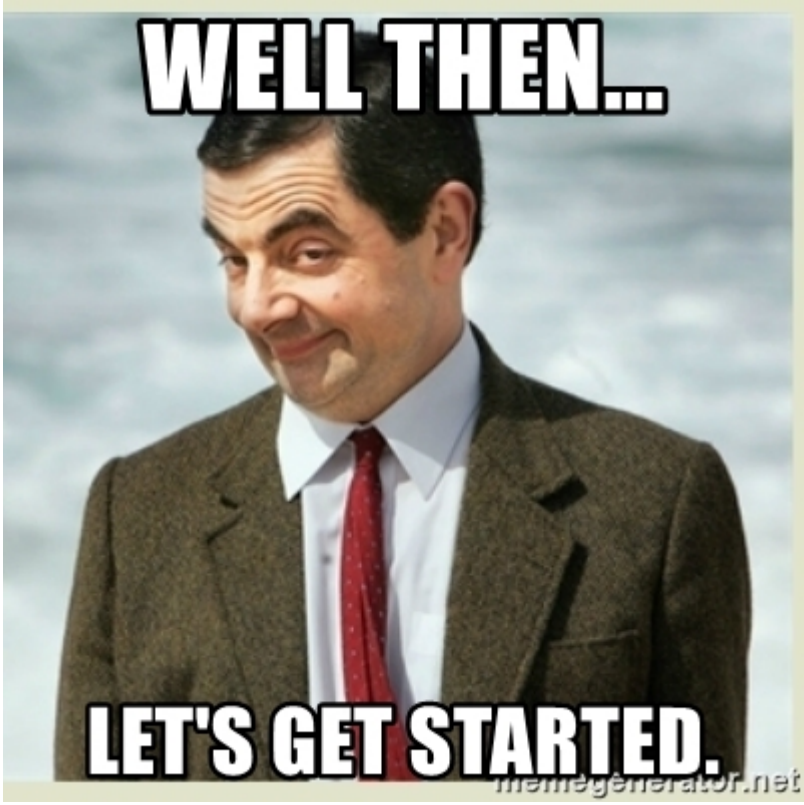
NLP Assignment 4: ML based NLP Project
Sentiment Analysis

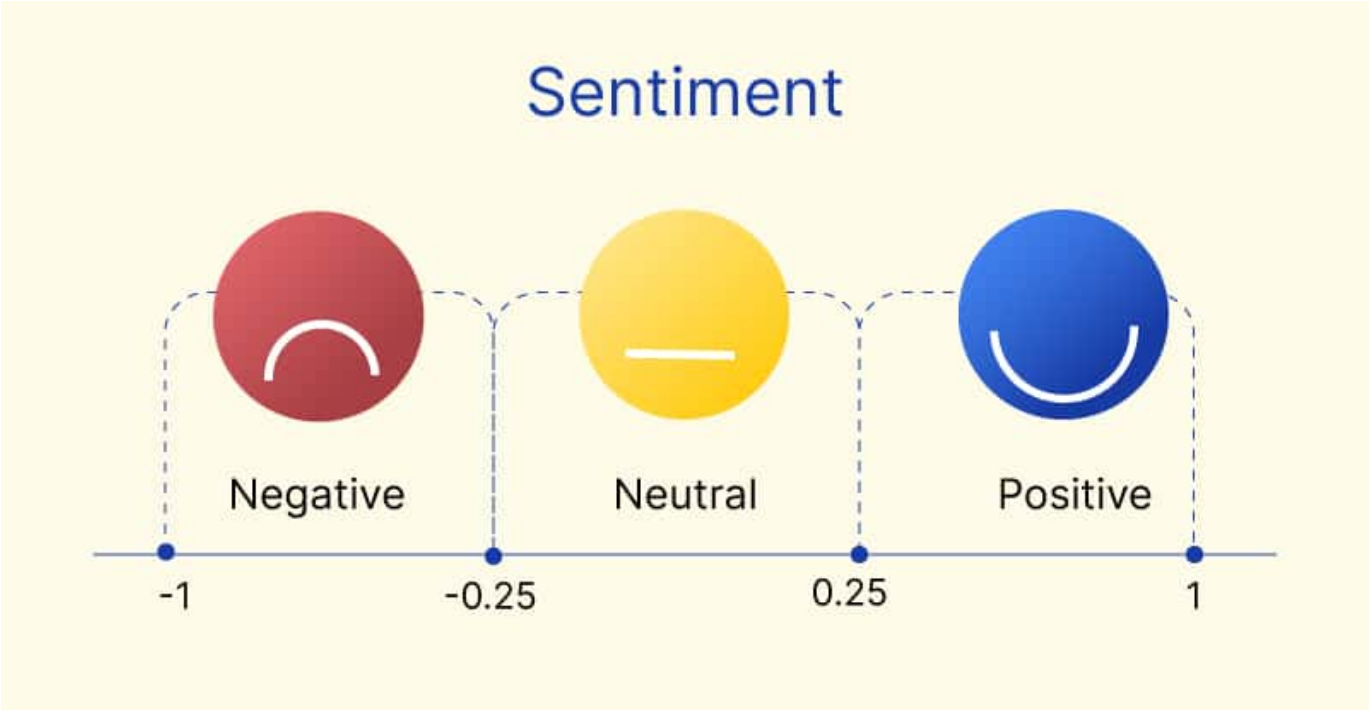
Welcome to the Assignment-4 of the NLP series!!

