracy: 0.7747
Epoch 4/10
20/20 [=====] - ETA: 0s - loss: 0.5600 - accuracy: 0.7743
Epoch 00004: val_accuracy improved from 0.77469 to 0.77518, saving model to best_model.hdf5
20/20 [=====] - 8s 392ms/step - loss: 0.5600 - accuracy: 0.7743 - val_loss: 0.5207 - val_accuracy: 0.7752
Epoch 5/10
20/20 [=====] - ETA: 0s - loss: 0.4621 - accuracy: 0.8069
Epoch 00005: val_accuracy improved from 0.77518 to 0.84135, saving model to best_model.hdf5
20/20 [=====] - 8s 384ms/step - loss: 0.4621 - accuracy: 0.8069 - val_loss: 0.4315 - val_accuracy: 0.8413
Epoch 6/10
20/20 [=====] - ETA: 0s - loss: 0.3727 - accuracy: 0.8764
Epoch 00006: val_accuracy improved from 0.84135 to 0.86701, saving model to best_model.hdf5
20/20 [=====] - 8s 423ms/step - loss: 0.3727 - accuracy: 0.8764 - val_loss: 0.3836 - val_accuracy: 0.8670
Epoch 7/10
20/20 [=====] - ETA: 0s - loss: 0.3271 - accuracy: 0.8917
Epoch 00007: val_accuracy improved from 0.86701 to 0.87637, saving model to best_model.hdf5
20/20 [=====] - 8s 379ms/step - loss: 0.3271 - accuracy: 0.8917 - val_loss: 0.3678 - val_accuracy: 0.8764
Epoch 8/10
20/20 [=====] - ETA: 0s - loss: 0.2989 - accuracy: 0.9028
Epoch 00008: val_accuracy improved from 0.87637 to 0.88073, saving model to best_model.hdf5
20/20 [=====] - 9s 430ms/step - loss: 0.2989 - accuracy: 0.9028 - val_loss: 0.3642 - val_accuracy: 0.8807
Epoch 9/10
20/20 [=====] - ETA: 0s - loss: 0.2771 - accuracy: 0.9085
Epoch 00009: val_accuracy did not improve from 0.88073
20/20 [=====] - 8s 381ms/step - loss: 0.2771 - accuracy: 0.9085 - val_loss: 0.3873 - val_accuracy: 0.8767
Epoch 10/10
20/20 [=====] - ETA: 0s - loss: 0.2692 - accuracy: 0.9120
Epoch 00010: val_accuracy improved from 0.88073 to 0.88202, saving model to best_model.hdf5
20/20 [=====] - 8s 420ms/step - loss: 0.2692 - accuracy: 0.9120 - val_loss: 0.3736 - val_accuracy: 0.8820
```

TRAIL 2:

#Training the Model using **TFIDF (F2) Features**

#Initializing X_train, X_test, y_train & y_test for F2

```
In [38]: from sklearn.model_selection import train_test_split
X = tfidf
X_train1, X_test1, y_train1, y_test1 = train_test_split(X, labels, random_state=42, test_size=0.2)
```

```
In [39]: history2 = model.fit(X_train1, y_train1, steps_per_epoch=20, epochs=10, validation_data=(X_test1, y_test1),
                             callbacks=[checkpoint])
```

