```
NLP Assignment 3 (Sentiment Analysis)
Name : Krishna Kant Verma
Roll No : 2211cs19
Name : Gourob Chatterjee
Roll No : 2211cs08
```

*Importing All Required Libraries*

```
import spacy
import re
import sys
import os
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from keras.layers import LSTM,Dense,Flatten
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow import keras
from progressbar import progressbar
import pandas as pd
import numpy as np
```

## Zip File Extraction

```
# Extracting the Positive Zip Folder and Negative Zip Folder to POS/NEG SubFolder
```

```
import shutil
pos_zip = "/content/pos.zip"
neg_zip = "/content/neg.zip"
shutil.unpack_archive(pos_zip, "/content/POS", "zip")
shutil.unpack_archive(neg_zip, "/content/NEG", "zip")
```

```
Import OS Libarary and Read File
```

```
import os
def read_file(file_location):
    with open(file_location, 'r', encoding='utf-8') as f:
        return f.readlines()
```

```
Reading Text Data and Processing it Make it Operable
```

```
def read_text(folder_locations, max_files_toread):
    textData = []
    textFileLocations = []
    if not isinstance(folder_locations, str):
        for location in folder_locations:
            textFileLocations.append([os.path.join(location, file_name) for file_name in os.listdir(location)][:max_files_
    else:
        print("Folder locations should be in list or tuple format.\nExample: [Folder loc1, Folder loc2, Folder loc3]")
        return

    classNames = [0 for _ in folder_locations]
    currLen = 0

    for location_ind, text_file_location in enumerate(textFileLocations):
        for text_file in text_file_location:
            try:
                textData.append(read_file(text_file))
            except Exception as e:
                print(e, text_file)
        classNames[location_ind] = len(textData) - currLen
        currLen = len(textData)

```

```
    classLabels = []
    for index, no_of_files_inclass in enumerate(classNames):
        classLabels += [index for number in range(no_of_files_inclass)]
    return textData, classLabels

textData, classLabels = read_text(["/content/NEG/neg", "/content/POS/pos"], 5000)
```

Printing textData 1 output (with labels)

```
textData[1],classLabels[1]
```

```
(["Who did the research for this film? It's set in Baghdad in 2004, however all the Soldiers are wearing ACUs and have
all Universal Camouflage Pattern gear. No one was wearing that stuff in 04. <br /><br />I just saw this film while
deployed overseas and I can say that the overwhelming feeling from the audience was WTF? This movie made no sense, had
characters come and go with no explanation, and people doing ridiculous things that would NEVER happen in real life. I
realize that it's a movie, but it's obviously trying to portray something realistic. It fails miserably, but it's
trying. <br /><br />It's like someone came up with a bunch of random ideas, chewed them up and swallowed, then vomited
out a film. I would not recommend this film to anyone. I'm still not sure why I sat through the whole thing. GI Joe
was one that really made you think compared to this. STAY AWAY!"],
 0)
```

```
textData[0]
```

```
["From the creators of Shrek\x85\x85\x85\x85.. OK, that grabbed my attention.<br /><br />Well the creators of Shrek
also made Madagascar. Madagascar was half as good as Shrek.<br /><br />And now Flushed Away is half as good as
Madagascar.<br /><br />That means Flushed Away isn't good. The animation and all that special effects were extremely
good but the movie wasn't.<br /><br />The story of this movie was only meant for kids. It's seriously not possible for
adults to actually love this flick.<br /><br />But there were many jokes meant for adults. I bet kids dint understand
the jokes.<br /><br />Despite that I dint like this flick.<br /><br />I am completely disappointed. 4/10"]
```

Eliminating all unnecessary HTML and other hyper Texts

```
regex = re.compile(r'<[^>]+>')
def removeHyperTags(string):
    return regex.sub(' ', string)
```

calling removeHyperText Function

```
textData = [removeHyperTags(text[0]) for text in textData]
```

