Achintya Agarwal

102115126

Twitter Sentiment Analysis

Sentiment analysis, also known as opinion mining, is the process of analyzing text data to determine the sentiment expressed within it. The goal of sentiment analysis is to automatically classify text into categories such as positive, negative, or neutral based on the sentiment conveyed by the language used in the text. It's a subfield of natural language processing (NLP) and text analytics that has numerous applications across various domains.

Here's how sentiment analysis typically works:

Text Preprocessing: The text data is preprocessed to remove noise, including punctuation, special characters, and stopwords (commonly used words like "and", "the", "is", etc.). It may also involve techniques like stemming or lemmatization to reduce words to their base form.

Feature Extraction: Features are extracted from the preprocessed text data. These features could include word frequency counts, n-grams (sequences of adjacent words), part-of-speech tags, or word embeddings (dense vector representations of words).

Sentiment Classification: Machine learning algorithms or statistical models are trained on labeled datasets to classify text into sentiment categories. Commonly used algorithms include logistic regression, support vector machines (SVM), naive Bayes, decision trees, and deep learning models like recurrent neural networks (RNNs) or convolutional neural networks (CNNs).

Model Evaluation: The trained model is evaluated on a separate validation or test dataset to assess its performance in classifying sentiment accurately. Evaluation metrics such as accuracy, precision, recall, F1-score, or area under the ROC curve (AUC-ROC) are commonly used to measure performance.

Deployment and Application: Once the model is trained and evaluated, it can be deployed to analyze sentiment in real-time on new, unseen text data. Sentiment analysis finds applications in various domains, including customer feedback analysis, social media monitoring, brand reputation management, market research, and product sentiment analysis.

Sentiment analysis can provide valuable insights by automatically processing and summarizing large volumes of text data, allowing organizations to understand public opinion, customer satisfaction, trends, and sentiments towards products, services, or topics of interest. It's widely used in business intelligence, customer relationship management, social media analytics, and other areas where understanding sentiment is crucial for decision-making.

About Data

The data is a CSV with emoticons removed. Data file format has 6 fields:

0 - the polarity of the tweet (0 = negative, 4 = positive)

1 - the id of the tweet (2087)

2 - the date of the tweet (Sat May 16 23:58:44 UTC 2009)

3 - the query (lyx). If there is no query, then this value is NO_QUERY.

4 - the user that tweeted (robotickilldozr)

5 - the text of the tweet (Lyx is cool)

training data was automatically created, as opposed to having humans manual annotate tweets. In my approach, I assume that any tweet with positive emoticons, like :), were positive, and tweets with negative emoticons, like :(, were negative.

In [2]:

```
import nltk
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from nltk.corpus import stopwords
from string import punctuation
from nltk.tokenize import word_tokenize
from nltk.stem import LancasterStemmer
from string import punctuation
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import LancasterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
import re
```

Dataset

In [3]:
```
df = pd.read_csv('training.1600000.processed.noemoticon.csv',
                 delimiter=',', encoding='ISO-8859-1')
df.columns = ['Sentiment','id','date','query','user','text']
df.head()
```

Out[3]:

	Sentiment	id	date	query	user	text
0	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
1	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
2	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
3	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....
4	0	1467811372	Mon Apr 06 22:20:00 PDT 2009	NO_QUERY	joy_wolf	@Kwesidei not the whole crew

In [4]:
```
df = df[['Sentiment','text']]
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Sentiment', 'text'], dtype='object')
```

In [6]:
```
df.Sentiment.value_counts()
```

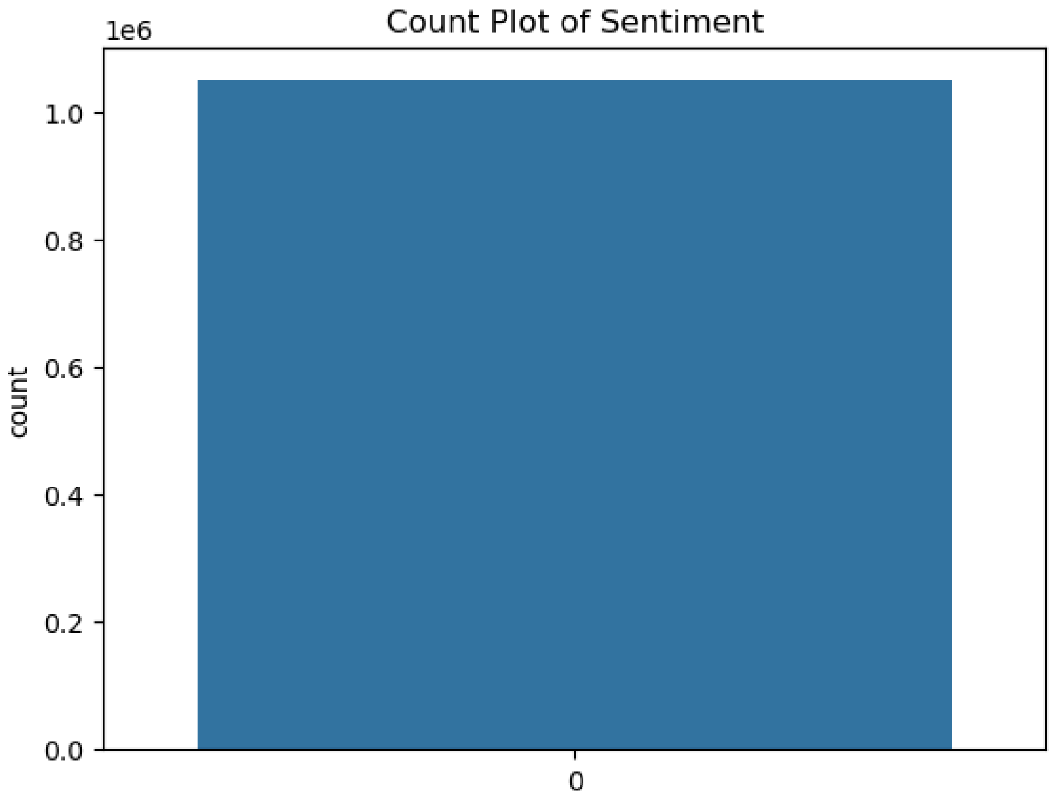
Out[6]:
```
Sentiment
0    799996
4    248576
Name: count, dtype: int64
```