Keep your devices ready as you have a lot of practical work in this assignment. Are you ready to get your hands dirty into code??



Sentiment Analysis

This assignment will guide you through a machine learning project on a NLP use case. The aim behind doing this project is to let you understand how NLP problems are solved using machine learning approach. So that further when we solve them using deep learning, you get to know the major differences between the approaches and which one is better to solve these use cases. Sentiment Analysis is the most common text classification tool that analyses an incoming message and tells whether the underlying sentiment is positive, negative or neutral. This is the most common use case in NLP that has been studied,applied and solved by many experts using easy as well as state-of-the-art approaches.



Prerequisite: You should have completed all the previous assignments to enter into this project as some of the concepts from previous assignments have been applied here. Also some Basic knowledge of Logistic regression implementation is required here.
if you don't know about Logistic regression,Kindly watch the video given below.

```
#### Refer Video
from IPython.display import YouTubeVideo
YouTubeVideo('VCJdg7YBbAQ', width=600, height=300)
```

Problem to be solved: Given a dataset containing some text related to a movie, the problem is to predict the sentiment behind the statement in the form of 0 and 1 label (0 for negative and 1 for positive)

Let's first begin with importing all the necessary libraries.Quickly go through the purpose of the libraries.

1. **Pandas**- for storing and analysing the data.
2. **CountVectorizer**-to get the frequency of words.
3. **re**-for locating and matching patterns in text.
4. **PorterStemmer**-for stemming words.
5. **nltk**-for accessing other packages like PorterStemmer and stopwords.
6. **stopwords**-for removal os stopwords.
7. **WordCloud**-for generating word cloud showing words according to their frequency in the text.
8. **matplotlib**-for plotting some necessary graphs.
9. **TfidVectorizer**-to generate word vectors based on the document weightage.

```
#importing pandas library

#import count vectorizer function from sklearn

#importing re library

#Importing stemmer function from NLTK library

#Importing NLTK library and stopwords.

#Importing Word cloud

#Importing matplotlib library to plot pie chart.

#importing tfiVectorizer from sklearn for feature extraction.
```

The dataset used in the project is present in the google drive. So the drive has to be first mounted to use the dataset. For that we use the drive library from google.colab and then mount the drive using mount function.

