DSCI-552 Section 32415D-Spring 2021

# Text Classification

**Ananya Sharma**
Department of Computer Science, University of Southern California,
Los Angeles, California 90089, USA

## Abstract:

In this report I first describe the kinds of data we have, how to ensure it's all in compliance and can gel along with each other. Then I start by testing a baseline Naive Bayes model. Then by measuring its performance metrics namely - Precision, recall, FScore, Accuracy. Then, I compare this model against 2 other models, one which uses a neural network and the other neural network with different activation functions and layers. Thus, finally deciding which model is better by fine tuning the parameters and eventually testing a new dataset to predict labels. Further, data augmentation is also done and hence it's found that although Naive Bayes is a good classifier for multi-class classification, neural nets work better.

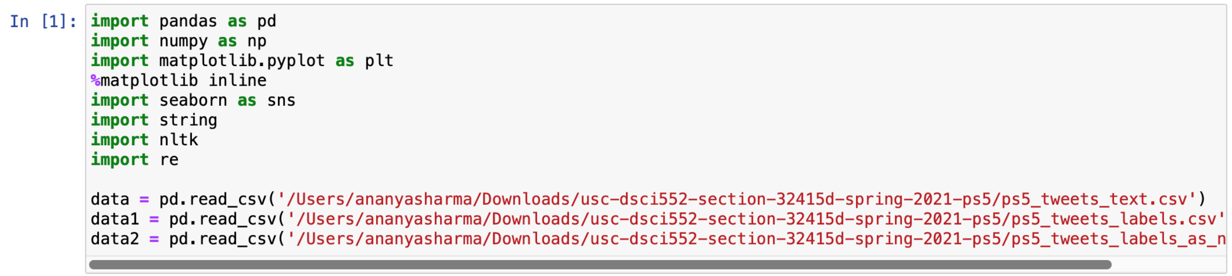
## Introduction :

For this task, tweets have been gathered and  have been classified as
0- Extremely Negative
1- Negative
2- Neutral
3- Positive
4- Extremely Positive

By training my model, these datasets just need to be run on it to get predictions.

## Model Selection and Evaluation:

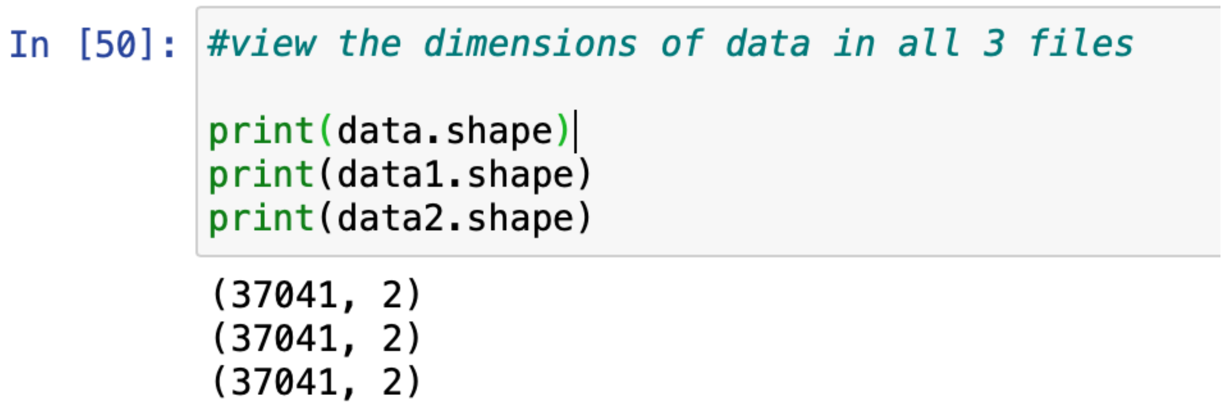
I started by importing the libraries required and the data ( 3 csv files storing the tweets, their labels and corresponding labels as texts)



```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        import seaborn as sns
        import string
        import nltk
        import re

        data = pd.read_csv('/Users/ananyasharma/Downloads/usc-dsci552-section-32415d-spring-2021-ps5/ps5_tweets_text.csv')
        data1 = pd.read_csv('/Users/ananyasharma/Downloads/usc-dsci552-section-32415d-spring-2021-ps5/ps5_tweets_labels.csv'
        data2 = pd.read_csv('/Users/ananyasharma/Downloads/usc-dsci552-section-32415d-spring-2021-ps5/ps5_tweets_labels_as_n
```

FIG 1 : How to load data

Then, to know what kind of data we have, its dimensions etc.

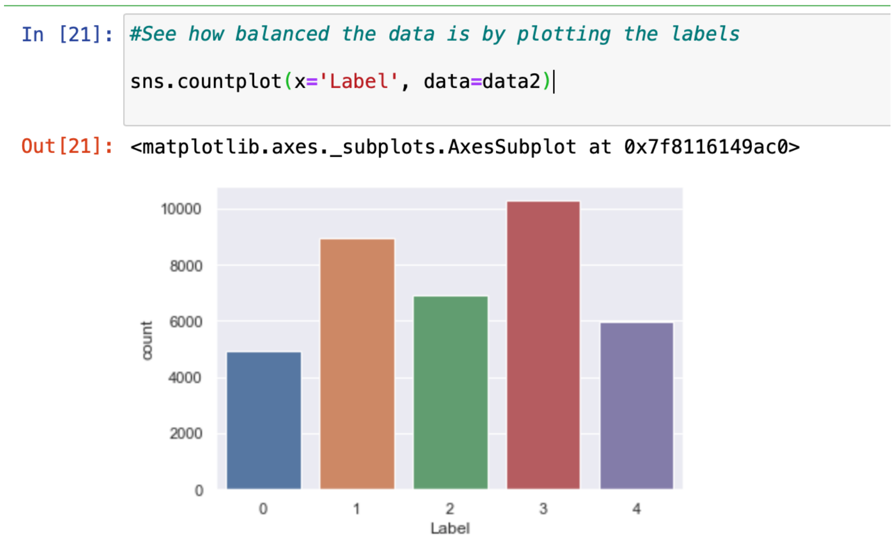


```
In [50]: #view the dimensions of data in all 3 files

         print(data.shape)
         print(data1.shape)
         print(data2.shape)

         (37041, 2)
         (37041, 2)
         (37041, 2)
```

FIG 2 : What dimensions of data

To proceed further, we first need to merge all this data into a single dataframe. Since these are tweets, a lot of preprocessing of the data will be required.

Now, let's see the labels and their corresponding aggregate tweets present in the dataset.

