**import** pandas **as** pd *#Basic packages for creating dataframes and loading dataset*  
**import** numpy **as** np  
  
**import** matplotlib.pyplot **as** plt *#Package for visualization*  
  
**import** re *#importing package for Regular expression operations*  
  
**from** sklearn.model\_selection **import** train\_test\_split *#Package for splitting the data*  
  
**from** sklearn.preprocessing **import** LabelEncoder *#Package for conversion of categorical to Numerical*  
  
**from** keras.preprocessing.text **import** Tokenizer *#Tokenization*  
**from** tensorflow.keras.preprocessing.sequence **import** pad\_sequences *#Add zeros or crop based on the length*  
**from** keras.models **import** Sequential *#Sequential Neural Network*  
**from** keras.layers **import** Dense, Embedding, LSTM, SpatialDropout1D *#For layers in Neural Network*  
**from** keras.utils.np\_utils **import** to\_categorical

[2]

**from** google.colab **import** drive   
drive.mount('/content/gdrive')

Mounted at /content/gdrive

[7]

**import** pandas **as** pd  
  
*# Load the dataset as a Pandas DataFrame*  
dataset = pd.read\_csv('/content/gdrive/My Drive/Sentiment.csv')  
  
*# Select only the necessary columns 'text' and 'sentiment'*  
mask = dataset.columns.isin(['text', 'sentiment'])  
data = dataset.loc[:, mask]  
  
*# Keeping only the necessary columns*  
data['text'] = data['text'].apply(**lambda** x: x.lower())  
data['text'] = data['text'].apply((**lambda** x: re.sub('[^a-zA-z0-9\s]', '', x)))

<ipython-input-7-d0e745dc69e5>:11: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 data['text'] = data['text'].apply(lambda x: x.lower())  
<ipython-input-7-d0e745dc69e5>:12: SettingWithCopyWarning:   
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead  
  
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy  
 data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

[8]

**for** idx, row **in** data.iterrows():  
    row[0] = row[0].replace('rt', ' ') *#Removing Retweets*  
    max\_fatures = 2000  
tokenizer = Tokenizer(num\_words=max\_fatures, split=' ') *#Maximum words is 2000 to tokenize sentence*  
tokenizer.fit\_on\_texts(data['text'].values)   
X = tokenizer.texts\_to\_sequences(data['text'].values) *#taking values to feature matrix*  
X = pad\_sequences(X) *#Padding the feature matrix*  
  
embed\_dim = 128 *#Dimension of the Embedded layer*  
lstm\_out = 196 *#Long short-term memory (LSTM) layer neurons*  
**def** createmodel():  
    model = Sequential() *#Sequential Neural Network*  
    model.add(Embedding(max\_fatures, embed\_dim,input\_length = X.shape[1])) *#input dimension 2000 Neurons, output dimension 128 Neurons*  
    model.add(LSTM(lstm\_out, dropout=0.2, recurrent\_dropout=0.2)) *#Drop out 20%, 196 output Neurons, recurrent dropout 20%*  
    model.add(Dense(3,activation='softmax')) *#3 output neurons[positive, Neutral, Negative], softmax as activation*  
    model.compile(loss = 'categorical\_crossentropy', optimizer='adam',metrics = ['accuracy']) *#Compiling the model*  
    **return** model  
*# print(model.summary())*  
labelencoder = LabelEncoder() *#Applying label Encoding on the label matrix*  
integer\_encoded = labelencoder.fit\_transform(data['sentiment']) *#fitting the model*  
y = to\_categorical(integer\_encoded)  
X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,y, test\_size = 0.33, random\_state = 42) *#67% training data, 33% test data split*  
batch\_size = 32 *#Batch size 32*  
model = createmodel() *#Function call to Sequential Neural Network*  
model.fit(X\_train, Y\_train, epochs = 1, batch\_size=batch\_size, verbose = 2) *#verbose the higher, the more messages*  
score,acc = model.evaluate(X\_test,Y\_test,verbose=2,batch\_size=batch\_size) *#evaluating the model*  
print(score)  
print(acc)

291/291 - 42s - loss: 0.8306 - accuracy: 0.6441 - 42s/epoch - 144ms/step  
144/144 - 3s - loss: 0.7514 - accuracy: 0.6791 - 3s/epoch - 22ms/step  
0.7513718008995056  
0.6791175007820129