```
Epoch 1/10
20/20 [=====] - ETA: 0s - loss: 0.7280 - accuracy: 0.7710
Epoch 00001: val_accuracy did not improve from 0.88202
20/20 [=====] - 64s 3s/step - loss: 0.7280 - accuracy: 0.7710 - val_loss: 0.6751 - val_accuracy: 0.773
0
Epoch 2/10
20/20 [=====] - ETA: 0s - loss: 0.6696 - accuracy: 0.7746
Epoch 00002: val_accuracy did not improve from 0.88202
20/20 [=====] - 63s 3s/step - loss: 0.6696 - accuracy: 0.7746 - val_loss: 0.6673 - val_accuracy: 0.773
0
Epoch 3/10
20/20 [=====] - ETA: 0s - loss: 0.6660 - accuracy: 0.7746
Epoch 00003: val_accuracy did not improve from 0.88202
20/20 [=====] - 68s 3s/step - loss: 0.6660 - accuracy: 0.7746 - val_loss: 0.6651 - val_accuracy: 0.773
0
Epoch 4/10
20/20 [=====] - ETA: 0s - loss: 0.6643 - accuracy: 0.7746
Epoch 00004: val_accuracy did not improve from 0.88202
20/20 [=====] - 72s 4s/step - loss: 0.6643 - accuracy: 0.7746 - val_loss: 0.6651 - val_accuracy: 0.773
0
Epoch 5/10
20/20 [=====] - ETA: 0s - loss: 0.6646 - accuracy: 0.7746
Epoch 00005: val_accuracy did not improve from 0.88202
20/20 [=====] - 72s 4s/step - loss: 0.6646 - accuracy: 0.7746 - val_loss: 0.6652 - val_accuracy: 0.773
0
Epoch 6/10
20/20 [=====] - ETA: 0s - loss: 0.6641 - accuracy: 0.7746
Epoch 00006: val_accuracy did not improve from 0.88202
20/20 [=====] - 72s 4s/step - loss: 0.6641 - accuracy: 0.7746 - val_loss: 0.6651 - val_accuracy: 0.773
0
Epoch 7/10
20/20 [=====] - ETA: 0s - loss: 0.6637 - accuracy: 0.7746
Epoch 00007: val_accuracy did not improve from 0.88202
20/20 [=====] - 73s 4s/step - loss: 0.6637 - accuracy: 0.7746 - val_loss: 0.6651 - val_accuracy: 0.773
0
Epoch 8/10
20/20 [=====] - ETA: 0s - loss: 0.6631 - accuracy: 0.7746
Epoch 00008: val_accuracy did not improve from 0.88202
20/20 [=====] - 74s 4s/step - loss: 0.6631 - accuracy: 0.7746 - val_loss: 0.6652 - val_accuracy: 0.773
0
Epoch 9/10
20/20 [=====] - ETA: 0s - loss: 0.6632 - accuracy: 0.7746
Epoch 00009: val_accuracy did not improve from 0.88202
20/20 [=====] - 73s 4s/step - loss: 0.6632 - accuracy: 0.7746 - val_loss: 0.6651 - val_accuracy: 0.773
0
Epoch 10/10
20/20 [=====] - ETA: 0s - loss: 0.6632 - accuracy: 0.7746
Epoch 00010: val_accuracy did not improve from 0.88202
20/20 [=====] - 72s 4s/step - loss: 0.6632 - accuracy: 0.7746 - val_loss: 0.6652 - val_accuracy: 0.773
0
```

TRAIL 3:

#Training the Model using **Bag of Words (F3) Features**

#Initializing X_train, X_test, y_train & y_test for F3

```
In [44]: X = bagofwords
          X_train2, X_test2, y_train2, y_test2 = train_test_split(X, labels, random_state=42, test_size=0.2)
```

```
In [45]: history3 = model.fit(X_train2, y_train2, epochs=10, steps_per_epoch=20, validation_data=(X_test2, y_test2),
                             callbacks=[checkpoint])
```