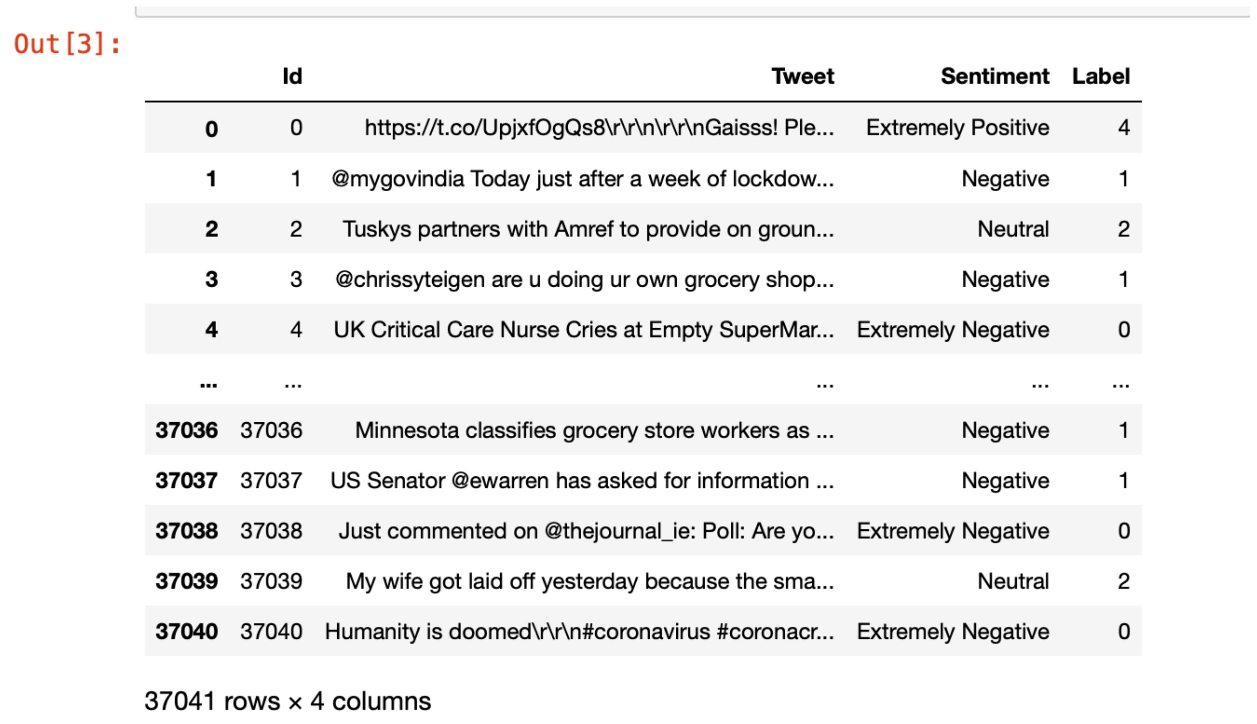


```
In [21]: #See how balanced the data is by plotting the labels

         sns.countplot(x='Label', data=data2)

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8116149ac0>
```

FIG 3 : Visualising the labels and their aggregate tweets in the dataset

From the above figure, one can see that the dataset is not at all balanced and infact label 3 is of the majority and label 0 is of minority.

Now, first we need to get consolidated data. It is as follows :



Out[3]:

| | Id | Tweet | Sentiment | Label |
|---|---|---|---|---|
| **0** | 0 | https://t.co/UpjxfOgQs8\r\r\n\r\r\nGaisss! Ple... | Extremely Positive | 4 |
| **1** | 1 | @mygovindia Today just after a week of lockdow... | Negative | 1 |
| **2** | 2 | Tuskys partners with Amref to provide on groun... | Neutral | 2 |
| **3** | 3 | @chrissyteigen are u doing ur own grocery shop... | Negative | 1 |
| **4** | 4 | UK Critical Care Nurse Cries at Empty SuperMar... | Extremely Negative | 0 |
| **...** | ... | ... | ... | ... |
| **37036** | 37036 | Minnesota classifies grocery store workers as ... | Negative | 1 |
| **37037** | 37037 | US Senator @ewarren has asked for information ... | Negative | 1 |
| **37038** | 37038 | Just commented on @thejournal_ie: Poll: Are yo... | Extremely Negative | 0 |
| **37039** | 37039 | My wife got laid off yesterday because the sma... | Neutral | 2 |
| **37040** | 37040 | Humanity is doomed\r\r\n#coronavirus #coronacr... | Extremely Negative | 0 |

37041 rows × 4 columns

FIG 4 : All 3 files merged into 1 dataframe

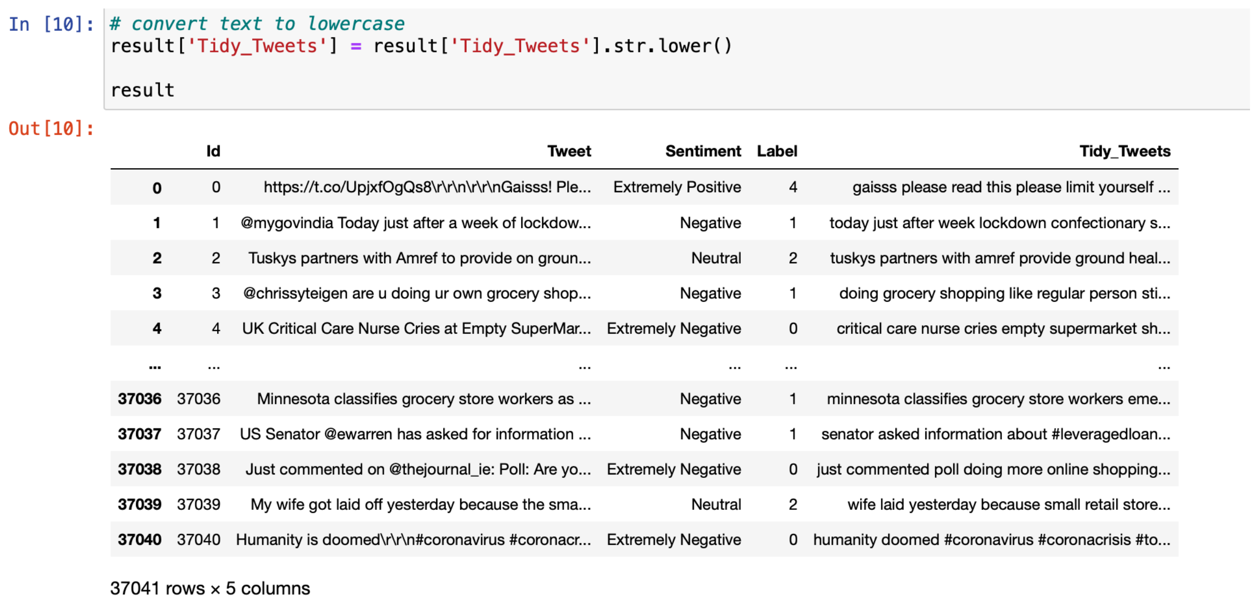
Preprocessing Data:

For Text Classification, numerous steps have to be carried out.

1. Removal of @user, since it does not help in text classification. It is the content of the tweet and not who is tweeting it or whom it is tweeted to as anyone can have positive/negative tweets at any time.
2. Removal of links from the tweets. Since the links in themselves do not help the model in classifying anything. It is the content of the tweet that matters.
3. Removing punctuation marks, digits, special characters, whitespaces etc.
4. Converting it all into lowercase (so that its uniform)

Additionally, I also removed words of length 3 and less. As words like the, and, etc. do not contribute to the sentiment of a text.