In [7]:
```
df['Sentiment'] = df['Sentiment'].replace({4:1})
```

- 0 represent Negative sentiment
- 1 represents Positive sentiment

Visualizing the count

In [8]:
```
sns.countplot(df["Sentiment"])
plt.title("Count Plot of Sentiment")
plt.show()
```



In [9]:
```
df.isna().sum().sum()
```

Out[9]: 0

Inference: The data is unbalanced therfore we will downsample the data to have same count for each sentiment

Downsampling the dataset

In [10]:
```
from sklearn.utils import resample
```

In [11]:
```
## majority class 0
df_majority = df[df['Sentiment']==0]
## minority class 1
df_minority = df[df['Sentiment']==1]
```

In [12]:
```
df_minority.shape
```

Out[12]: (248576, 2)

In [13]:

```
# downsample the majority class
df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples=len(df_minority),
                                   random_state=1234)
```

In [14]:

```
df = pd.concat([df_majority_downsampled, df_minority])
df.head()
```

Out[14]:

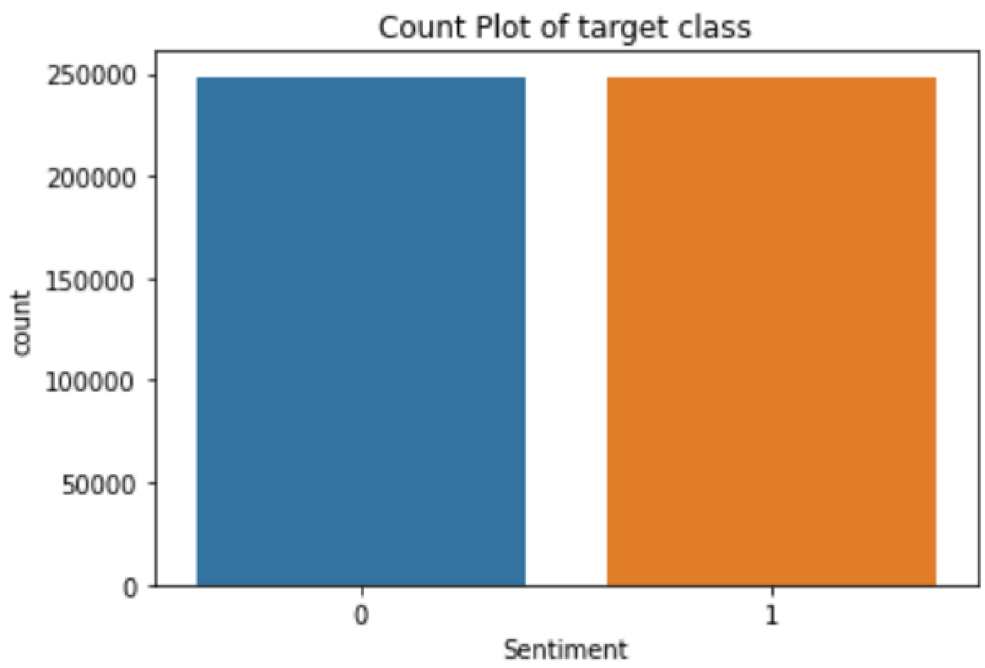
	Sentiment	text
74567	0	Wow slept for almost 12hours. Sleepy me!! Uni ...
668722	0	gets bored with an idea too easily ... like tw...
286706	0	To my girls - sorry i've been a homebody latel...
632911	0	BK once again for the weekend...If it wasnt fo...
356735	0	@DonnieWahlberg Now why didn't you do that las...

Visualizing after downsampling

248576 data for each class

In [ ]:

```
# Plot count plot of sentiment
sns.countplot(df["Sentiment"])
plt.title("Count Plot of target class")
plt.show()
```



Data Preprocessing

1. removing stop words
2. removing punctuations

3. Lemmatizing
4. removing tags
5. removing special characters
6. lowercase conversion

In [16]:
```
## remove stopwords and punctuation marks
stuff_to_be_removed = list(stopwords.words('english'))+list(punctuatior
stemmer = LancasterStemmer()

corpus = df['text'].tolist()
print(len(corpus))
print(corpus[0])
```

```
497152
Wow slept for almost 12hours. Sleepy me!! Uni now, boo! I wanna stay h
ome, drink tea and watch house...
```

In [17]:
```
final_corpus = []
final_corpus_joined = []
for i in df.index:

    text = re.sub('[^a-zA-Z]', ' ', df['text'][i])
    #Convert to lowercase
    text = text.lower()
    #remove tags
    text=re.sub("&lt;/?.*?&gt;"," &lt;&gt; ",text)

    # remove special characters and digits
    text=re.sub("(\\d|\\W)+"," ",text)

    ##Convert to list from string
    text = text.split()

    #Lemmatisation
    lem = WordNetLemmatizer()
    text = [lem.lemmatize(word) for word in text
            if not word in stuff_to_be_removed]
    text1 = " ".join(text)
    final_corpus.append(text)
    final_corpus_joined.append(text1)
```

Storing the cleaned data seperately

In [18]:
```
data_cleaned = pd.DataFrame()
data_cleaned["text"] = final_corpus_joined
data_cleaned["Sentiment"] = df["Sentiment"].values
```