```
Epoch 1/10
20/20 [=====] - ETA: 0s - loss: 0.6631 - accuracy: 0.7746
Epoch 00001: val_accuracy did not improve from 0.88202
20/20 [=====] - 5s 251ms/step - loss: 0.6631 - accuracy: 0.7746 - val_loss: 0.6654 - val_accuracy: 0.7730
Epoch 2/10
20/20 [=====] - ETA: 0s - loss: 0.6631 - accuracy: 0.7746
Epoch 00002: val_accuracy did not improve from 0.88202
20/20 [=====] - 4s 217ms/step - loss: 0.6631 - accuracy: 0.7746 - val_loss: 0.6654 - val_accuracy: 0.7730
Epoch 3/10
20/20 [=====] - ETA: 0s - loss: 0.6629 - accuracy: 0.7746
Epoch 00003: val_accuracy did not improve from 0.88202
20/20 [=====] - 4s 213ms/step - loss: 0.6629 - accuracy: 0.7746 - val_loss: 0.6655 - val_accuracy: 0.7730
Epoch 4/10
20/20 [=====] - ETA: 0s - loss: 0.6629 - accuracy: 0.7746
Epoch 00004: val_accuracy did not improve from 0.88202
20/20 [=====] - 5s 236ms/step - loss: 0.6629 - accuracy: 0.7746 - val_loss: 0.6654 - val_accuracy: 0.7730
Epoch 5/10
20/20 [=====] - ETA: 0s - loss: 0.6628 - accuracy: 0.7746
Epoch 00005: val_accuracy did not improve from 0.88202
20/20 [=====] - 5s 229ms/step - loss: 0.6628 - accuracy: 0.7746 - val_loss: 0.6652 - val_accuracy: 0.7730
Epoch 6/10
20/20 [=====] - ETA: 0s - loss: 0.6625 - accuracy: 0.7746
Epoch 00006: val_accuracy did not improve from 0.88202
20/20 [=====] - 5s 227ms/step - loss: 0.6625 - accuracy: 0.7746 - val_loss: 0.6651 - val_accuracy: 0.7730

Epoch 7/10
20/20 [=====] - ETA: 0s - loss: 0.6617 - accuracy: 0.7746
Epoch 00007: val_accuracy did not improve from 0.88202
20/20 [=====] - 5s 272ms/step - loss: 0.6617 - accuracy: 0.7746 - val_loss: 0.6652 - val_accuracy: 0.7730
Epoch 8/10
20/20 [=====] - ETA: 0s - loss: 0.6616 - accuracy: 0.7746
Epoch 00008: val_accuracy did not improve from 0.88202
20/20 [=====] - 6s 300ms/step - loss: 0.6616 - accuracy: 0.7746 - val_loss: 0.6651 - val_accuracy: 0.7730
Epoch 9/10
20/20 [=====] - ETA: 0s - loss: 0.6616 - accuracy: 0.7746
Epoch 00009: val_accuracy did not improve from 0.88202
20/20 [=====] - 5s 274ms/step - loss: 0.6616 - accuracy: 0.7746 - val_loss: 0.6652 - val_accuracy: 0.7730
Epoch 10/10
20/20 [=====] - ETA: 0s - loss: 0.6613 - accuracy: 0.7746
Epoch 00010: val_accuracy did not improve from 0.88202
20/20 [=====] - 6s 315ms/step - loss: 0.6613 - accuracy: 0.7746 - val_loss: 0.6652 - val_accuracy: 0.7730
```

TRAIL 4:

#Training the Model using (F1 + F2 + F3) Features

#Initializing X_train, X_test, y_train & y_test for F1 + F2 + F3

```
In [49]: X = modelling_features
X_train3, X_test3, y_train3, y_test3 = train_test_split(X, labels, random_state=42, test_size=0.2)
```

```
In [50]: history4= model.fit(X_train3, y_train3, epochs=10, steps_per_epoch=20, validation_data=(X_test3, y_test3),
                           callbacks=[checkpoint])
```