Post these, our data now looks like :



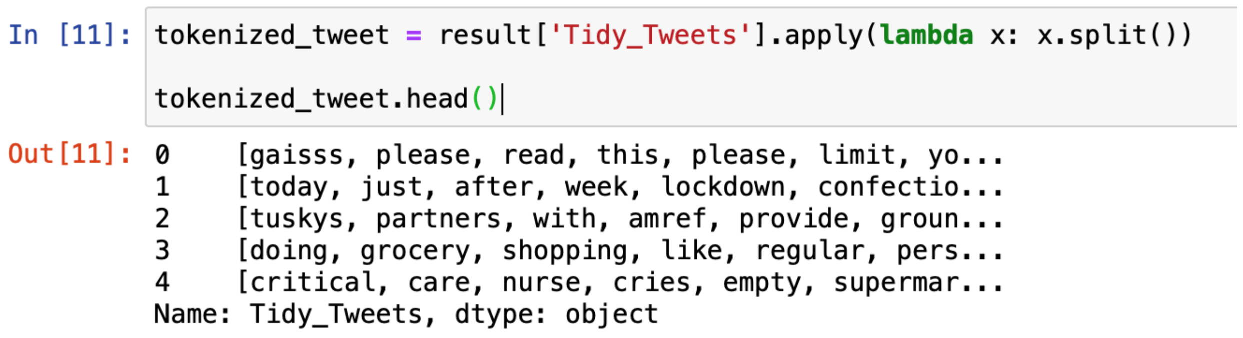
```
In [10]: # convert text to lowercase
         result['Tidy_Tweets'] = result['Tidy_Tweets'].str.lower()

         result
Out[10]:
```

| | Id | Tweet | Sentiment | Label | Tidy_Tweets |
|---|---|---|---|---|---|
| **0** | 0 | https://t.co/UpjxfOgQs8\r\r\n\r\r\nGaisss! Ple... | Extremely Positive | 4 | gaisss please read this please limit yourself ... |
| **1** | 1 | @mygovindia Today just after a week of lockdow... | Negative | 1 | today just after week lockdown confectionary s... |
| **2** | 2 | Tuskys partners with Amref to provide on groun... | Neutral | 2 | tuskys partners with amref provide ground heal... |
| **3** | 3 | @chrissyteigen are u doing ur own grocery shop... | Negative | 1 | doing grocery shopping like regular person sti... |
| **4** | 4 | UK Critical Care Nurse Cries at Empty SuperMar... | Extremely Negative | 0 | critical care nurse cries empty supermarket sh... |
| **...** | ... | ... | ... | ... | ... |
| **37036** | 37036 | Minnesota classifies grocery store workers as ... | Negative | 1 | minnesota classifies grocery store workers eme... |
| **37037** | 37037 | US Senator @ewarren has asked for information ... | Negative | 1 | senator asked information about #leveragedloan... |
| **37038** | 37038 | Just commented on @thejournal_ie: Poll: Are yo... | Extremely Negative | 0 | just commented poll doing more online shopping... |
| **37039** | 37039 | My wife got laid off yesterday because the sma... | Neutral | 2 | wife laid yesterday because small retail store... |
| **37040** | 37040 | Humanity is doomed\r\r\n#coronavirus #coronacr... | Extremely Negative | 0 | humanity doomed #coronavirus #coronacrisis #to... |

37041 rows × 5 columns

FIG 5 : Now processed data is stored under column 'Tidy_Tweets'. This is done so to ensure that the differences post processing can also be seen.

The next step is to tokenize the data.
Tokenization occurs so that sentences can be broken down into individual words. Tokenization is a step which splits longer strings of text into smaller pieces, or tokens. Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words, etc. Further processing is generally performed after a piece of text has been appropriately tokenized. Tokenization is also referred to as text segmentation or lexical analysis.

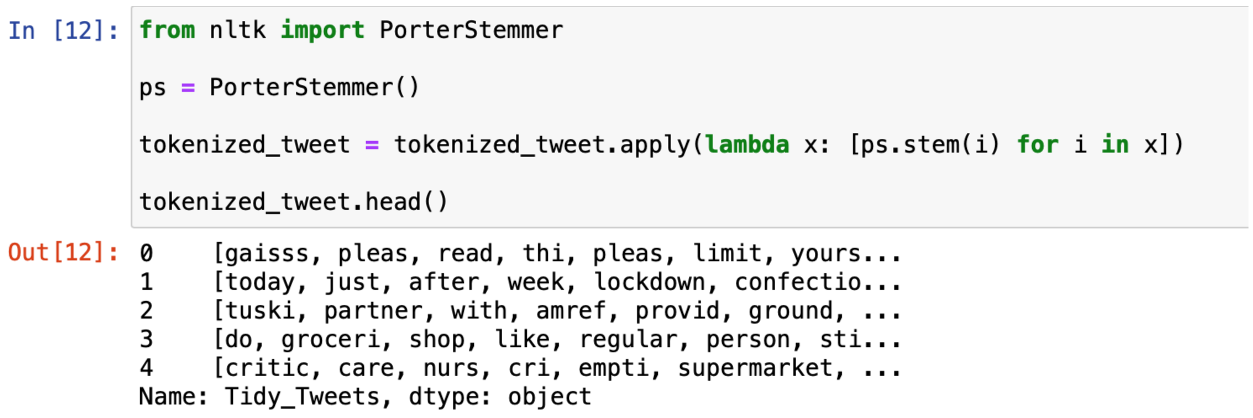


```
In [11]: tokenized_tweet = result['Tidy_Tweets'].apply(lambda x: x.split())

         tokenized_tweet.head()
Out[11]: 0    [gaisss, please, read, this, please, limit, yo...
         1    [today, just, after, week, lockdown, confectio...
         2    [tuskys, partners, with, amref, provide, groun...
         3    [doing, grocery, shopping, like, regular, pers...
         4    [critical, care, nurse, cries, empty, supermar...
         Name: Tidy_Tweets, dtype: object
```

FIG 6 : Data after tokenization

Next, we stem the data.