Text After Processing all the removal of hypertexts

```
textData[0]
```

```
'From the creators of Shrek\x85\x85\x85\x85.. OK, that grabbed my attention.
Well the creators of Shrek also made Madagascar. Madagascar was half as good
as Shrek.  And now Flushed Away is half as good as Madagascar.  That means F
lushed Away isn't good. The animation and all that special effects were extr
emely good but the movie wasn't.  The story of this movie was only meant for
```

## Tokenizing Texts

Function to Tokenize text

```
def tokenize(texts):
    for text_ind,text in enumerate(texts):
        texts[text_ind]=text.lower()
    nlp = spacy.load('en_core_web_sm')
    tokenizedTexts = []
    for ind,text in enumerate(texts):
        print(ind,len(texts))
        doc = nlp(text)
        tokens = [token.text for token in doc]
        tokenizedTexts.append(tokens)
    return tokenizedTexts
```

```
Calling Tokenize Text Function over First Dataset
```

```
tokenizedTexts = tokenize(textData)
```

```
9942 10000
9943 10000
9944 10000
9945 10000
9946 10000
9947 10000
9948 10000
9949 10000
9950 10000
9951 10000
9952 10000
9953 10000
9954 10000
9955 10000
9956 10000
9957 10000
9958 10000
9959 10000
9960 10000
9961 10000
9962 10000
9963 10000
9964 10000
9965 10000
9966 10000
9967 10000
9968 10000
9969 10000
9970 10000
9971 10000
9972 10000
9973 10000
9974 10000
9975 10000
9976 10000
9977 10000
9978 10000
9979 10000
9980 10000
9981 10000
9982 10000
9983 10000
9984 10000
9985 10000
9986 10000
9987 10000
9988 10000
9989 10000
9990 10000
9991 10000
9992 10000
9993 10000
9994 10000
9995 10000
9996 10000
9997 10000
9998 10000
9999 10000
```

## Finding Out Most Frequent Tokens In DataSet

```
Function to find Out most frequent tokens
```

```
def maxFrequentTokens( textData, frequency=5 ):
    counter={}
    for text in textData:
        for token in text:
            if token in counter:
                counter[token]+=1
            else:
                counter[token]=1
    FrequentTokens=[]
    for i in counter.items():
```

```
      if i[1]>frequency:
        FrequentTokens.append(i[0])
    return FrequentTokens

FrequentTokens = maxFrequentTokens(tokenizedTexts,frequency=300)

f" Most frequent Tokens length  {len(FrequentTokens)}"
```

```
' Most frequent Tokens length  814'
```

```
tokenizedTexts[1][:10]
```

```
['who', 'did', 'the', 'research', 'for', 'this', 'film', '?', 'it', "'s"]
```

## Finding Out Padding Sequences

```
def padSeq(sequences):
    averageLen=0
    for text in sequences:
        averageLen+=len(text)
    averageLen=int(averageLen/len(tokenizedTexts))
    for text_ind,text in enumerate(sequences):
        if len(text)>=averageLen:
            sequences[text_ind]=text[:averageLen]
        else:
            for i in range(averageLen-len(text)):
              sequences[text_ind].append('PAD')

```

```
padSeq(tokenizedTexts)
```

## Encoding Words to One-Hot-Encoding

```
def oneHotEncoding(tokenizedTexts):
    token_to_number={x:to_categorical(ind,num_classes=len(FrequentTokens)+1,dtype='uint8') for ind,x in enumerate(Frequen
    token_to_number['unk']=np.array([0 for x in range(len(FrequentTokens))]+[1])
    for text in tokenizedTexts:
      for token_ind,token in enumerate(text):
        if token in token_to_number:
          text[token_ind]=token_to_number[token]
        else:
          text[token_ind]=token_to_number['unk']
oneHotEncoding(tokenizedTexts)
```

```
X_array = np.array(tokenizedTexts,dtype='uint8')
```

```
dataSet_1 = X_array
```