In [19]:
```
data_cleaned['Sentiment'].value_counts()
```

Out[19]:
```
Sentiment
0    248576
1    248576
Name: count, dtype: int64
```

In [20]:
```
data_cleaned.head()
```

Out[20]:

	text	Sentiment
0	wow slept almost hour sleepy uni boo wanna sta...	0
1	get bored idea easily like twitter	0
2	girl sorry homebody lately dont feel well does...	0
3	bk weekend wasnt puppy stay as	0
4	donniewahlberg last night atlanta	0

EDA

In [21]:
```
data_eda = pd.DataFrame()
data_eda['text'] = final_corpus
data_eda['Sentiment'] = df["Sentiment"].values
data_eda.head()
```

Out[21]:

	text	Sentiment
0	[wow, slept, almost, hour, sleepy, uni, boo, w...	0
1	[get, bored, idea, easily, like, twitter]	0
2	[girl, sorry, homebody, lately, dont, feel, we...	0
3	[bk, weekend, wasnt, puppy, stay, as]	0
4	[donniewahlberg, last, night, atlanta]	0

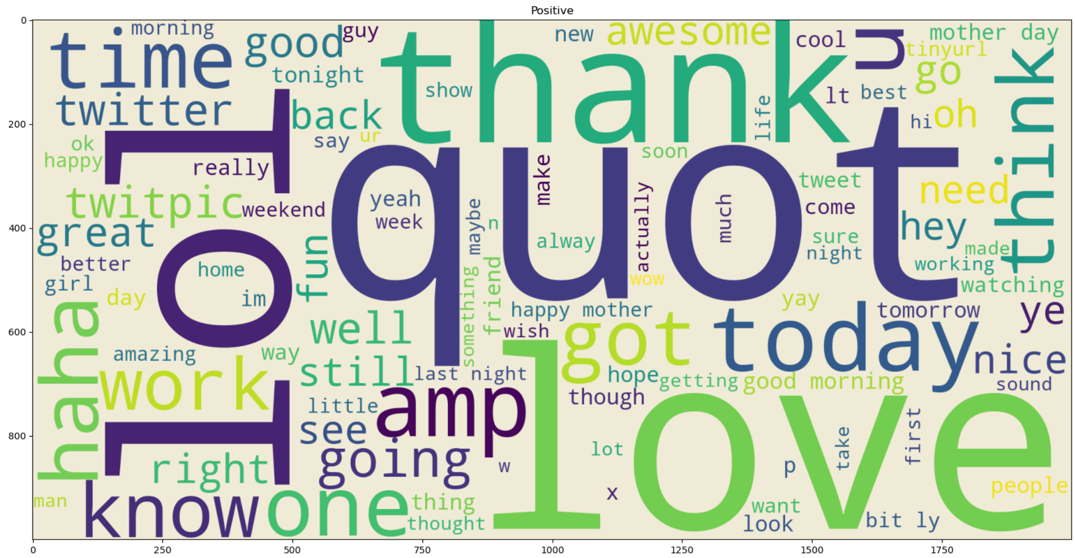
In [22]:
```
# Storing positive data seperately
positive = data_eda[data_eda['Sentiment'] == 1]
positive_list = positive['text'].tolist()

# Storing negative data seperately

negative = data_eda[data_eda['Sentiment'] == 0]
negative_list = negative['text'].tolist()
```

In [23]:
```
positive_all = " ".join([word for sent in positive_list for word in ser
negative_all = " ".join([word for sent in negative_list for word in ser
```

Word Cloud Positive data

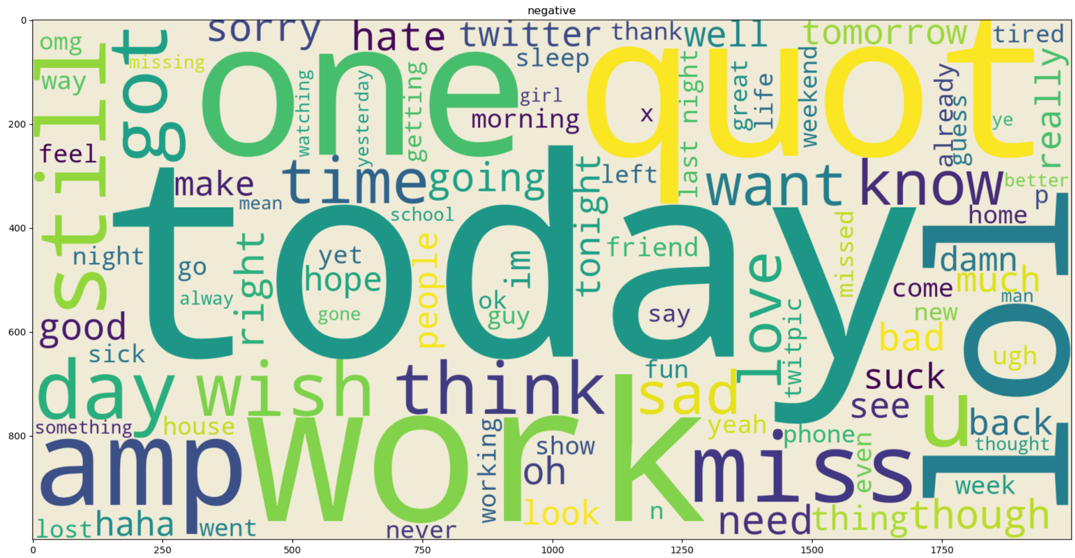
In [24]:

```
from wordcloud import WordCloud
WordCloud()
wordcloud = WordCloud(width=2000,
                      height=1000,
                      background_color='#F2EDD7FF',
                      max_words = 100).generate(positive_all)

plt.figure(figsize=(20,30))
plt.imshow(wordcloud)
plt.title("Positive")
plt.show()
```



Word CLoud Negative data

In [25]:
```
from wordcloud import WordCloud
WordCloud()
wordcloud = WordCloud(width=2000,
                      height=1000,
                      background_color='#F2EDD7FF',
                      max_words = 100).generate(negative_all)

plt.figure(figsize=(20,30))
plt.imshow(wordcloud)
plt.title("negative")
plt.show()
```