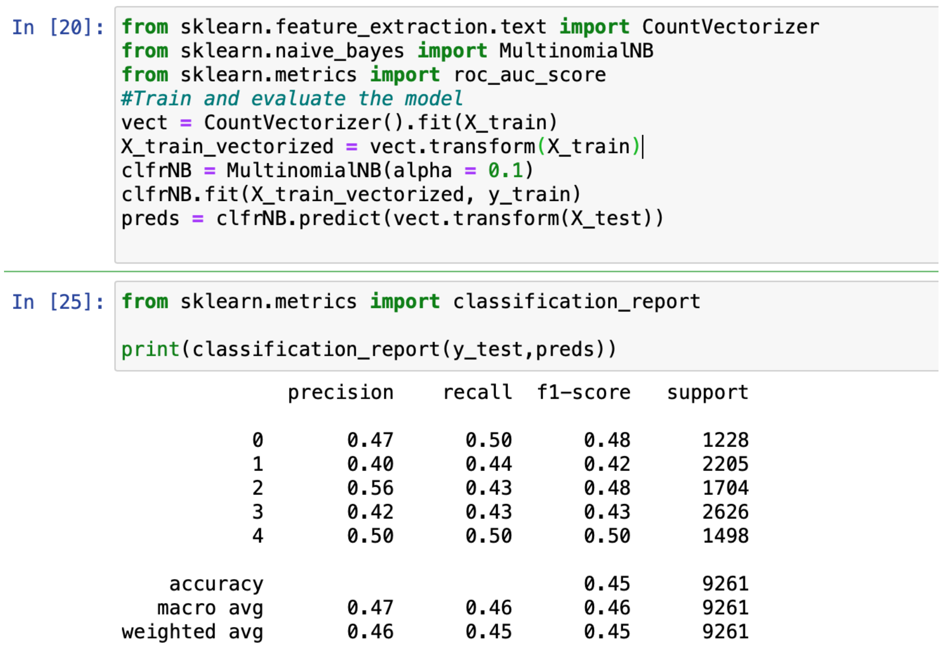
Stemming is done for eliminating affixes (suffixed, prefixes, infixes, circumfixes) from a word in order to obtain a word stem. Eg - running becomes run



```
In [12]: from nltk import PorterStemmer

         ps = PorterStemmer()

         tokenized_tweet = tokenized_tweet.apply(lambda x: [ps.stem(i) for i in x])

         tokenized_tweet.head()

Out[12]: 0    [gaisss, pleas, read, thi, pleas, limit, yours...
         1    [today, just, after, week, lockdown, confectio...
         2    [tuski, partner, with, amref, provid, ground, ...
         3    [do, groceri, shop, like, regular, person, sti...
         4    [critic, care, nurs, cri, empti, supermarket, ...
         Name: Tidy_Tweets, dtype: object
```

FIG 7 : Stemmed data

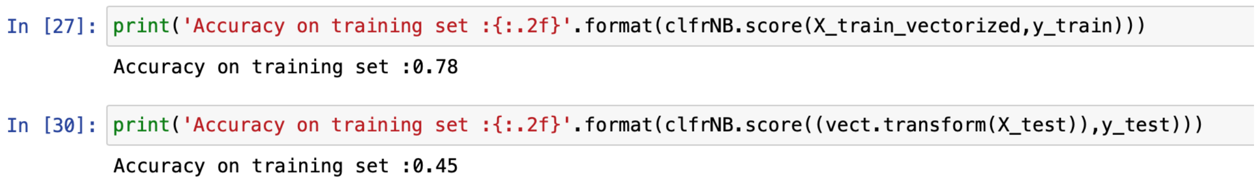
Let’s see how well the model performs on this imbalanced data and if there is actually a need to augment it.
Data has to be vectorized i.e. coded as numbers for the machine to understand it.
Now, after building the model, it’s classification report stating it’s metrics will help us understand how effective the model is



```
In [20]: from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.naive_bayes import MultinomialNB
         from sklearn.metrics import roc_auc_score
         #Train and evaluate the model
         vect = CountVectorizer().fit(X_train)
         X_train_vectorized = vect.transform(X_train)
         clfrNB = MultinomialNB(alpha = 0.1)
         clfrNB.fit(X_train_vectorized, y_train)
         preds = clfrNB.predict(vect.transform(X_test))

In [25]: from sklearn.metrics import classification_report

         print(classification_report(y_test,preds))

                       precision    recall  f1-score   support

                    0       0.47      0.50      0.48      1228
                    1       0.40      0.44      0.42      2205
                    2       0.56      0.43      0.48      1704
                    3       0.42      0.43      0.43      2626
                    4       0.50      0.50      0.50      1498

             accuracy                           0.45      9261
            macro avg       0.47      0.46      0.46      9261
         weighted avg       0.46      0.45      0.45      9261
```

FIG 8 : The Naive Bayes Classifier along with its classification report

Further, let’s also test the model accuracy on training and testing data both.

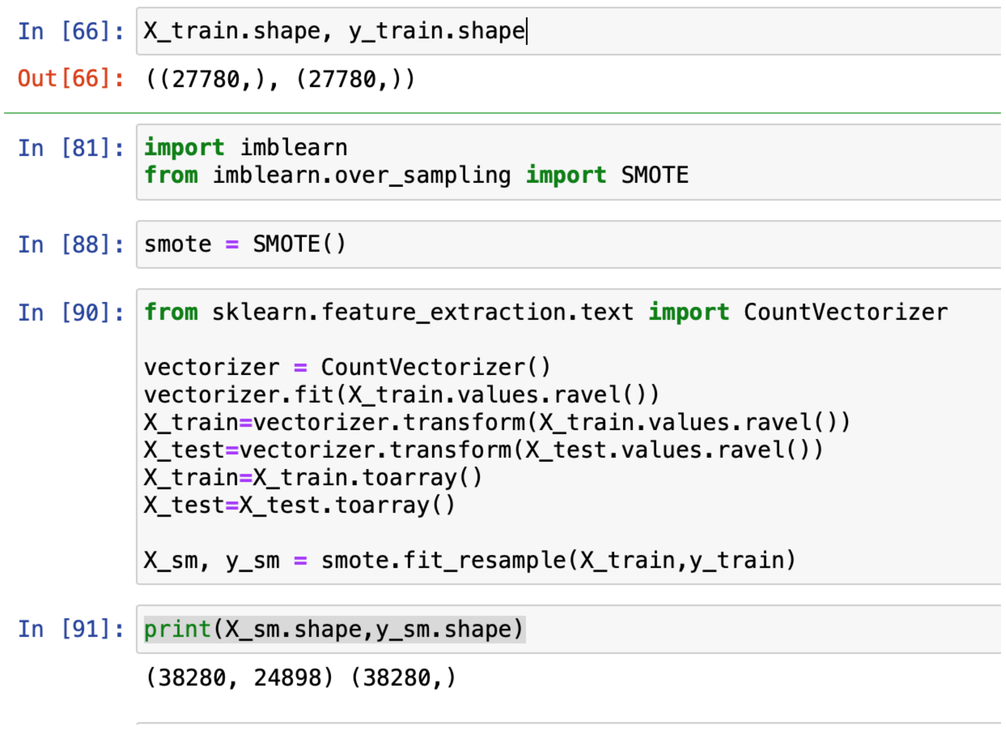


```
In [27]: print('Accuracy on training set :{:.2f}'.format(clfrNB.score(X_train_vectorized,y_train)))

         Accuracy on training set :0.78

In [30]: print('Accuracy on training set :{:.2f}'.format(clfrNB.score((vect.transform(X_test)),y_test)))

         Accuracy on training set :0.45
```

FIG 9 : Accuracy metric

As can be seen from the figure above, the accuracy of the Naive Bayes model is low at 0.45. Although it’s a decent score, this can still be improved. Alos, to be kept in mind is that the data here is imbalanced and not augmented.

SMOTE (Synthetic Minority Over-sampling Technique) is an over-sampling method.Over-sampling adds more samples from under-represented classes (majorly label 0 here) It creates synthetic samples of the minority class. This helps in balancing data. We use imblearn python package to over-sample the minority classes .