[9]

print(model.metrics\_names) *#metrics of the model*

['loss', 'accuracy']

[10]

*#1. Save the model and use the saved model to predict on new text data (ex, “A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump”)*

[11]

model.save('sentimentAnalysis.h5') *#Saving the model*

[12]

**from** keras.models **import** load\_model *#Importing the package for importing the saved model*  
model= load\_model('sentimentAnalysis.h5') *#loading the saved model*

[13]

print(integer\_encoded)  
print(data['sentiment'])

[1 2 1 ... 2 0 2]  
0 Neutral  
1 Positive  
2 Neutral  
3 Positive  
4 Positive  
 ...   
13866 Negative  
13867 Positive  
13868 Positive  
13869 Negative  
13870 Positive  
Name: sentiment, Length: 13871, dtype: object

[14]

*# Predicting on the text data*  
sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing.@realDonaldTrump']  
sentence = tokenizer.texts\_to\_sequences(sentence) *# Tokenizing the sentence*  
sentence = pad\_sequences(sentence, maxlen=28, dtype='int32', value=0) *# Padding the sentence*  
sentiment\_probs = model.predict(sentence, batch\_size=1, verbose=2)[0] *# Predicting the sentence text*  
sentiment = np.argmax(sentiment\_probs)  
  
print(sentiment\_probs)  
**if** sentiment == 0:  
    print("Neutral")  
**elif** sentiment < 0:  
    print("Negative")  
**elif** sentiment > 0:  
    print("Positive")  
**else**:  
    print("Cannot be determined")

1/1 - 0s - 270ms/epoch - 270ms/step  
[0.72844136 0.10584743 0.16571125]  
Neutral

[15]

*#2. Apply GridSearchCV on the source code provided in the class*

[16]

**from** keras.wrappers.scikit\_learn **import** KerasClassifier *#importing Keras classifier*  
**from** sklearn.model\_selection **import** GridSearchCV *#importing Grid search CV*  
  
model = KerasClassifier(build\_fn=createmodel,verbose=2) *#initiating model to test performance by applying multiple hyper parameters*  
batch\_size= [10, 20, 40] *#hyper parameter batch\_size*  
epochs = [1, 2] *#hyper parameter no. of epochs*  
param\_grid= {'batch\_size':batch\_size, 'epochs':epochs} *#creating dictionary for batch size, no. of epochs*  
grid  = GridSearchCV(estimator=model, param\_grid=param\_grid) *#Applying dictionary with hyper parameters*  
grid\_result= grid.fit(X\_train,Y\_train) *#Fitting the model*  
*# summarize results*  
print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_)) *#best score, best hyper parameters*

<ipython-input-16-6c99b49150f4>:4: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. See https://www.adriangb.com/scikeras/stable/migration.html for help migrating.  
 model = KerasClassifier(build\_fn=createmodel,verbose=2) #initiating model to test performance by applying multiple hyper parameters