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

After importing all the libraries, let's download the stopwords from nltk and load the english language stopwords in a variable using the words method from the stopwords class.

```
#downloading stopwords from nltk
```

```
#loading englsih stopwords in the 'stop' variable
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In the next cell,

1. porter(any name can be chosen) object of the PorterStemmer class is created as object is an instance of the class that can implement all the methods and functionalities of the class.
2. count(any name can be chosen) object of the CountVectorizer class is created as object is an instance of the class that can implement all the methods and functionalities of the class.

```
#creating objects of PorterStemmer and CountVectorizer class
```

Next, we load the data into the 'data' variable using the 'read_csv' method from pandas. Don't forget to download the trainig data and upload it into the project location before implementing the project. Also we print the first five rows of the dataset using the 'head' method.

```
#Loading the data to the variable as DataFrame
data=pd.read_csv("/content/drive/MyDrive/NLP_CLoudyML/NLP_Cloudy/Assignment 4/Train.csv")
data.head()
```

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1

In the next part, we segregate the data based on the positive and negative labels. Text with positive label is stored in 'pos' variable and text with negative label is stored in 'neg' variable.

```
#finding the positive Data
pos=data[data['label']==1]
print("Positive text \n",pos.head())
#try for negative Data
```

```
Positive text
```

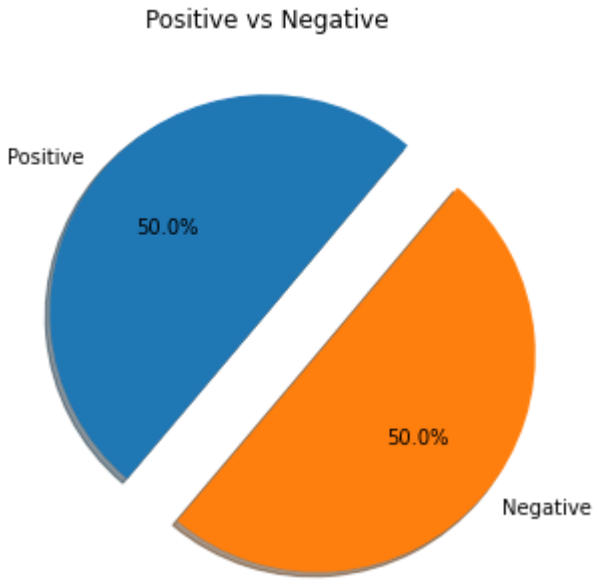
```
                                                  text  label
4   Im a die hard Dads Army fan and nothing will e...      1
6   Finally watched this shocking movie last night...      1
8   It may be the remake of 1987 Autumn's Tale aft...      1
9   My Super Ex Girlfriend turned out to be a plea...      1
10  I can't believe people are looking for a plot ...      1

Negative text
                                                  text  label
0  I grew up (b. 1965) watching and loving the Th...      0
1  When I put this movie in my DVD player, and sa...      0
2  Why do people who do not know what a particula...      0
3  Even though I have great interest in Biblical ...      0
5  A terrible movie as everyone has said. What ma...      0
```

Here we can see that we have both postive and negative text in the data or dataset. So we can visulize the data by using matplotlib.

Now we plot a pie chart of the positive and negative data to understand the ratio of the data.
Matplotlib library is used to plot the pie chart using various parameters.
Do refer to the given documentation to know more about the used parameters:
https://matplotlib.org/stable/gallery/pie_and_polar_charts/pie_features.html

```
#Plotting the Postive vs Negative in piechart.
```

```
Text(0.5, 1.0, 'Positive vs Negative')
```



We can see that the percentage of positive and negative data is equal i.e. 50% each. Which means that there is enough training data of both the classes.

Next, we define a preprocessing function using regular expressions to clean the data and remove unnecessary things in the text.
Any of the html tags are represented by '<[^>]*>' and replaced with a whitespace. Non word

characters are replaced with a whitespace and the text is converted into lowercase. Emoji symbols in the text are joined at the end of the text by replacing the '-' in the emoji with a whitespace.

Do refer to the given documentation to know more about Regular expressions:
https://docs.python.org/3/library/re.html

```
[ ]  #Defining preprocessing function to process the data
     def preprocess(text):
             text=re.sub('<[^>]*>','',text)
             emoji=re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)',text)
             text=re.sub('[\W]+',' ',text.lower()) +' '.join(emoji).replace('-','')
             return text
```

Write the above code in below cell for preprocessing

```
#Defining preprocessing function to process the data
```

The above defined function is now applied to each text in the row using the apply function in python.