Inference:

- Positive data has words like Thank, love , LOL, Haha ets
- Negative data has words like work, sad, tired, suck sorry
  Some of the words are still common in both such as Lol, quot

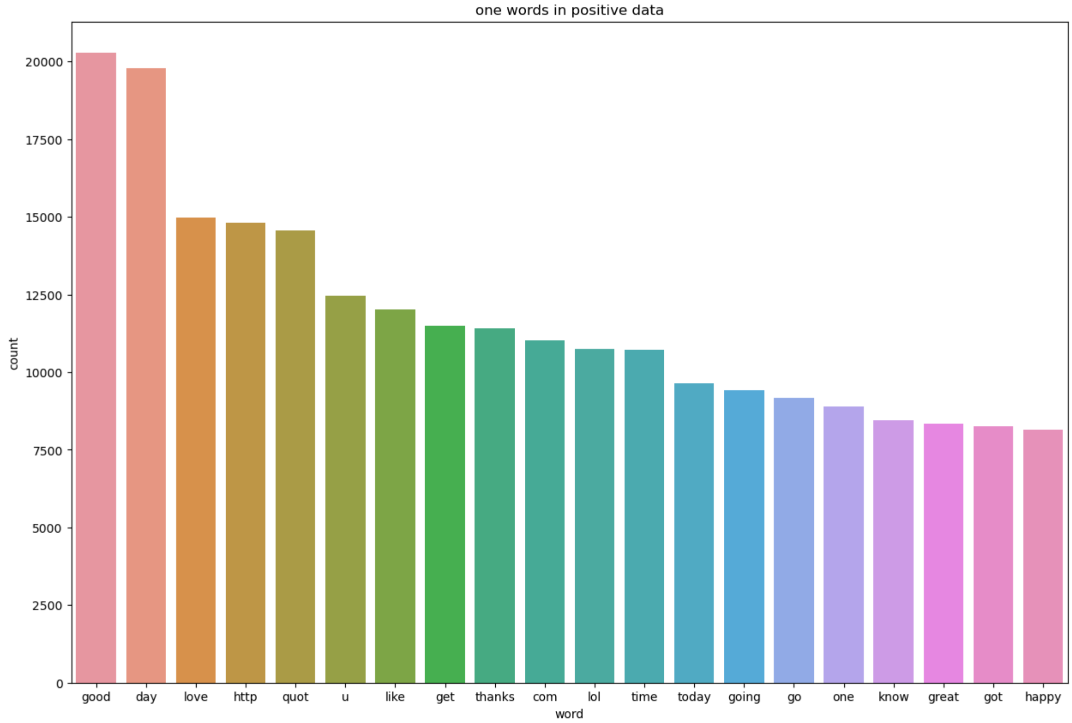
One word count

In [26]:
```
def get_count(data):
    dic = {}
    for i in data:
        for j in i:
            if j not in dic:
                dic[j]=1
            else:
                dic[j]+=1

    return(dic)
count_corpus = get_count(positive_list)
```

In [27]:
```
count_corpus = pd.DataFrame({"word":count_corpus.keys(),"count":count_c
count_corpus = count_corpus.sort_values(by = "count", ascending = False
```

In [28]:

```
import seaborn as sns
plt.figure(figsize = (15,10))
sns.barplot(x = count_corpus["word"][:20], y = count_corpus["count"][:2
plt.title('one words in positive data')
plt.show()
```



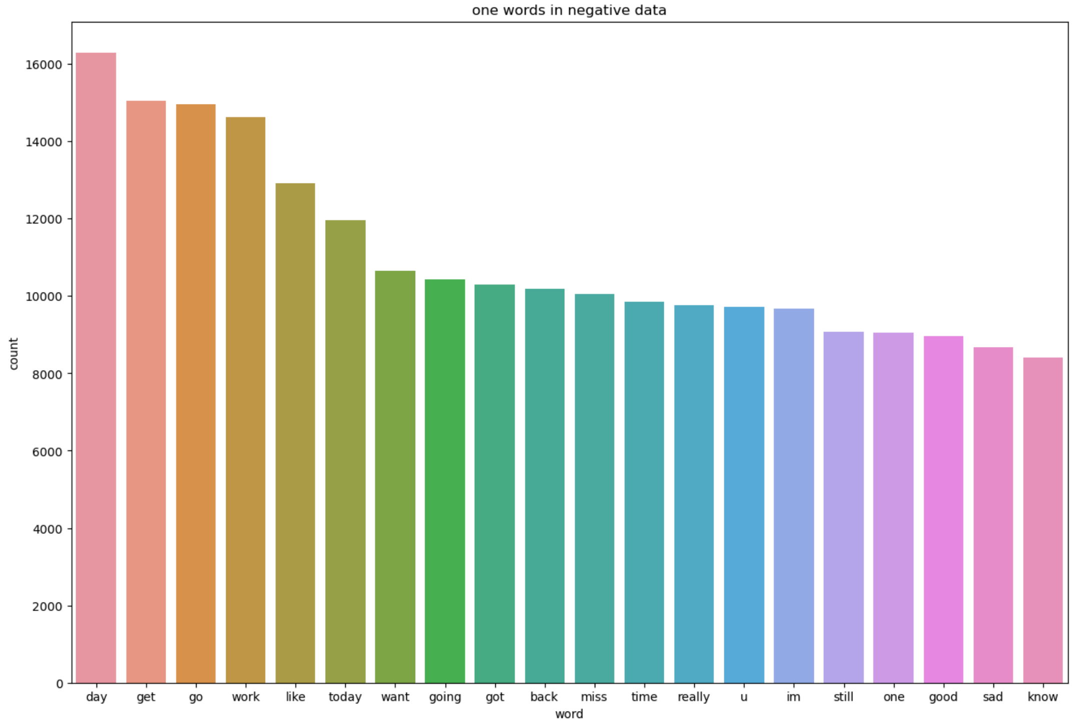
In [29]:

```
def get_count(data):
    dic = {}
    for i in data:
        for j in i:
            if j not in dic:
                dic[j]=1
            else:
                dic[j]+=1

    #dic = dict(sorted(dic.items() , key = lambda x:x[1],reverse=True))
    return(dic)
count_corpus = get_count(negative_list)
```