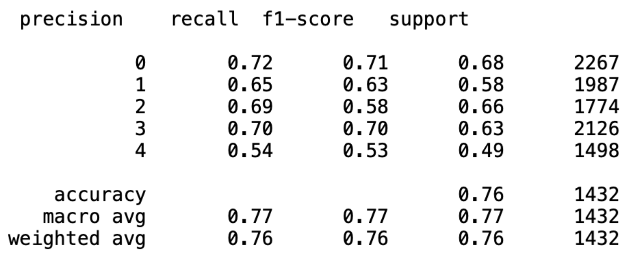


```
In [66]: X_train.shape, y_train.shape

Out[66]: ((27780,), (27780,))

In [81]: import imblearn
         from imblearn.over_sampling import SMOTE

In [88]: smote = SMOTE()

In [90]: from sklearn.feature_extraction.text import CountVectorizer

         vectorizer = CountVectorizer()
         vectorizer.fit(X_train.values.ravel())
         X_train=vectorizer.transform(X_train.values.ravel())
         X_test=vectorizer.transform(X_test.values.ravel())
         X_train=X_train.toarray()
         X_test=X_test.toarray()

         X_sm, y_sm = smote.fit_resample(X_train,y_train)

In [91]: print(X_sm.shape,y_sm.shape)

         (38280, 24898) (38280,)
```

FIG 10 : Oversampling using SMOTE

As can be seen, the training data now has increased data in order to be a balanced set.

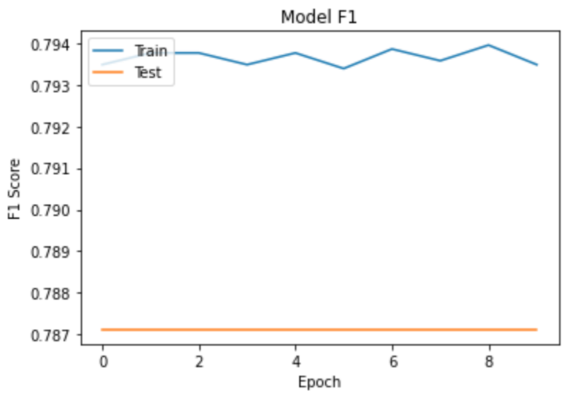
Now, this data is used for the Naive Bayes classifier again (augmented data this time) and the results are as follows :



```
  precision    recall  f1-score   support

           0       0.72      0.71      0.68      2267
           1       0.65      0.63      0.58      1987
           2       0.69      0.58      0.66      1774
           3       0.70      0.70      0.63      2126
           4       0.54      0.53      0.49      1498

    accuracy                           0.76      1432
   macro avg       0.77      0.77      0.77      1432
weighted avg       0.76      0.76      0.76      1432
```

FIG 11 : Performance metrics of augmented data Naive Bayes Classifier

We can see that the accuracy, f score, precision and recall have improved drastically.
So while this model is alright, I will now be going ahead to train a neural network to classify the given data.



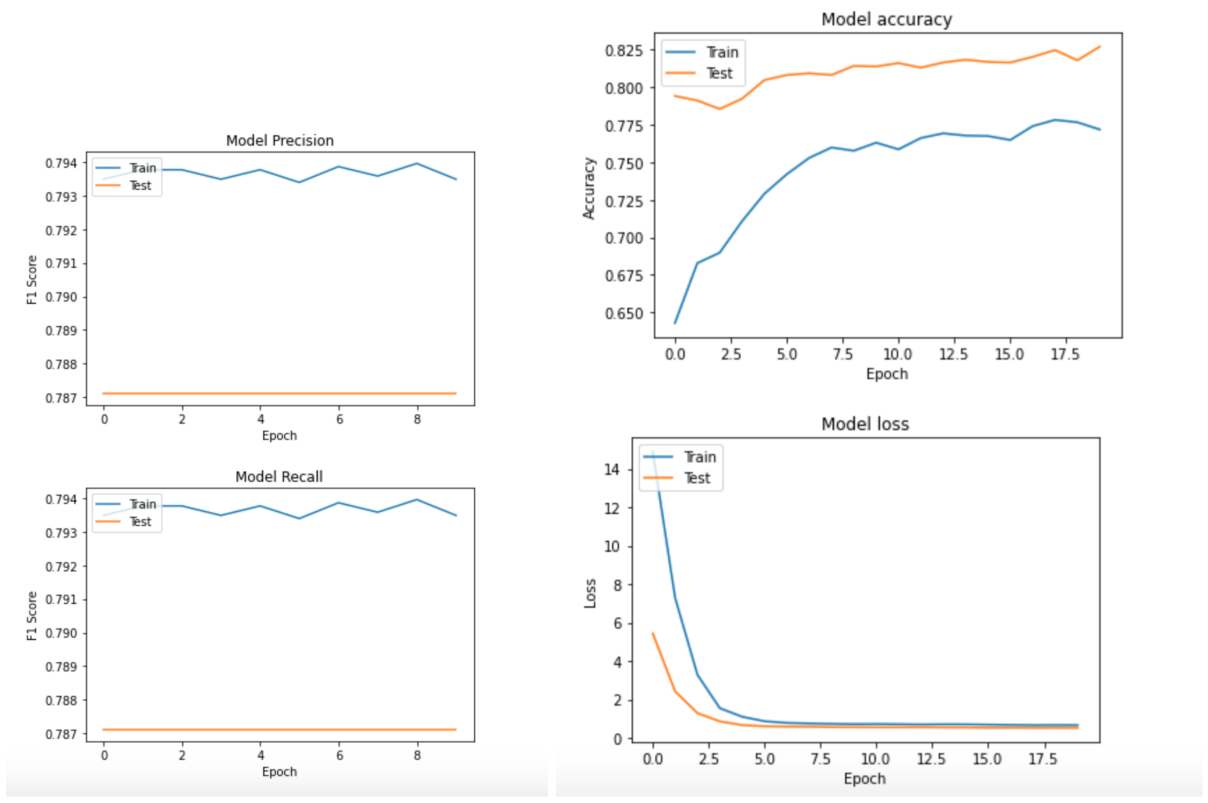
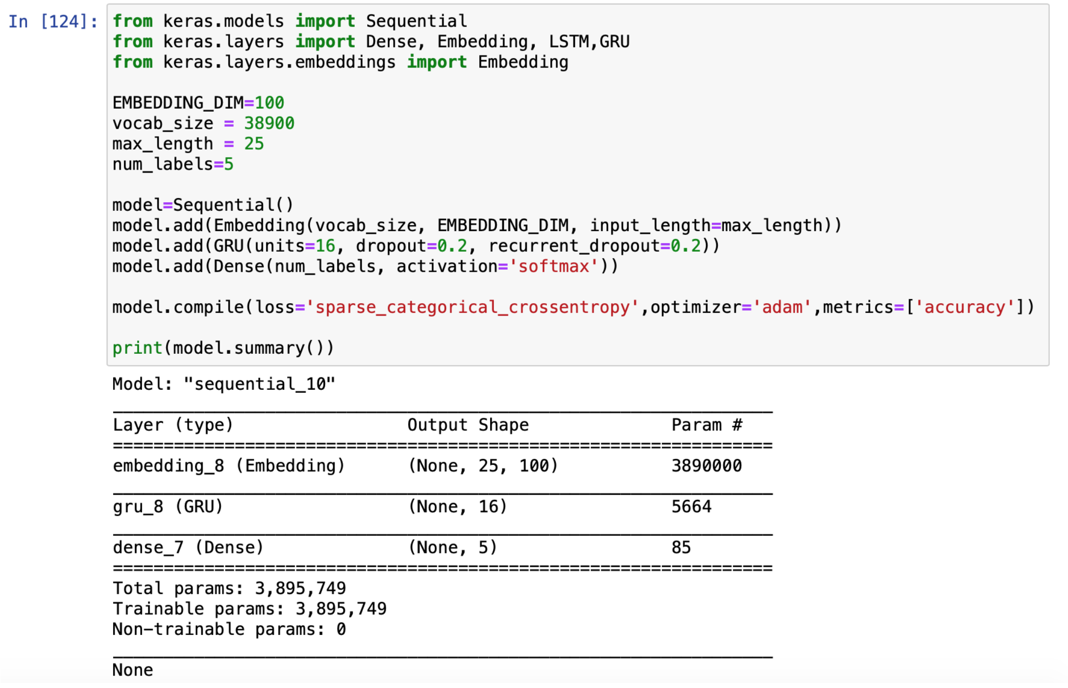