```
#Applying the function preprocess on the data
```

Lets display the top 5 rows of the preprocessed data to check exactly what changes have been made. We use the head function of pandas for this.

```
#Displaying the dataframe after applying the preprocessing using head().
```

	text	label
0	i grew up b 1965 watching and loving the thund...	0
1	when i put this movie in my dvd player and sat...	0
2	why do people who do not know what a particula...	0
3	even though i have great interest in biblical ...	0
4	im a die hard dads army fan and nothing will e...	1

As we can see, all the text is converted into lowercase and other unnecessary stuff is removed.

Next, a function tokenize is defined which returns the spillted form of the text. The split function from strings is used for splitting the text into tokens. A sample sentence has been passed into the tokenizer function that we created to check the output.

```
#Defining a function called tokenizer which splits the sentence
def tokenizer(text):
  #your code here for splitting
```

Here, a function tokenizer_porter is defined that stems the splitted words using the stem function from PorterStemmer class.

```
#Defining function for Tokenizer porter
def tokenizer_porter(text):
    #your code for here for porter
```

Ahead we segregate the processed data into positive and negative data based on the labels.

```
#getting positive
positive_data = data[ data['label'] == 1]
positive_data = positive_data['text']
#try to get negative data
```

Now we define a function that will plot the word cloud for our data. The words 'movie' and 'film' are ignored for cleaning the words. Next the wordcloud is formed by giving various parameters like stopwords, background colour, etc. Refer to the given documentation: https://www.analyticsvidhya.com/blog/2020/10/word-cloud-or-tag-cloud-in-python/

```
#Defining the function to plot the data in wordcloud
def plot_wordcloud(data, color = 'white'):
  #your code for plotiing in wordcloud
```

Now that we have the word cloud plotting function, we will plot the word cloud for the positive data using white background colour.

```
#Printing the positive data in wordcloud
#print("Positive words")
plot_wordcloud(positive_data,'white')
```

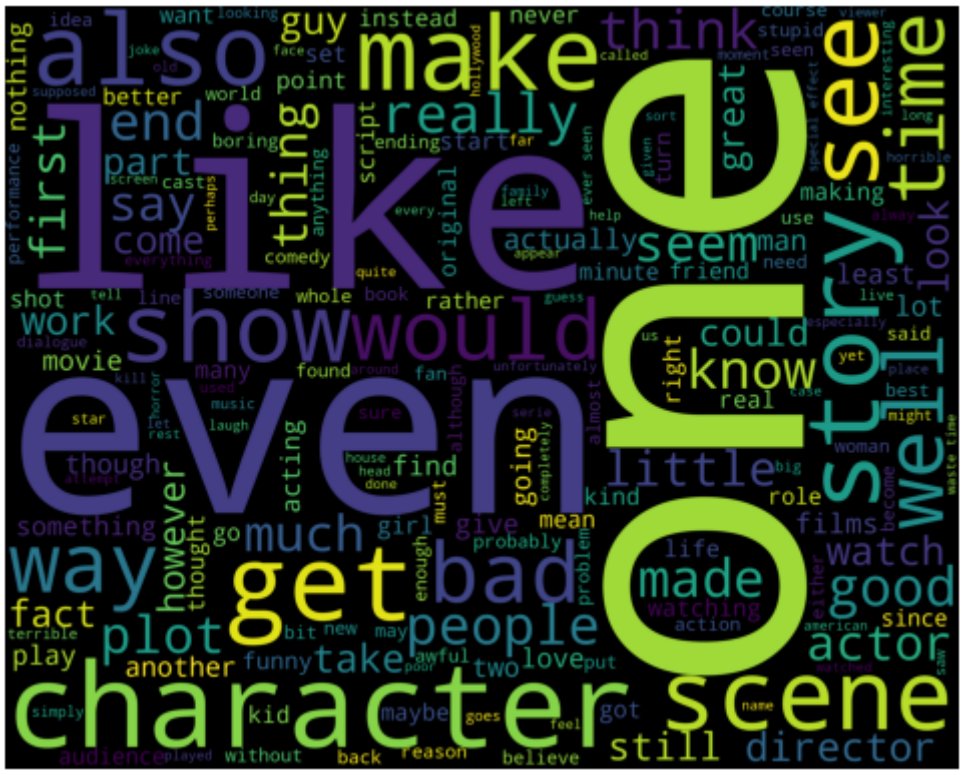
Positive words

The word cloud is plotted and we see that the most occuring words are one, like, character. Other words can also be seen with different frequencies.

Next, we will plot the word cloud for the negative data using black background colour.