## Second Dataset (SemEval Tweet Dataset)

```
data_2 = pd.read_csv("/content/2013semeval_train.csv")
```

```
data_2.tweet
```

```
0       Gas by my house hit $3.39!!!! I\u2019m going t...
1       Theo Walcott is still shit\u002c watch Rafa an...
2       its not that I\u2019m a GSP fan\u002c i just h...
3       Iranian general says Israel\u2019s Iron Dome c...
4       Tehran\u002c Mon Amour: Obama Tried to Establi...
                              ...
9679    RT @MNFootNg It's monday and Monday Night Foot...
9680    All I know is the road for that Lomardi start ...
9681    "All Blue and White fam, we r meeting at Golde...
9682    @DariusButler28   Have a great game agaist Tam...
9683    "I'm pisseeedddd that I missed Kid Cudi's show...
Name: tweet, Length: 9684, dtype: object
```

## unicode Cleaning

```
uniEscRegx = re.compile(r'\\u([0-9a-fA-F]{4})')
def convert_escape_sequence(match):
    return chr(int(match.group(1), 16))
```

```
result = uniEscRegx.sub(convert_escape_sequence, data_2.tweet[1])
def changeString(string):
    return uniEscRegx.sub(convert_escape_sequence,string)
```

```
print(result)
```

```
Theo Walcott is still shit, watch Rafa and Johnny deal with him on Saturday.
```

```
data_2.tweet = data_2.tweet.apply(changeString)
```

## Tokenizing Second Tweet DataSet

```
tokenizedTexts = tokenize( data_2.tweet )
```

```
9626 9684
9627 9684
9628 9684
9629 9684
9630 9684
9631 9684
9632 9684
9633 9684
9634 9684
9635 9684
9636 9684
9637 9684
9638 9684
9639 9684
9640 9684
9641 9684
9642 9684
9643 9684
9644 9684
9645 9684
9646 9684
9647 9684
9648 9684
9649 9684
9650 9684
9651 9684
9652 9684
9653 9684
9654 9684
9655 9684
9656 9684
9657 9684
9658 9684
9659 9684
9660 9684
9661 9684
9662 9684
9663 9684
9664 9684
9665 9684
9666 9684
9667 9684
9668 9684
9669 9684
9670 9684
9671 9684
9672 9684
9673 9684
9674 9684
9675 9684
9676 9684
9677 9684
9678 9684
9679 9684
9680 9684
9681 9684
9682 9684
9683 9684
```

## Finding Out Most Freuent Tokens

```
FrequentTokens = maxFrequentTokens(tokenizedTexts,frequency=200)
f" Most frequent Tokens length  {len(FrequentTokens)}"
```

```
' Most frequent Tokens length  138'
```

## Conversion to One Hot Encoding after Padding Sequences

```
padSeq(tokenizedTexts)
oneHotEncoding(tokenizedTexts)
```

```
X_array = np.array(tokenizedTexts,dtype='uint8')
```

```
dataSet_2 = X_array
```

```
dataSet_1.shape,dataSet_2.shape
```

```
((10000, 273, 815), (9684, 23, 139))
```

```
# converting first dataset labels to numpy array
```

```
dataSet_1_labels=np.array(classLabels)
```

```
# converting second dataset labels to numpy array
```

```
uniqLabels={x:ind for ind,x in enumerate(data_2.label.unique())}
```

```
data_2.label = data_2.label.apply(lambda x:uniqLabels[x])
```

```
dataSet_2_labels = data_2.label.values
```

## Finalizing Our DataSet (Shaping Our Dataset)

```
dataSet_1.shape
dataSet_1_labels.shape
dataSet_2.shape
dataSet_2_labels.shape
```

```
(9684,)
```