```
Epoch 1/10
20/20 [=====] - ETA: 0s - loss: 0.3324 - accuracy: 0.8962
Epoch 00001: val_accuracy improved from 0.88202 to 0.90418, saving model to best_model.hdf5
20/20 [=====] - 111s 6s/step - loss: 0.3324 - accuracy: 0.8962 - val_loss: 0.3003 - val_accuracy: 0.9042
Epoch 2/10
20/20 [=====] - ETA: 0s - loss: 0.2740 - accuracy: 0.9112
Epoch 00002: val_accuracy did not improve from 0.90418
20/20 [=====] - 114s 6s/step - loss: 0.2740 - accuracy: 0.9112 - val_loss: 0.2939 - val_accuracy: 0.9034
Epoch 3/10
20/20 [=====] - ETA: 0s - loss: 0.2502 - accuracy: 0.9174
Epoch 00003: val_accuracy did not improve from 0.90418
20/20 [=====] - 116s 6s/step - loss: 0.2502 - accuracy: 0.9174 - val_loss: 0.2961 - val_accuracy: 0.9036
Epoch 4/10
20/20 [=====] - ETA: 0s - loss: 0.2337 - accuracy: 0.9223
Epoch 00004: val_accuracy did not improve from 0.90418
20/20 [=====] - 117s 6s/step - loss: 0.2337 - accuracy: 0.9223 - val_loss: 0.2931 - val_accuracy: 0.9001
Epoch 5/10
20/20 [=====] - ETA: 0s - loss: 0.2171 - accuracy: 0.9271
Epoch 00005: val_accuracy did not improve from 0.90418
20/20 [=====] - 122s 6s/step - loss: 0.2171 - accuracy: 0.9271 - val_loss: 0.2966 - val_accuracy: 0.8995
Epoch 6/10
20/20 [=====] - ETA: 0s - loss: 0.2010 - accuracy: 0.9320
Epoch 00006: val_accuracy did not improve from 0.90418
20/20 [=====] - 119s 6s/step - loss: 0.2010 - accuracy: 0.9320 - val_loss: 0.2953 - val_accuracy: 0.9034
Epoch 7/10
20/20 [=====] - ETA: 0s - loss: 0.1885 - accuracy: 0.9348
Epoch 00007: val_accuracy did not improve from 0.90418
20/20 [=====] - 119s 6s/step - loss: 0.1885 - accuracy: 0.9348 - val_loss: 0.3046 - val_accuracy: 0.8957
Epoch 8/10
20/20 [=====] - ETA: 0s - loss: 0.1790 - accuracy: 0.9380
Epoch 00008: val_accuracy did not improve from 0.90418
20/20 [=====] - 119s 6s/step - loss: 0.1790 - accuracy: 0.9380 - val_loss: 0.3067 - val_accuracy: 0.9028
Epoch 9/10
20/20 [=====] - ETA: 0s - loss: 0.1695 - accuracy: 0.9398
Epoch 00009: val_accuracy did not improve from 0.90418
20/20 [=====] - 118s 6s/step - loss: 0.1695 - accuracy: 0.9398 - val_loss: 0.3173 - val_accuracy: 0.8989
Epoch 10/10
20/20 [=====] - ETA: 0s - loss: 0.1641 - accuracy: 0.9433
Epoch 00010: val_accuracy did not improve from 0.90418
20/20 [=====] - 120s 6s/step - loss: 0.1641 - accuracy: 0.9433 - val_loss: 0.3369 - val_accuracy: 0.8943
```

#LSTM Model Summary

```
In [61]: model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
embedding (Embedding)	(None, None, 20)	100000
lstm (LSTM)	(None, 15)	2160
flatten (Flatten)	(None, 15)	0
dropout (Dropout)	(None, 15)	0
dense (Dense)	(None, 512)	8192
dense_1 (Dense)	(None, 3)	1539
=====		
Total params: 111,891		
Trainable params: 111,891		
Non-trainable params: 0		

MODULE 5:

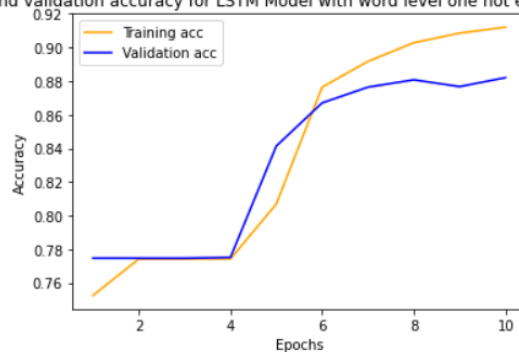
#Accuracy and Loss curve for LSTM Model Trained using Feature F1 [TRAIL 1]

```
In [33]: import matplotlib.pyplot as plt
acc = history1.history['accuracy']
val_acc = history1.history['val_accuracy']

epochs = range(1, 11)

plt.plot(epochs, acc, 'orange', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy for LSTM Model with word level one hot encoding as Feature')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()
plt.show()
```

Training and validation accuracy for LSTM Model with word level one hot encoding as Feature