FIG 12 : Graphs of Model 2's parameter metrics

Now, to start with the Neural Network used for Text Classification is Keras Sequential Model.



```
In [124]: from keras.models import Sequential
          from keras.layers import Dense, Embedding, LSTM,GRU
          from keras.layers.embeddings import Embedding

          EMBEDDING_DIM=100
          vocab_size = 38900
          max_length = 25
          num_labels=5

          model=Sequential()
          model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
          model.add(GRU(units=16, dropout=0.2, recurrent_dropout=0.2))
          model.add(Dense(num_labels, activation='softmax'))

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

          print(model.summary())

          Model: "sequential_10"
          _________________________________________________________________
          Layer (type)                 Output Shape              Param #
          =================================================================
          embedding_8 (Embedding)      (None, 25, 100)           3890000
          _________________________________________________________________
          gru_8 (GRU)                  (None, 16)                5664
          _________________________________________________________________
          dense_7 (Dense)              (None, 5)                 85
          =================================================================
          Total params: 3,895,749
          Trainable params: 3,895,749
          Non-trainable params: 0
          _________________________________________________________________
          None
```

FIG 13 : A Neural network for Text Classification (3 layers)

Here, the type of model used is sequential. A sequential model helps stack layers one on top of the other.

Layers *early* in the network architecture (i.e., closer to the actual input image) learn *fewer* convolutional filters while layers *deeper* in the network (i.e., closer to the output predictions) will learn *more* filters.

Keras offers an Embedding layer that can be used for neural networks on text data.

It requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API also provided with Keras.

The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.

It is a flexible layer that can be used in a variety of ways, such as:

- It can be used alone to learn a word embedding that can be saved and used in another model later.
- It can be used as part of a deep learning model where the embedding is learned along with the model itself.
- It can be used to load a pre-trained word embedding model, a type of transfer learning.

The Embedding layer is defined as the first hidden layer of a network. It must specify 3 arguments:

It must specify 3 arguments:

- input_dim: This is the size of the vocabulary in the text data. For example, if our data is integer encoded to values between 0-10, then the size of the vocabulary would be 11 words.
- output_dim: This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word. For example, it could be 32 or 100 or even larger.
- input_length: This is the length of input sequences, as you would define for any input layer of a Keras model. For example, if all of your input texts are comprised of 1000 words, this would be 1000.

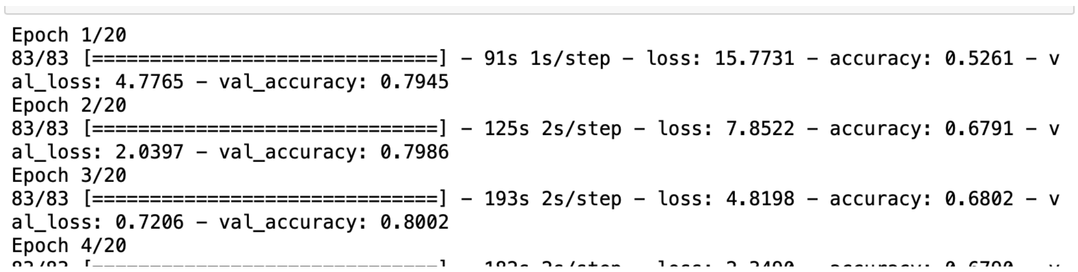
The GRU (Gated Recurrent unit) layer makes this network a Recurrent Neural network. It comprises the reset gate and the update gate instead of the input, output and forget gate.

Dense implements the operation:
output = activation(dot product (input, kernel) + bias)
where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer.

The final Dense layer outputs 5 labels with a softmax activation. Softmax is an activation function because it not only maps our output to a [0,1] range but also maps each output in such a way that the total sum is 1. The output of Softmax is therefore a probability distribution.

Direct integer values have been used so there is Sparse categorical cross entropy, however, not all models do and hence encoding is preferred. Sparse categorical cross entropy is used when the classes are mutually exclusive (i.e. when each sample belongs exactly to one class) and categorical cross entropy is used when one sample can have multiple classes or labels are soft probabilities (eg - [0.5, 0.3, 0.2]).

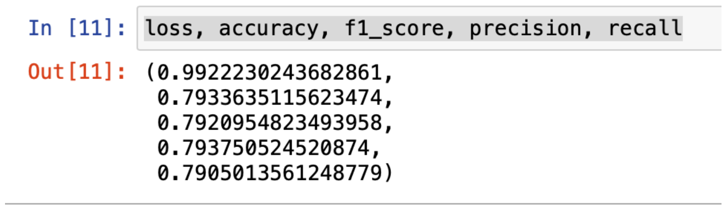
Next the model is trained.



```
Epoch 1/20
83/83 [==============================] - 91s 1s/step - loss: 15.7731 - accuracy: 0.5261 - v
al_loss: 4.7765 - val_accuracy: 0.7945
Epoch 2/20
83/83 [==============================] - 125s 2s/step - loss: 7.8522 - accuracy: 0.6791 - v
al_loss: 2.0397 - val_accuracy: 0.7986
Epoch 3/20
83/83 [==============================] - 193s 2s/step - loss: 4.8198 - accuracy: 0.6802 - v
al_loss: 0.7206 - val_accuracy: 0.8002
Epoch 4/20
```

FIG 14 : Now training the model by fitting it for 20 epochs (iterations)

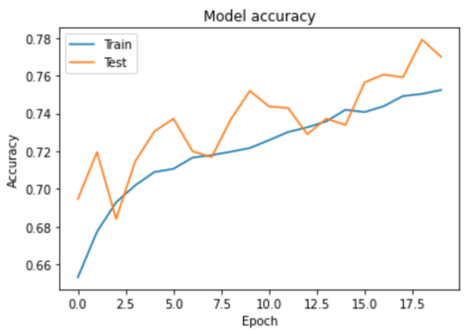
When the epochs are completed, now i want to check how efficient my model really is. For that I would be plotting graphs for the model's performance on both the training and test sets for the following parameters - Loss, Accuracy, F-score, Precision and Recall.