744/744 - 89s - loss: 0.8275 - accuracy: 0.6466 - 89s/epoch - 120ms/step  
186/186 - 3s - loss: 0.7607 - accuracy: 0.6676 - 3s/epoch - 18ms/step  
744/744 - 82s - loss: 0.8253 - accuracy: 0.6473 - 82s/epoch - 111ms/step  
186/186 - 3s - loss: 0.7795 - accuracy: 0.6676 - 3s/epoch - 15ms/step  
744/744 - 86s - loss: 0.8231 - accuracy: 0.6434 - 86s/epoch - 116ms/step  
186/186 - 2s - loss: 0.7761 - accuracy: 0.6686 - 2s/epoch - 13ms/step  
744/744 - 84s - loss: 0.8271 - accuracy: 0.6425 - 84s/epoch - 113ms/step  
186/186 - 2s - loss: 0.7908 - accuracy: 0.6738 - 2s/epoch - 12ms/step  
744/744 - 84s - loss: 0.8205 - accuracy: 0.6451 - 84s/epoch - 113ms/step  
186/186 - 2s - loss: 0.7877 - accuracy: 0.6615 - 2s/epoch - 12ms/step  
Epoch 1/2  
744/744 - 88s - loss: 0.8231 - accuracy: 0.6426 - 88s/epoch - 119ms/step  
Epoch 2/2  
744/744 - 83s - loss: 0.6856 - accuracy: 0.7103 - 83s/epoch - 112ms/step  
186/186 - 2s - loss: 0.7281 - accuracy: 0.6859 - 2s/epoch - 13ms/step  
Epoch 1/2  
744/744 - 85s - loss: 0.8195 - accuracy: 0.6469 - 85s/epoch - 114ms/step  
Epoch 2/2  
744/744 - 82s - loss: 0.6761 - accuracy: 0.7093 - 82s/epoch - 110ms/step  
186/186 - 2s - loss: 0.7433 - accuracy: 0.6772 - 2s/epoch - 12ms/step  
Epoch 1/2  
744/744 - 85s - loss: 0.8310 - accuracy: 0.6395 - 85s/epoch - 114ms/step  
Epoch 2/2  
744/744 - 80s - loss: 0.6790 - accuracy: 0.7116 - 80s/epoch - 108ms/step  
186/186 - 2s - loss: 0.7463 - accuracy: 0.6864 - 2s/epoch - 12ms/step  
Epoch 1/2  
744/744 - 89s - loss: 0.8242 - accuracy: 0.6443 - 89s/epoch - 119ms/step  
Epoch 2/2  
744/744 - 85s - loss: 0.6745 - accuracy: 0.7134 - 85s/epoch - 114ms/step  
186/186 - 4s - loss: 0.7515 - accuracy: 0.6663 - 4s/epoch - 20ms/step  
Epoch 1/2  
744/744 - 85s - loss: 0.8125 - accuracy: 0.6500 - 85s/epoch - 114ms/step  
Epoch 2/2  
744/744 - 84s - loss: 0.6706 - accuracy: 0.7145 - 84s/epoch - 113ms/step  
186/186 - 2s - loss: 0.7821 - accuracy: 0.6615 - 2s/epoch - 13ms/step  
372/372 - 49s - loss: 0.8329 - accuracy: 0.6423 - 49s/epoch - 131ms/step  
93/93 - 2s - loss: 0.7590 - accuracy: 0.6509 - 2s/epoch - 17ms/step  
372/372 - 53s - loss: 0.8226 - accuracy: 0.6422 - 53s/epoch - 142ms/step  
93/93 - 2s - loss: 0.7991 - accuracy: 0.6686 - 2s/epoch - 20ms/step  
372/372 - 49s - loss: 0.8241 - accuracy: 0.6446 - 49s/epoch - 132ms/step  
93/93 - 2s - loss: 0.7603 - accuracy: 0.6772 - 2s/epoch - 18ms/step  
372/372 - 48s - loss: 0.8259 - accuracy: 0.6467 - 48s/epoch - 130ms/step  
93/93 - 2s - loss: 0.7427 - accuracy: 0.6803 - 2s/epoch - 17ms/step  
372/372 - 47s - loss: 0.8233 - accuracy: 0.6424 - 47s/epoch - 125ms/step  
93/93 - 2s - loss: 0.8340 - accuracy: 0.6588 - 2s/epoch - 16ms/step  
Epoch 1/2  
372/372 - 53s - loss: 0.8374 - accuracy: 0.6414 - 53s/epoch - 142ms/step  
Epoch 2/2  
372/372 - 49s - loss: 0.6863 - accuracy: 0.7063 - 49s/epoch - 132ms/step  
93/93 - 2s - loss: 0.7550 - accuracy: 0.6762 - 2s/epoch - 19ms/step  
Epoch 1/2  
372/372 - 49s - loss: 0.8281 - accuracy: 0.6494 - 49s/epoch - 132ms/step  
Epoch 2/2  