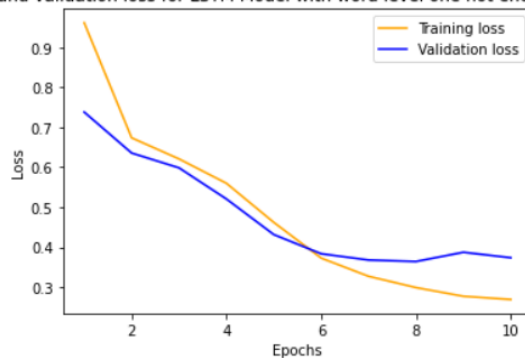


```
In [34]: loss = history1.history['loss']
val_loss = history1.history['val_loss']

epochs = range(1, 11)

plt.plot(epochs, loss, 'orange', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss for LSTM Model with word level one hot encoding as Feature')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Training and validation loss for LSTM Model with word level one hot encoding as Feature

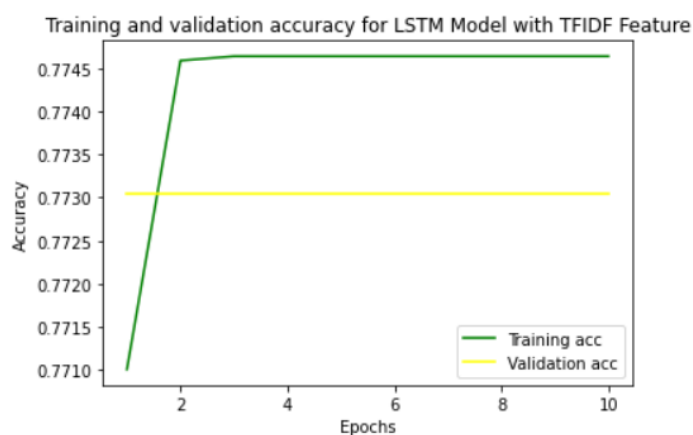


#Accuracy and Loss curve for LSTM Model Trained using Feature F2 [TRAIL 2]

```
In [40]: import matplotlib.pyplot as plt
acc = history2.history['accuracy']
val_acc = history2.history['val_accuracy']

epochs = range(1, 11)

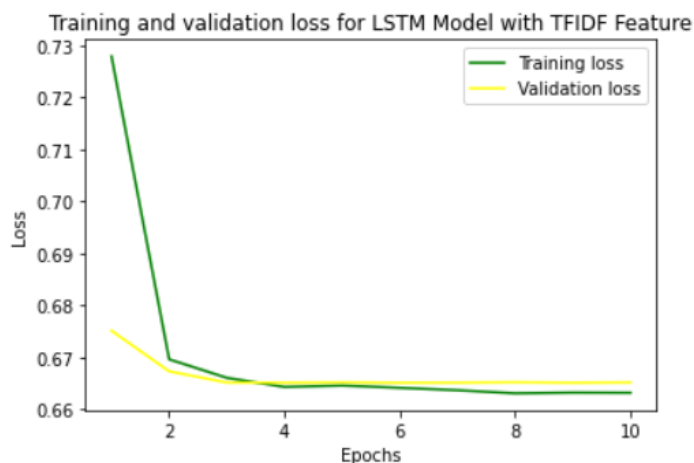
plt.plot(epochs, acc, 'green', label='Training acc')
plt.plot(epochs, val_acc, 'yellow', label='Validation acc')
plt.title('Training and validation accuracy for LSTM Model with TFIDF Feature')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()
plt.show()
```



```
In [41]: loss = history2.history['loss']
val_loss = history2.history['val_loss']

epochs = range(1, 11)

plt.plot(epochs, loss, 'green', label='Training loss')
plt.plot(epochs, val_loss, 'yellow', label='Validation loss')
plt.title('Training and validation loss for LSTM Model with TFIDF Feature')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