## Creating Model Using RNN

```
Creating Model For First Dataset
```

```
model_1_RNN=keras.Sequential([
    LSTM(256,input_shape=dataSet_1.shape[1:]),
    Dense(2,activation='sigmoid')
])
```

```
model_1_RNN.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
# Creating Model For Second Dataset
```

```
model_2_RNN=keras.Sequential([
    LSTM(256,input_shape=dataSet_2.shape[1:]),
```

```
    Dense(3,activation='sigmoid')
])
```

```
model_2_RNN.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
from sklearn.model_selection import train_test_split
print(len(dataSet_1_labels))

xTrain_1, xTest_1, yTrain_1, yTest_1 = train_test_split(dataSet_1, dataSet_1_labels, test_size=0.25, random_state=42)
xTrain_2, xTest_2, yTrain_2, yTest_2 = train_test_split(dataSet_2, dataSet_2_labels, test_size=0.25, random_state=42)
```

```
10000
```

## Excuetion of Model 1

```
execModel_1 = model_1_RNN.fit(xTrain_1,yTrain_1,epochs=10)
```

```
Epoch 1/10
235/235 [==============================] - 15s 27ms/step - loss: 0.6940 - accuracy: 0.5060
Epoch 2/10
235/235 [==============================] - 6s 27ms/step - loss: 0.6943 - accuracy: 0.5444
Epoch 3/10
235/235 [==============================] - 6s 27ms/step - loss: 0.7051 - accuracy: 0.4980
Epoch 4/10
235/235 [==============================] - 6s 27ms/step - loss: 0.6981 - accuracy: 0.5063
Epoch 5/10
235/235 [==============================] - 6s 27ms/step - loss: 0.6948 - accuracy: 0.5151
Epoch 6/10
235/235 [==============================] - 6s 27ms/step - loss: 0.6924 - accuracy: 0.5169
Epoch 7/10
235/235 [==============================] - 7s 28ms/step - loss: 0.6891 - accuracy: 0.5331
Epoch 8/10
235/235 [==============================] - 6s 27ms/step - loss: 0.6880 - accuracy: 0.5333
Epoch 9/10
235/235 [==============================] - 7s 29ms/step - loss: 0.6819 - accuracy: 0.5409
Epoch 10/10
235/235 [==============================] - 6s 28ms/step - loss: 0.6768 - accuracy: 0.5481
```

## Excuetion of Model 2

```
execModel_2 = model_2_RNN.fit(xTrain_2,yTrain_2,epochs=10)
```

```
Epoch 1/10
227/227 [==============================] - 7s 5ms/step - loss: 0.9602 - accuracy: 0.5372
Epoch 2/10
227/227 [==============================] - 1s 4ms/step - loss: 0.8888 - accuracy: 0.5918
Epoch 3/10
227/227 [==============================] - 1s 4ms/step - loss: 0.8743 - accuracy: 0.5974
Epoch 4/10
227/227 [==============================] - 1s 4ms/step - loss: 0.8643 - accuracy: 0.6042
Epoch 5/10
227/227 [==============================] - 1s 5ms/step - loss: 0.8502 - accuracy: 0.6142
Epoch 6/10
227/227 [==============================] - 1s 4ms/step - loss: 0.8355 - accuracy: 0.6165
Epoch 7/10
227/227 [==============================] - 1s 4ms/step - loss: 0.8307 - accuracy: 0.6188
Epoch 8/10
227/227 [==============================] - 1s 4ms/step - loss: 0.8131 - accuracy: 0.6292
Epoch 9/10
227/227 [==============================] - 1s 4ms/step - loss: 0.8084 - accuracy: 0.6329
Epoch 10/10
227/227 [==============================] - 1s 5ms/step - loss: 0.7920 - accuracy: 0.6468
```

## Calculating Precision Accuracy and Recall for First Dataset

```
from sklearn.metrics import precision_recall_fscore_support
y_pred_1=model_1_RNN.predict(xTest_1)
scores=precision_recall_fscore_support(np.argmax(y_pred_1,axis=1),yTest_1,average='macro')
print(f"""\n\nRNN Model_1 For DataSet1
    Precision = {scores[0]}
    Recall    = {scores[1]}
    f1_score  = {scores[2]}
""")
```

```
79/79 [==============================] - 1s 13ms/step


RNN Model_1 For DataSet1
    Precision = 0.497400794870219
    Recall    = 0.4802216538789429
    f1_score  = 0.3542091504669417
```