372/372 - 46s - loss: 0.6790 - accuracy: 0.7088 - 46s/epoch - 124ms/step  
93/93 - 2s - loss: 0.7583 - accuracy: 0.6600 - 2s/epoch - 18ms/step  
Epoch 1/2  
372/372 - 49s - loss: 0.8330 - accuracy: 0.6394 - 49s/epoch - 132ms/step  
Epoch 2/2  
372/372 - 45s - loss: 0.6831 - accuracy: 0.7144 - 45s/epoch - 120ms/step  
93/93 - 3s - loss: 0.7555 - accuracy: 0.6837 - 3s/epoch - 28ms/step  
Epoch 1/2  
372/372 - 54s - loss: 0.8312 - accuracy: 0.6424 - 54s/epoch - 146ms/step  
Epoch 2/2  
372/372 - 50s - loss: 0.6755 - accuracy: 0.7126 - 50s/epoch - 135ms/step  
93/93 - 2s - loss: 0.7513 - accuracy: 0.6717 - 2s/epoch - 19ms/step  
Epoch 1/2  
372/372 - 49s - loss: 0.8253 - accuracy: 0.6475 - 49s/epoch - 132ms/step  
Epoch 2/2  
372/372 - 46s - loss: 0.6669 - accuracy: 0.7196 - 46s/epoch - 125ms/step  
93/93 - 2s - loss: 0.7966 - accuracy: 0.6561 - 2s/epoch - 17ms/step  
186/186 - 30s - loss: 0.8402 - accuracy: 0.6375 - 30s/epoch - 163ms/step  
47/47 - 1s - loss: 0.7865 - accuracy: 0.6374 - 1s/epoch - 23ms/step  
186/186 - 33s - loss: 0.8433 - accuracy: 0.6355 - 33s/epoch - 180ms/step  
47/47 - 1s - loss: 0.7775 - accuracy: 0.6713 - 1s/epoch - 26ms/step  
186/186 - 32s - loss: 0.8462 - accuracy: 0.6342 - 32s/epoch - 169ms/step  
47/47 - 2s - loss: 0.7659 - accuracy: 0.6789 - 2s/epoch - 39ms/step  
186/186 - 31s - loss: 0.8494 - accuracy: 0.6336 - 31s/epoch - 164ms/step  
47/47 - 1s - loss: 0.7577 - accuracy: 0.6787 - 1s/epoch - 24ms/step  
186/186 - 33s - loss: 0.8412 - accuracy: 0.6383 - 33s/epoch - 179ms/step  
47/47 - 1s - loss: 0.7749 - accuracy: 0.6642 - 1s/epoch - 26ms/step  
Epoch 1/2  
186/186 - 31s - loss: 0.8417 - accuracy: 0.6414 - 31s/epoch - 167ms/step  
Epoch 2/2  
186/186 - 30s - loss: 0.6924 - accuracy: 0.7037 - 30s/epoch - 161ms/step  
47/47 - 1s - loss: 0.7302 - accuracy: 0.6832 - 1s/epoch - 28ms/step  
Epoch 1/2  
186/186 - 31s - loss: 0.8377 - accuracy: 0.6377 - 31s/epoch - 166ms/step  
Epoch 2/2  
186/186 - 27s - loss: 0.6905 - accuracy: 0.7086 - 27s/epoch - 148ms/step  
47/47 - 2s - loss: 0.7384 - accuracy: 0.6826 - 2s/epoch - 41ms/step  
Epoch 1/2  
186/186 - 31s - loss: 0.8403 - accuracy: 0.6391 - 31s/epoch - 168ms/step  
Epoch 2/2  
186/186 - 29s - loss: 0.6859 - accuracy: 0.7066 - 29s/epoch - 153ms/step  
47/47 - 2s - loss: 0.7515 - accuracy: 0.6724 - 2s/epoch - 42ms/step  
Epoch 1/2  
186/186 - 31s - loss: 0.8492 - accuracy: 0.6293 - 31s/epoch - 164ms/step  
Epoch 2/2  
186/186 - 28s - loss: 0.6843 - accuracy: 0.7050 - 28s/epoch - 150ms/step  
47/47 - 2s - loss: 0.7519 - accuracy: 0.6787 - 2s/epoch - 41ms/step  
Epoch 1/2  
186/186 - 30s - loss: 0.8361 - accuracy: 0.6401 - 30s/epoch - 160ms/step  
Epoch 2/2  
186/186 - 27s - loss: 0.6828 - accuracy: 0.7119 - 27s/epoch - 148ms/step  
47/47 - 3s - loss: 0.7860 - accuracy: 0.6625 - 3s/epoch - 58ms/step  
Epoch 1/2  
233/233 - 40s - loss: 0.8312 - accuracy: 0.6396 - 40s/epoch - 170ms/step  
Epoch 2/2  
233/233 - 37s - loss: 0.6839 - accuracy: 0.7096 - 37s/epoch - 158ms/step  
Best: 0.675884 using {'batch\_size': 40, 'epochs': 2}