In [30]:

```
count_corpus = pd.DataFrame({"word":count_corpus.keys(),"count":count_c
count_corpus = count_corpus.sort_values(by = "count", ascending = False
```

In [31]:
```
import seaborn as sns
plt.figure(figsize = (15,10))
sns.barplot(x = count_corpus["word"][:20], y = count_corpus["count"][:2
plt.title('one words in negative data')
plt.show()
```



Inference

- Positive data has words like good, day, thanks, great, happy
- Negative data has words like work, miss, sad etc

Classification

Naive Bayes (NB): Naive Bayes is a simple yet effective probabilistic classifier based on Bayes' theorem with the "naive" assumption of feature independence. Despite its simplicity, Naive Bayes often performs well in text classification and other tasks. It calculates the probability of each class given a set of features and selects the class with the highest probability. Naive Bayes classifiers are easy to train, computationally efficient, and require relatively few training data. However, the assumption of feature independence may not hold true in all cases, which can lead to suboptimal performance in some scenarios.

Naive bayes for sentiment analysis

In [32]:
```
def get_tweets_for_model(cleaned_tokens_list):
    for tweet_tokens in cleaned_tokens_list:
        yield dict([token, True] for token in tweet_tokens)

positive_tokens_for_model = get_tweets_for_model(positive_list)
negative_tokens_for_model = get_tweets_for_model(negative_list)
```

In [33]:
```
import random

positive_dataset = [(review_dict, "Positive")
                     for review_dict in positive_tokens_for_model]

negative_dataset = [(review_dict, "Negative")
                     for review_dict in negative_tokens_for_model]
dataset = positive_dataset + negative_dataset

random.shuffle(dataset)

train_data = dataset[:333091]
test_data = dataset[333091:]
```

In [34]:

```
from nltk import classify
from nltk import NaiveBayesClassifier
classifier = NaiveBayesClassifier.train(train_data)

print(" Training Accuracy is:", round(classify.accuracy(classifier, tra

print("Testing Accuracy is:", round(classify.accuracy(classifier, test_

print(classifier.show_most_informative_features(10))
```

```
 Training Accuracy is: 86.0
Testing Accuracy is: 77.0
Most Informative Features
                    iran = True           Negati : Positi =     53.7 :
1.0
             squarespace = True           Negati : Positi =     53.3 :
1.0
                  farrah = True           Negati : Positi =     41.3 :
1.0
                  booooo = True           Negati : Positi =     32.9 :
1.0
                 saddest = True           Negati : Positi =     32.9 :
1.0
                hotwords = True           Positi : Negati =     30.4 :
1.0
                 unhappy = True           Negati : Positi =     27.7 :
1.0
                hayfever = True           Negati : Positi =     27.2 :
1.0
            followfriday = True           Positi : Negati =     27.1 :
1.0
              devastated = True           Negati : Positi =     26.3 :
1.0
None
```

TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF is a numerical statistic that reflects the importance of a term in a document relative to a collection of documents. It's commonly used in text mining and information retrieval to represent the significance of terms in documents.

TFIDF for sentiment analysis

In [35]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
vector = tfidf.fit_transform(data_cleaned['text'])
y = data_cleaned['Sentiment']
```

In [36]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(vector,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=42,
                                                    stratify = y)
```

In [37]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy_score,confusion_matrix,classificat
from sklearn.linear_model import LogisticRegression
```

In [38]:

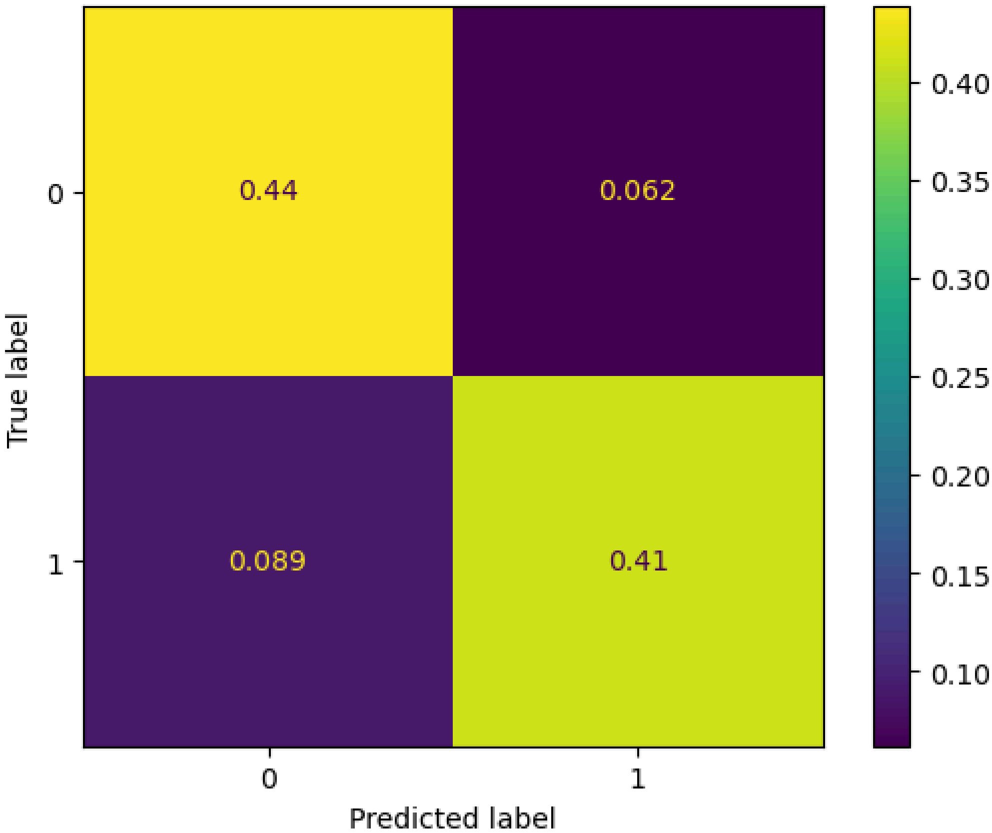
```
def metrics(y_train,y_train_pred,y_test,y_test_pred):
  print("training accuracy = ",round(accuracy_score(y_train,y_train_pre
  ConfusionMatrixDisplay.from_predictions(y_train,y_train_pred,normaliz
  print(classification_report(y_train,y_train_pred))
  plt.show()