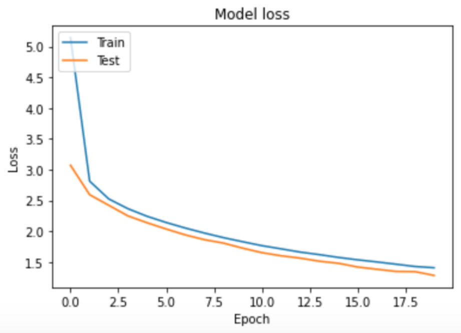


```
In [11]: loss, accuracy, f1_score, precision, recall

Out[11]: (0.9922230243682861,
          0.7933635115623474,
          0.7920954823493958,
          0.793750524520874,
          0.7905013561248779)
```

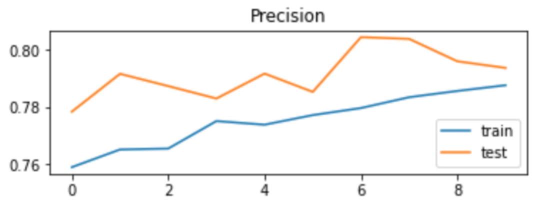
FIG 15 : Model 3 Metrics

Its graphs are :

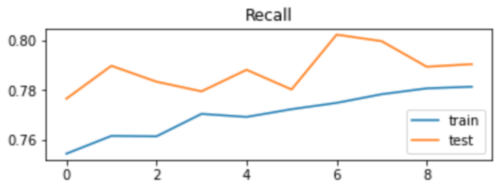




```
Out[35]: <matplotlib.legend.Legend at 0x7fc17618beb0>
```



```
Out[34]: <matplotlib.legend.Legend at 0x7fc17623d730>
```



```
Out[36]: <matplotlib.legend.Legend at 0x7fc17626a8e0>
```

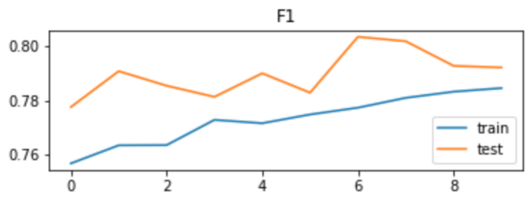


FIG 16 : Model 3 parameter metrics graphs

As can be seen from the accuracy and loss graphs, the model is alright (better than Naive Bayes).

Accuracy is one metric for evaluating classification models. It is the fraction of predictions our model got right. Accuracy = Number of correct predictions/ Total number of predictions.

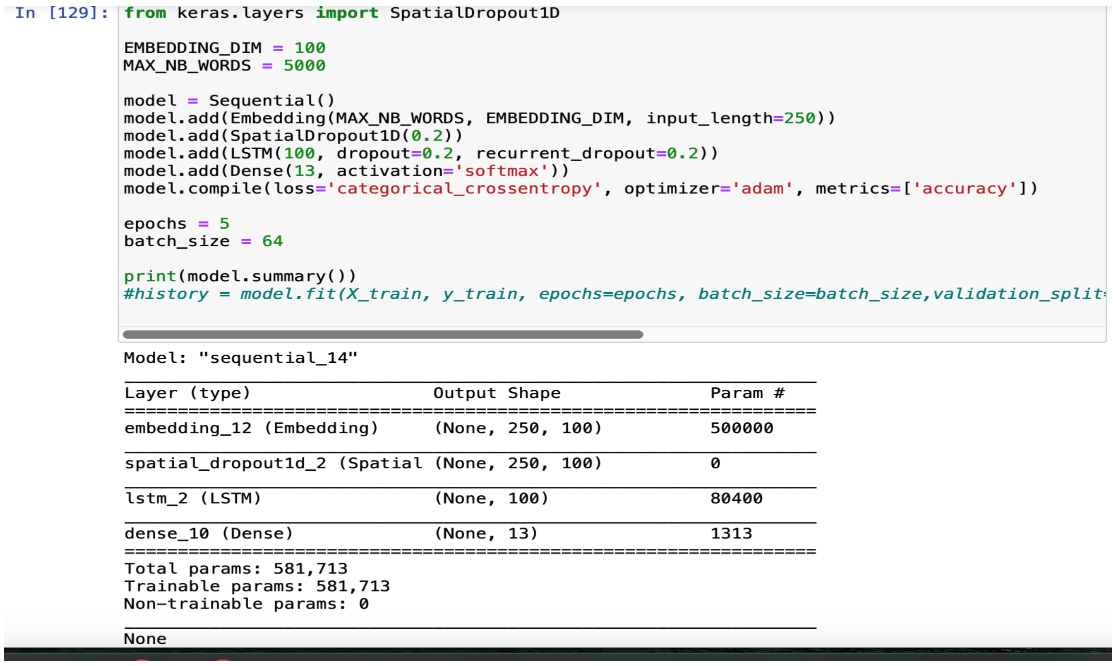
The Loss Function is one of the important components of Neural Networks. Loss is a prediction error of the Neural Net. And the method to calculate the loss is called LossFunction. Loss is used to calculate the gradients. And gradients are used to update the weights of the Neural Net.

Precision is the number of true positives (i.e. the number of items correctly labelled as belonging to the positive class) divided by the total number of elements labelled as belonging to the positive class (i.e. the sum of true positives and false positives). It is the degree of refinement with which an operation is performed.

When the recall is high, it means the model can classify all the positive samples correctly as Positive. Thus, the model can be trusted in its ability to detect positive samples.

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst at 0.

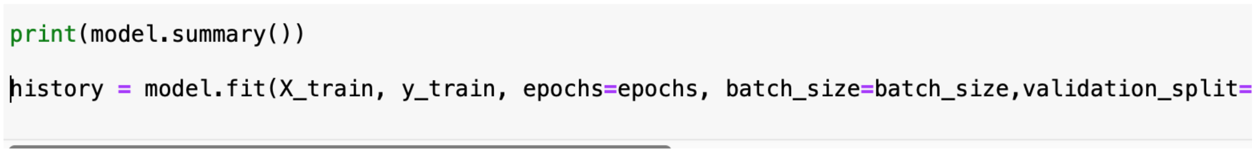
The last and fourth model I used is :



```
In [129]: from keras.layers import SpatialDropout1D

          EMBEDDING_DIM = 100
          MAX_NB_WORDS = 5000

          model = Sequential()
          model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=250))
          model.add(SpatialDropout1D(0.2))
          model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
          model.add(Dense(13, activation='softmax'))
          model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

          epochs = 5
          batch_size = 64

          print(model.summary())
          #history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,validation_split
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_12 (Embedding)     (None, 250, 100)          500000
_________________________________________________________________
spatial_dropout1d_2 (Spatial (None, 250, 100)          0
_________________________________________________________________
lstm_2 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_10 (Dense)             (None, 13)                1313
=================================================================
Total params: 581,713
Trainable params: 581,713
Non-trainable params: 0
_________________________________________________________________
None
```