```
#Printing the negative data in wordcloud
# try for negative words
```

Negative words



After plotting we see that words like even,one,like,character,get,scene,etc have higher frequencies of appearing in the text.

Next, tfid object of the TfidVectorizer class is created by giving various parameters. The tokenizer_porter function that we defined earlier has been passed as the tokenizer in this vectorizer. Know more about the parameters at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
#using TfidVectorizer

#assigning values in y i.e label.values
```

Next, the data is scaled using fit_transform function into the x variable which means that vectorization has been applied to the data and stored into the x variable which will be further used for training.

```
#scaling the data
```

It is time to split our data as it is now ready for further processing. train_test_split from sklearn helps us to do that. It will split the data into training and testing data so that we can check the model accuracy easily by referring or comparing with the original values in tha dataset. test size is taken as 0.5 which means that 50% of the data is for training and remaining 50% for testing. The splitted data is stored in four different variables.
X_train: independent training data
X_test: independent testing data
y_train: prediction of the training data (labels in this case)
y_test: prediction of the testing data(labels in this case)

```
 #splitting the train and test split using train_test_split function of sklearn
```

Now we import the LogisticRegressionCV module from sklearn which will be used to train the model. This regressor comes with cross validation so that there is no need to implement cross validation seperately. A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables. For example, a logistic regression could be used to predict whether a political candidate will win or lose an election or whether a high school student will be admitted to a particular college. Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations.

Refer the given documentation to know more about LogisticRegressionCV: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html

A model is created by passing various parameters. Number of folds in cross validation is given as 6, scoring is estimated based on accuracy and maximum iterations are set to 500 (you may try out with some other values). The model is then fitted on the training data and further predicted on the test data(X_train) to get y_pred which is the predicted labels.

```
#Importing Logisitic RegressionCV from sklearn library

#create model of LogisticRegression of cv=6,scoring of accuracy and max_iter=500

#fiting our model

#making predication using predict
```

Wohoo!! our model is trained. Its time to check how well it has learnt everything. For that we use accuracy_score from metrics package of the sklearn library. The predicted labels and actual test labels are passed as parameters to get the accuracy score.

```
#Importing metrics from sklesrn to calculate accuracy

# Accuracy of our built model


    Accuracy of our model: 0.8922
```

That's cool, 89% is the accuracy of our model.
We have come to an end of this project but don't stop here, try as many projects of the similar type to get a better understanding of the use cases. Solve the practice sheet of this project to test yourself.!!

FEEDBACK FORM

https://forms.zohopublic.in/cloudyml/form/CloudyMLDeepLearningFeedbackForm/formperma/VCFbldnXAnbcgAIl0lWv2blgHdSldheO4RfktMdgK7s