  print("testing accuracy = ",round(accuracy_score(y_test,y_test_pred),
  ConfusionMatrixDisplay.from_predictions(y_test,y_test_pred,normalize
  print(classification_report(y_test,y_test_pred))
  plt.show()
```

Multinomial NB

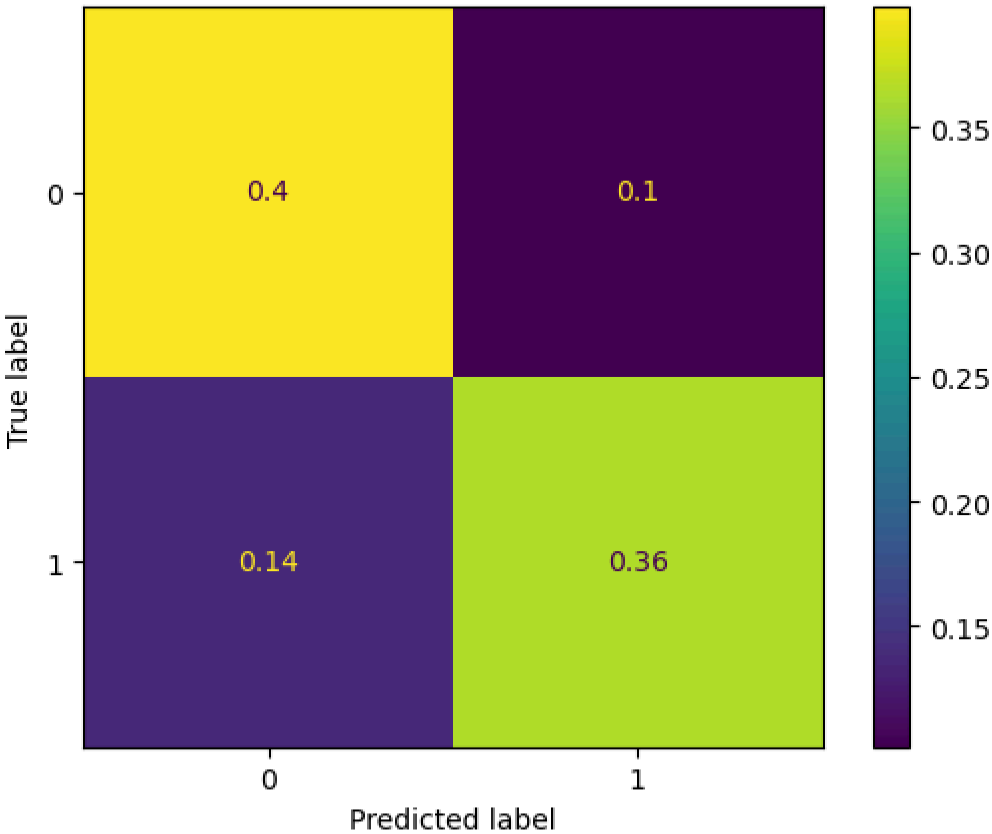
Multinomial Naive Bayes (Multinomial NB): Multinomial Naive Bayes is a variant of Naive Bayes specifically designed for text classification tasks where features represent word counts or term frequencies. It assumes that features follow a multinomial distribution and calculates the probability of each class given the frequency of each feature. Multinomial NB is widely used in document classification, spam filtering, and sentiment analysis. It handles large feature spaces efficiently and is robust to irrelevant features. However, like other Naive Bayes classifiers, Multinomial NB relies on the strong assumption of feature independence, which may not always hold true in practice.

In [39]:
```
NB = MultinomialNB()
NB.fit(X_train,y_train)
y_train_pred = NB.predict(X_train)
y_test_pred = NB.predict(X_test)
metrics(y_train,y_train_pred,y_test,y_test_pred)
```

```
training accuracy =  85.0
              precision    recall  f1-score   support

           0       0.83      0.88      0.85    166545
           1       0.87      0.82      0.85    166546

    accuracy                           0.85    333091
   macro avg       0.85      0.85      0.85    333091
weighted avg       0.85      0.85      0.85    333091
```




```
testing accuracy =  76.0
              precision    recall  f1-score   support

           0       0.75      0.80      0.77     82031
           1       0.78      0.73      0.75     82030