## Calculating Precision Accuracy and Recall for Second Dataset

```
from sklearn.metrics import precision_recall_fscore_support
y_pred_2=model_2_RNN.predict(xTest_2)
scores=precision_recall_fscore_support(np.argmax(y_pred_2,axis=1),yTest_2,average='macro')
print(f"""\n\nRNN Model_2 For DataSet_2
    Precision = {scores[0]}
    Recall    = {scores[1]}
    f1_score  = {scores[2]}
""")
```

```
76/76 [==============================] - 1s 3ms/step


RNN Model_2 For DataSet_2
    Precision = 0.4864269280775426
    Recall    = 0.545716590829335
    f1_score  = 0.48656399547127904
```

## Logs after Each Epocs

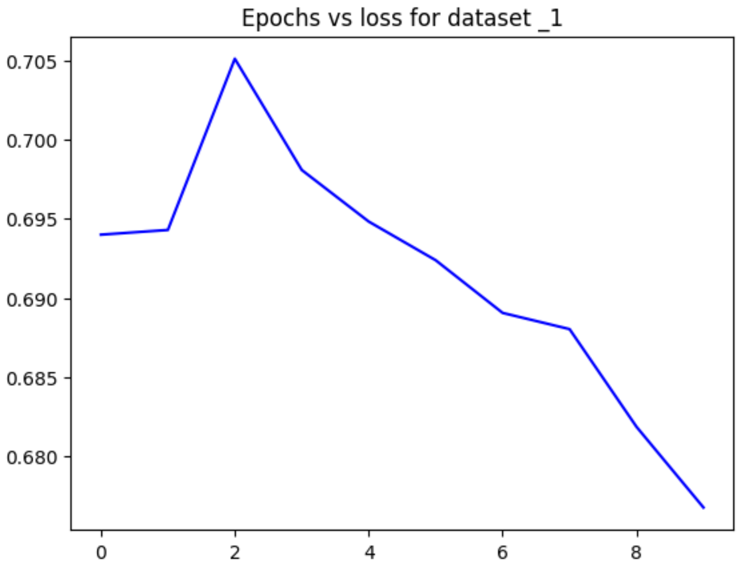
```
execModel_1.history
```

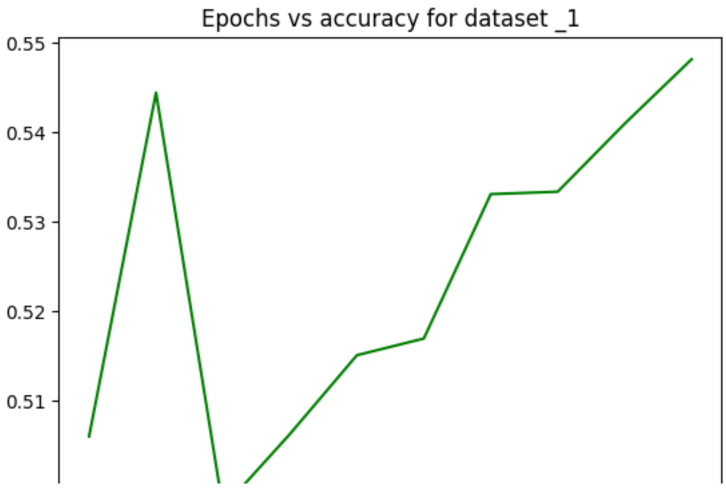

```
{'loss': [0.6939970254898071,
  0.6942936182022095,
  0.7051113247871399,
  0.6980825662612915,
  0.6948240995407104,
  0.6923822164535522,
  0.6890540719032288,
  0.6880289316177368,
  0.6818565726280212,
  0.6767584681510925],
 'accuracy': [0.5059999823570251,
  0.5443999767303467,
  0.49799999594688416,
  0.5062666535377502,
  0.5150666832923889,
  0.5169333219528198,
  0.5330666899681091,
  0.5333333611488342,
  0.5409333109855652,
  0.5481333136558533]}
```