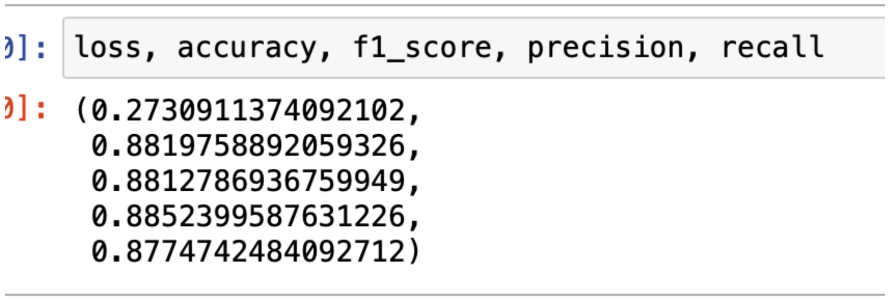
FIG 17 : Model 4 with 4 layers



```
print(model.summary())

history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,validation_split=
```

FIG 18 : Training the model

Doing so gives the following metrics :

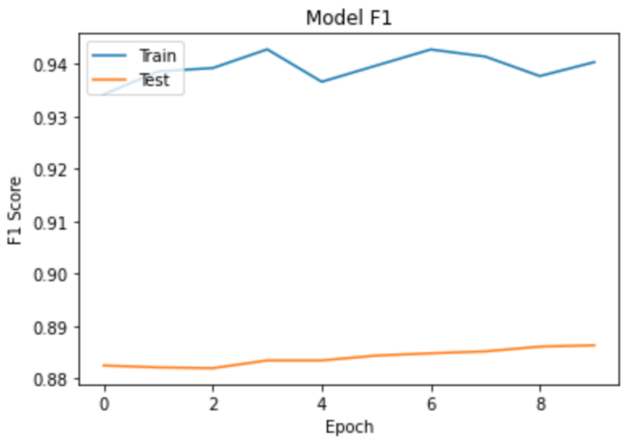


```
0]: loss, accuracy, f1_score, precision, recall

0]: (0.2730911374092102,
     0.8819758892059326,
     0.8812786936759949,
     0.8852399587631226,
     0.8774742484092712)
```

FIG 18 : Performance metric measure

We see that these metrics are the highest of the 4 models tried and hence this is the best fit for text classification for our dataset.

It’s graphs are :



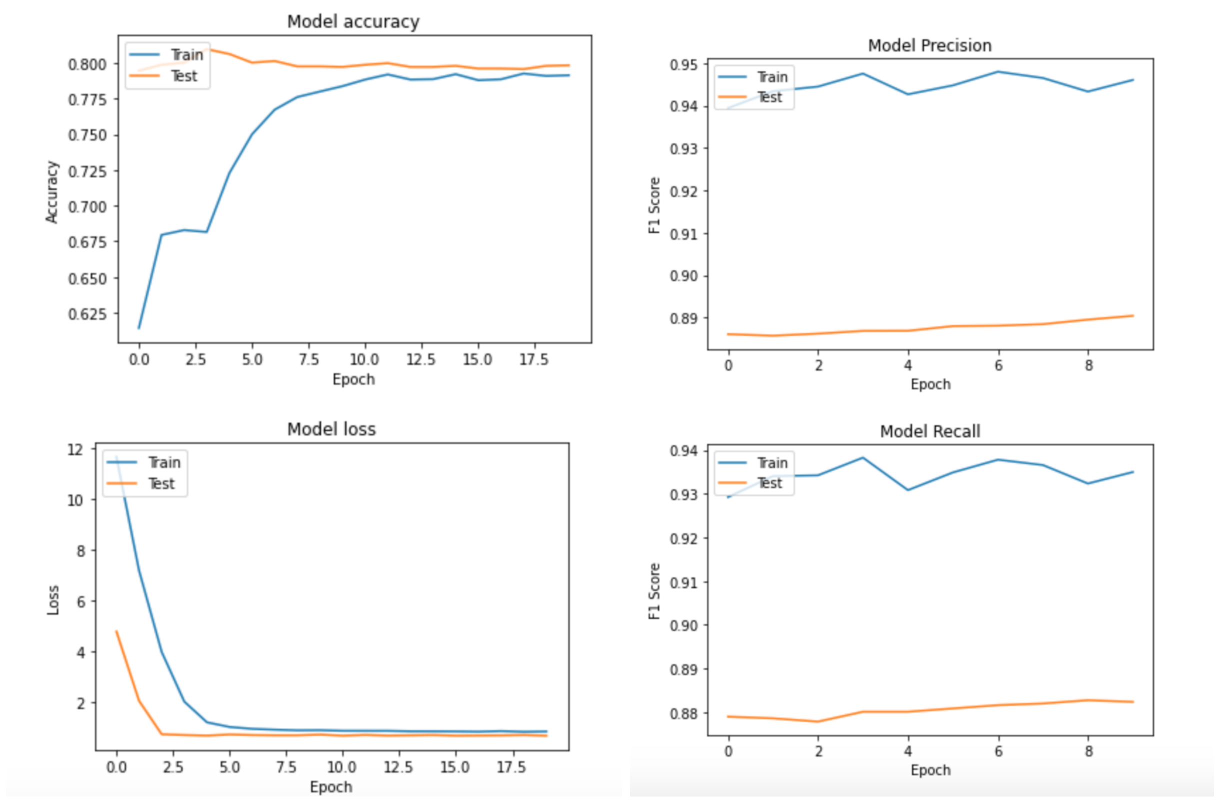


FIG 19 : Performance Parameters of the fourth model

## Conclusion:

So, 4 models have been compared on the basis of their performance metrics and data.

The first model had an imbalanced dataset and performed the poorest of all. The next 3 datasets were equally sampled. This showed the importance of a good dataset that is balanced. We see that due to highly imbalanced data( class 0 having the least amount of samples ), data augmentation is the path that boosted the model performance.

I also showed which model is the best to use (the last one with 4 layers) on the basis of performance metrics, which were the best for this model, with the least loss and highest accuracy.

[1]https://medium.com/vickdata/detecting-hate-speech-in-tweets-natural-language-processing-in-python-for-beginners-4e591952223
[2]https://towardsdatascience.com/machine-learning-multiclass-classification-with-imbalanced-data-set-29f6a177c1a
[3]https://towardsdatascience.com/machine-learning-word-embedding-sentiment-classification-using-keras-b83c28087456
[4]https://www.kdnuggets.com/2017/12/general-approach-preprocessing-text-data.html
[5]https://towardsdatascience.com/algorithms-for-text-classification-part-1-naive-bayes-3ff1d116fdd8
[6]https://towardsdatascience.com/text-classification-using-naive-bayes-theory-a-working-example-2ef4b7eb7d5a

DATA AVAILABLE AT
https://www.kaggle.com/c/usc-dsci552-section-32415d-spring-2021-ps5/data

CODE AVAILABLE AT
https://github.com/usc-dsci552-32415D-spring2021/problem-set-05-AnanyaSharma25/tree/main