    accuracy                           0.76    164061
   macro avg       0.76      0.76      0.76    164061
weighted avg       0.76      0.76      0.76    164061
```



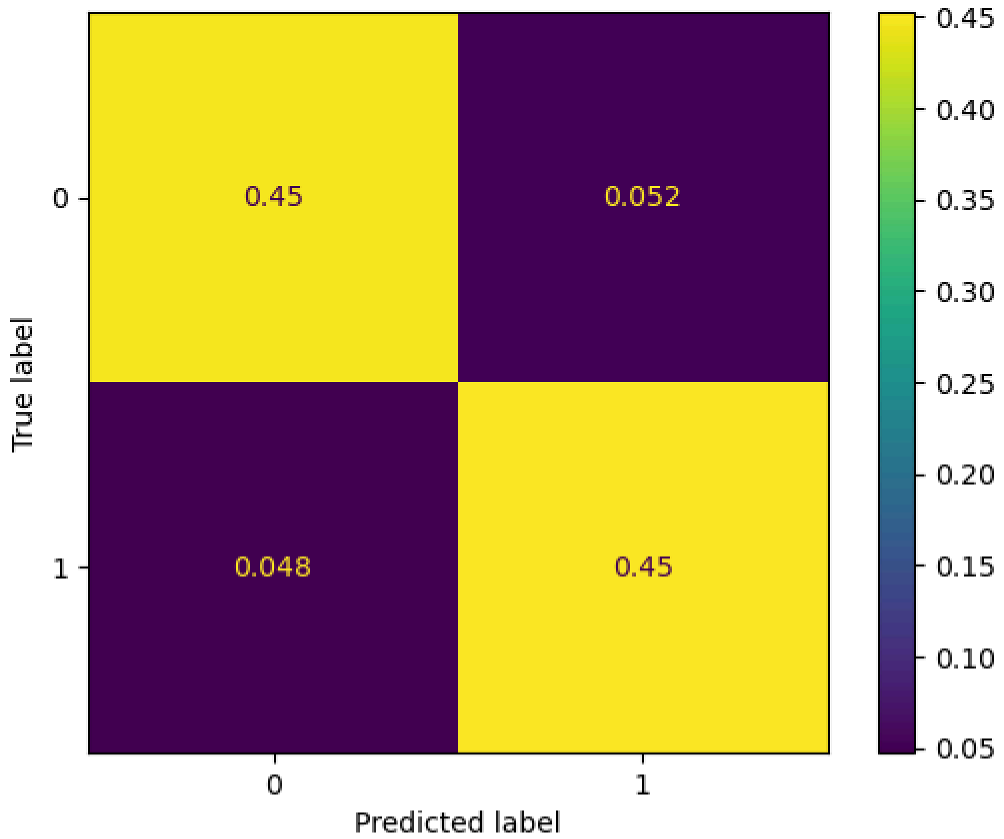
Linear SVC

Linear Support Vector Classification (Linear SVC): Linear SVC is a variant of Support Vector Machine (SVM) classifier that works by finding the hyperplane that best separates classes in the feature space. It is particularly effective for linearly separable datasets, where classes can be separated by a straight line. Linear SVC aims to maximize the margin between classes while minimizing classification errors. It is computationally efficient and scales well to large datasets. Linear SVC is commonly used for text classification, sentiment analysis, and other classification tasks. However, it may not perform well on datasets with complex non-linear relationships between features and the target variable.

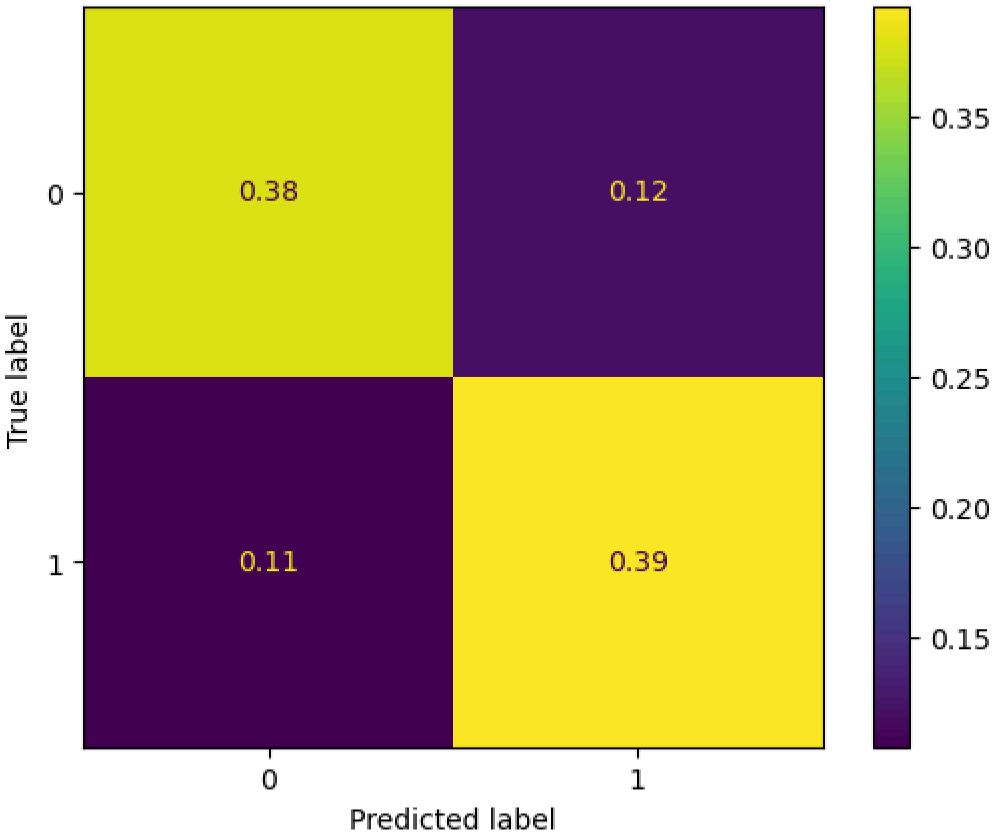
In [40]:

```
svc = LinearSVC()
svc.fit(X_train,y_train)
y_train_pred = svc.predict(X_train)
y_test_pred = svc.predict(X_test)
metrics(y_train,y_train_pred,y_test,y_test_pred)
```

```
training accuracy =  90.0
              precision    recall  f1-score   support

           0       0.90      0.90      0.90    166545
           1       0.90      0.90      0.90    166546

    accuracy                           0.90    333091
   macro avg       0.90      0.90      0.90    333091
weighted avg       0.90      0.90      0.90    333091
```




```
testing accuracy =  77.0
              precision    recall  f1-score   support