## Plots

plotting epocs Vs Loss For Model 1

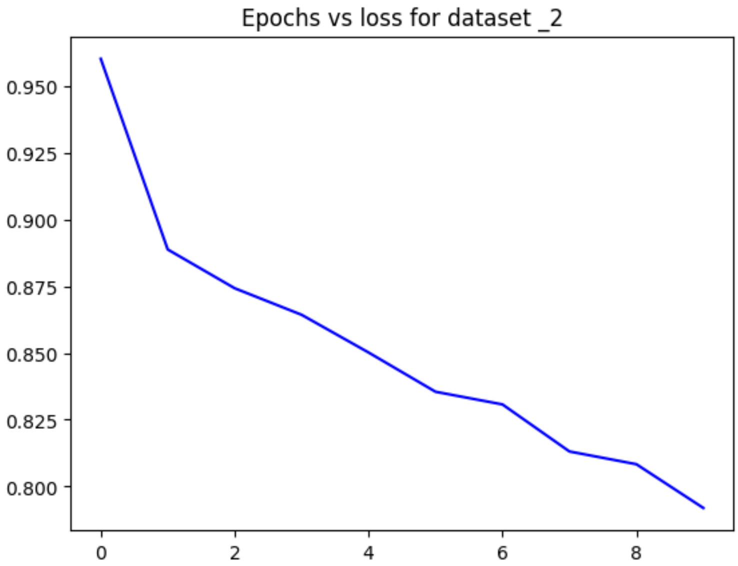
```
plt.plot([x for x in range(10)],execModel_1.history['loss'],c='blue')
plt.title("Epochs vs loss for dataset _1")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
plt.plot([x for x in range(10)],execModel_1.history['accuracy'],c='g')
plt.title("Epochs vs accuracy for dataset _1")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
```




Plotting Epocs Vs Loss for Model 2

```
plt.plot([x for x in range(10)],execModel_2.history['loss'],c='blue')
plt.title("Epochs vs loss for dataset _2")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
plt.plot([x for x in range(10)],execModel_2.history['accuracy'],c='green')
plt.title("Epochs vs accuracy for dataset _2")
plt.x_label="Epochs"
plt.y_label="Loss"
plt.show()
```




## Feed forward Neural Networks Architecture

My proposed feed-forward neural network (FFNN) architecture consists of two hidden layers connected by non-linear activation functions. The first hidden layer, which has 256 neurons, is coupled to the input layer. The second hidden layer, which includes 128 neurons, is connected to the first hidden layer. The output layer, whose size is determined by the number of classes in the problem, is then connected to the second hidden layer.

Here is a visual representation of the architecture:

''' Input Layer (Size depends on input data) --> Hidden Layer 1 (Size: 256) with Non-linearity ---> Hidden Layer 2 (Size: 128) with Non-linearity ---> Output Layer (Size depends on the number of classes) '''

There are several prominent options for non-linearity in this architecture, including the Rectified Linear Unit (ReLU), the hyperbolic tangent (tanh), and the Gaussian Error Linear Unit (GELU).

The binary cross-entropy loss function is frequently used for binary classification issues. The categorical cross-entropy loss function is frequently used for multi-class classification issues.

Feed-Forward There are various uses for neural networks:

```
They are able to simulate intricate non-linear input–output interactions.
They are relatively easy to comprehend and use.
They can be trained well on sizable datasets using methods like backpropagation and stochastic gradient descent.
They can be used to solve many different types of issues, like as classification, regression, and prediction.
```

However, because FFNNs do not account for the temporal correlations between inputs, they are only partially capable of handling sequential or time-series data. Additionally, if the dataset is too little or the model is too complicated, they are more likely to overfit.

Thanking You So Much