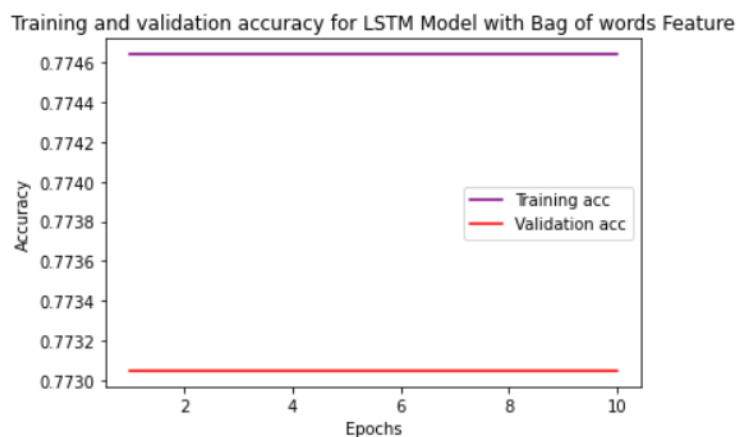


#Accuracy and Loss curve for LSTM Model Trained using Feature F3 [TRAIL 3]

```
In [46]: import matplotlib.pyplot as plt
acc = history3.history['accuracy']
val_acc = history3.history['val_accuracy']

epochs = range(1, 11)

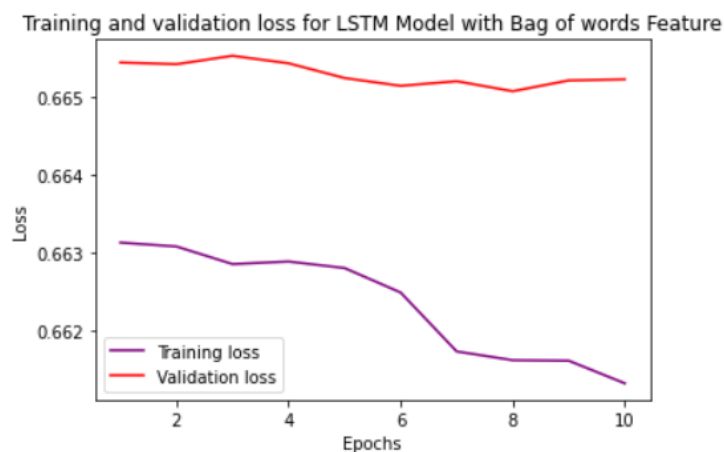
plt.plot(epochs, acc, 'purple', label='Training acc')
plt.plot(epochs, val_acc, 'red', label='Validation acc')
plt.title('Training and validation accuracy for LSTM Model with Bag of words Feature')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()
plt.show()
```



```
In [47]: loss = history3.history['loss']
val_loss = history3.history['val_loss']

epochs = range(1, 11)

plt.plot(epochs, loss, 'purple', label='Training loss')
plt.plot(epochs, val_loss, 'red', label='Validation loss')
plt.title('Training and validation loss for LSTM Model with Bag of words Feature')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



#Accuracy and Loss curve for LSTM Model Trained using Feature F1+F2+F3

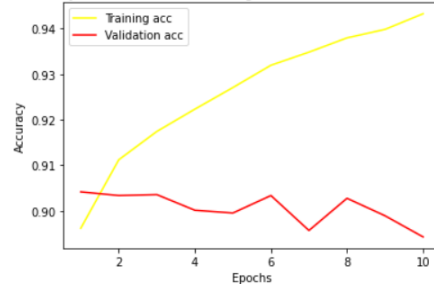
[TRAIL 4]

```
In [51]: acc = history4.history['accuracy']
val_acc = history4.history['val_accuracy']

epochs = range(1, 11)

plt.plot(epochs, acc, 'yellow', label='Training acc')
plt.plot(epochs, val_acc, 'red', label='Validation acc')
plt.title('Training and validation accuracy for LSTM Model with [Bag of words + TFIDF + Word level one hot encoding] Feature')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.figure()
plt.show()
```

Training and validation accuracy for LSTM Model with [Bag of words + TFIDF + Word level one hot encoding] Feature

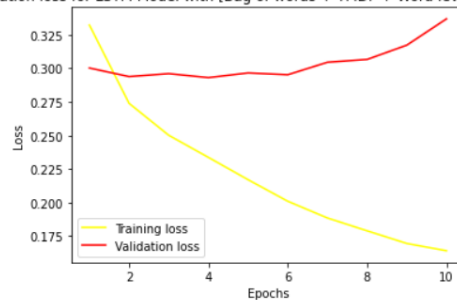


```
In [52]: loss = history4.history['loss']
val_loss = history4.history['val_loss']

epochs = range(1, 11)

plt.plot(epochs, loss, 'yellow', label='Training loss')
plt.plot(epochs, val_loss, 'red', label='Validation loss')
plt.title('Training and validation loss for LSTM Model with [Bag of words + TFIDF + Word level one hot encoding] Feature')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.figure()
plt.show()
```

Training and validation loss for LSTM Model with [Bag of words + TFIDF + Word level one hot encoding] Feature



#Printing the Test Accuracy and Test Loss for Trail 1,2,3 & 4

```
In [53]: score1 = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score1[0])
print('Test accuracy:', score1[1])
```

```
Test loss: 0.20353253185749054
Test accuracy: 0.9351194500923157
```

```
In [54]: score2 = model.evaluate(X_test1, y_test1, verbose=0)
print('Test loss:', score2[0])
print('Test accuracy:', score2[1])
```

```
Test loss: 0.6697127819061279
Test accuracy: 0.7730482220649719
```

```
In [55]: score3 = model.evaluate(X_test2, y_test2, verbose=0)
print('Test loss:', score2[0])
print('Test accuracy:', score2[1])
```

```
Test loss: 0.6697127819061279
Test accuracy: 0.7730482220649719
```

```
In [76]: score4 = model.evaluate(X_test3, y_test3, verbose=0)
print('Test loss:', score4[0])
print('Test accuracy:', score4[1])
```

```
Test loss: 0.33693215250968933
Test accuracy: 0.8942909240722656
```

INFERENCE:

The **TEST ACCURACY SCORE** is **high** when the LSTM Model is Trained using Features like

- Word Level One Hot Encoding (F1)
- Word Level One Hot Encoding + TFIDF + Bag of Words (F1 + F2 + F3)

Where as the **TEST ACCURACY SCORE** is **low** for the LSTM Model Trained using features like:

- TFIDF (F2)
- Bag of Words (F3)

#Printing Confusion Matrix for TRAIL 1:

```
In [57]: from sklearn.metrics import confusion_matrix
import seaborn as sns
fig = plt.figure(figsize=(3, 3))

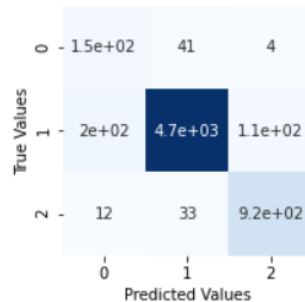
y_preds = model.predict(X_test)

Y_pred = np.argmax(y_preds, 1)
Y_test = np.argmax(y_test, 1)

mat = confusion_matrix(Y_test, Y_pred)

sns.heatmap(mat.T, square=True, annot=True, cbar=False, cmap=plt.cm.Blues)
plt.title('Confusion Matrix for LSTM Model with word level one hot encoding as Feature ')
plt.xlabel('Predicted Values')
plt.ylabel('True Values');
plt.show();
```

Confusion Matrix for LSTM Model with word level one hot encoding as Feature



#Printing Confusion Matrix for TRAIL 2:

```
In [58]: fig = plt.figure(figsize=(3, 3))

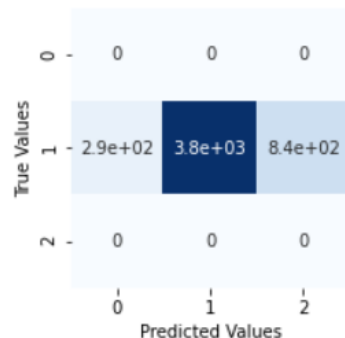
y_preds = model.predict(X_test1)

Y_pred = np.argmax(y_preds, 1)
Y_test = np.argmax(y_test1, 1)

mat = confusion_matrix(Y_test, Y_pred)

sns.heatmap(mat.T, square=True, annot=True, cbar=False, cmap=plt.cm.Blues)
plt.title('Confusion Matrix for LSTM Model with TFIDF Feature')
plt.xlabel('Predicted Values')
plt.ylabel('True Values');
plt.show();
```

Confusion Matrix for LSTM Model with TFIDF Feature



#Printing Confusion Matrix for TRAIL 3:

```
In [59]: fig = plt.figure(figsize=(3, 3))

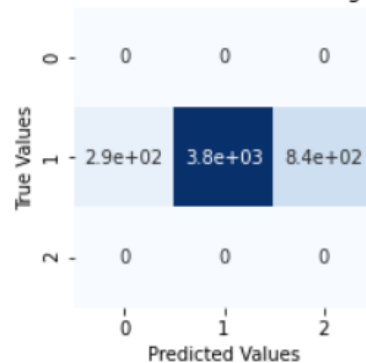
y_preds = model.predict(X_test2)

Y_pred = np.argmax(y_preds, 1)
Y_test = np.argmax(y_test2, 1)

mat = confusion_matrix(Y_test, Y_pred)

sns.heatmap(mat.T, square=True, annot=True, cbar=False, cmap=plt.cm.Blues)
plt.title('Confusion Matrix for LSTM Model with Bag of words Feature')
plt.xlabel('Predicted Values')
plt.ylabel('True Values');
plt.show();
```

Confusion Matrix for LSTM Model with Bag of words Feature



#Printing Confusion Matrix for TRAIL 4:

```
In [60]: fig = plt.figure(figsize=(3, 3))

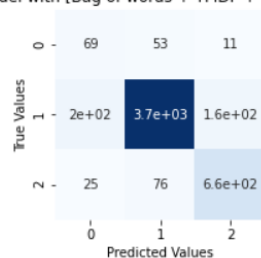
y_preds = model.predict(X_test3)

Y_pred = np.argmax(y_preds, 1)
Y_test = np.argmax(y_test3, 1)

mat = confusion_matrix(Y_test, Y_pred)

sns.heatmap(mat.T, square=True, annot=True, cbar=False, cmap=plt.cm.Blues)
plt.title('Confusion Matrix for LSTM Model with [Bag of words + TFIDF + Word level one hot encoding] Feature')
plt.xlabel('Predicted Values')
plt.ylabel('True Values');
plt.show();
```

Confusion Matrix for LSTM Model with [Bag of words + TFIDF + Word level one hot encoding] Feature



MODULE 6:

#Predicting Output for random input

```
In [63]: import keras
best_model = keras.models.load_model("best_model.hdf5")

In [64]: sentiment = ['Hatespeech', 'Offensivelanguage', 'Neither']

In [65]: sequence = tokenizer.texts_to_sequences(['this experience has been the worst , want my money back'])
test = pad_sequences(sequence, maxlen=max_len)
sentiment[np.argmax(best_model.predict(test), decimals=0).argmax(axis=1)[0]]

Out[65]: 'Hatespeech'

In [66]: sequence = tokenizer.texts_to_sequences(['as a woman you should not complain about cleaning up your house'])
test = pad_sequences(sequence, maxlen=max_len)
sentiment[np.argmax(best_model.predict(test), decimals=0).argmax(axis=1)[0]]

Out[66]: 'Offensivelanguage'

In [67]: sequence = tokenizer.texts_to_sequences(['When twitter rappers dm me their trash links'])
test = pad_sequences(sequence, maxlen=max_len)
sentiment[np.argmax(best_model.predict(test), decimals=0).argmax(axis=1)[0]]

Out[67]: 'Neither'
```

RESULT:

The LSTM Model is trained using four different features and accuracy for the models with different features is obtained

CONCLUSION:

A deep learning approach like LSTM (Long Short-Term Memory) Model have been used for creating a model for detecting hate speech and offensive language detection. Proposed model is trained and tested with different features and it achieved good results compared to its simplicity.

REFERENCE:

- <https://github.com/sergiovirahonda/TweetsSentimentAnalysis/blob/main/TweetsSentimentPredictions.ipynb>
- https://github.com/Sachin-Jain-98/Detection-And-Classification-Of-Hate-Speech-In-Social-Media-Using-Python/blob/master/Hate_speech_detection_Final_code.ipynb