           0       0.78      0.76      0.77     82031
           1       0.76      0.78      0.77     82030

    accuracy                           0.77    164061
   macro avg       0.77      0.77      0.77    164061
weighted avg       0.77      0.77      0.77    164061
```

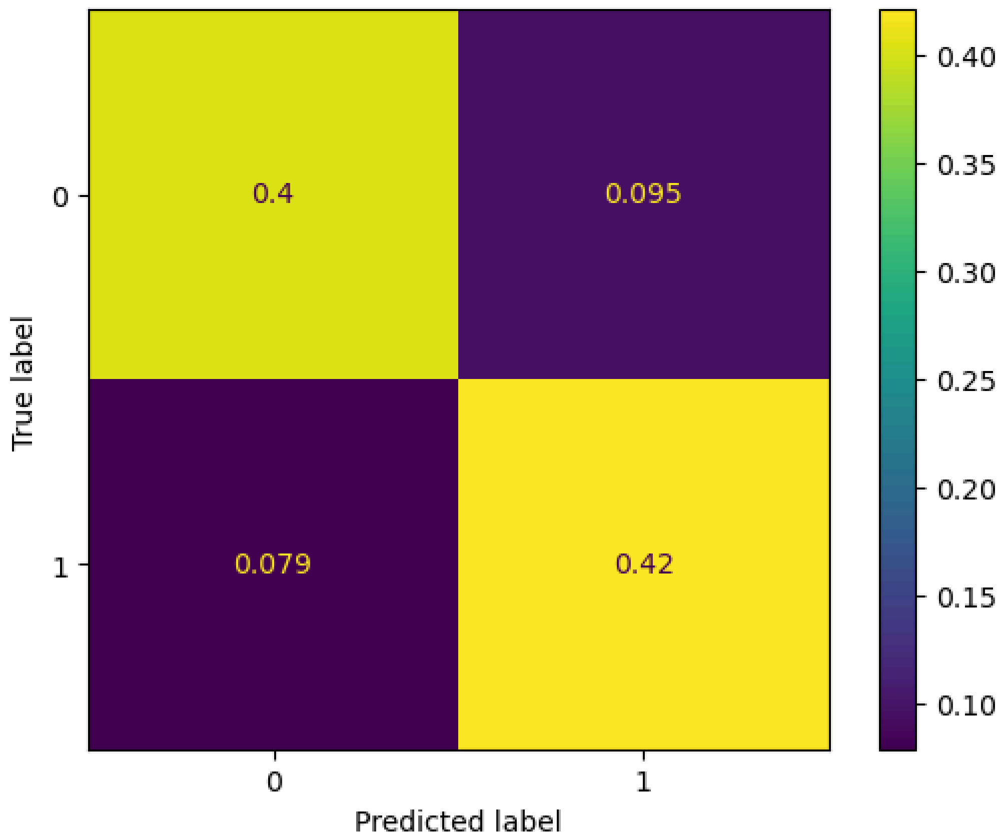


Logistic regression

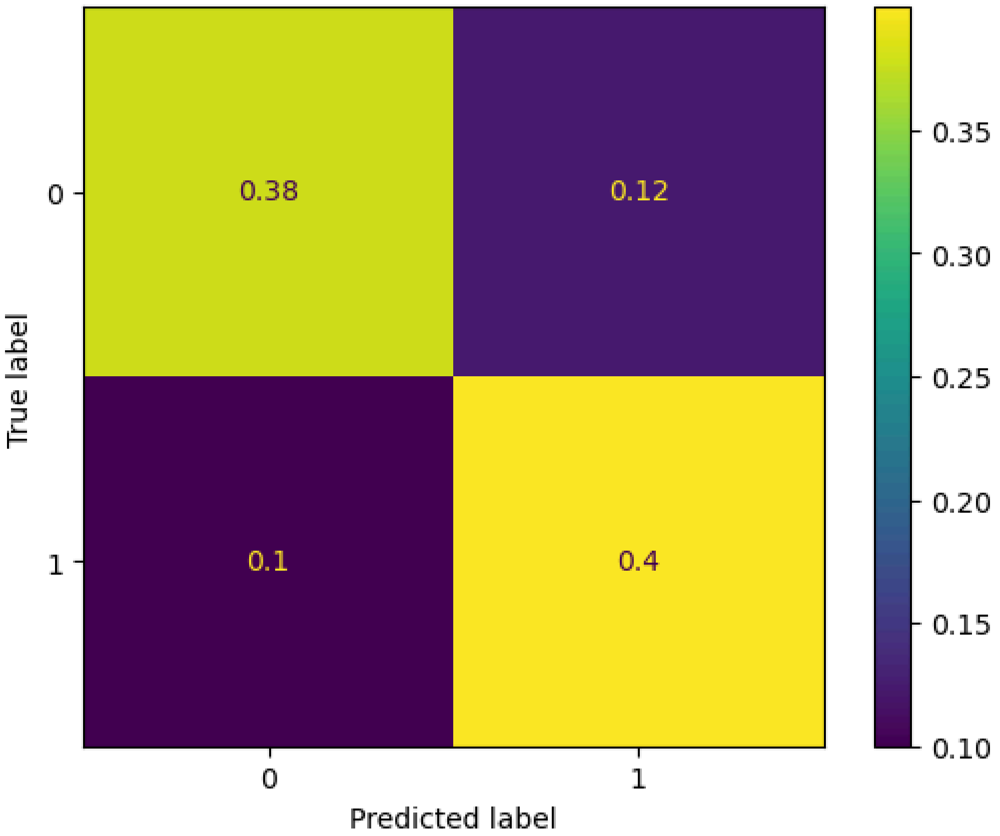
Logistic Regression: Logistic Regression is a linear classification model that predicts the probability of an instance belonging to a particular class. Despite its name, Logistic Regression is used for classification, not regression. It models the relationship between the independent variables (features) and the binary dependent variable (class labels) using the logistic function, also known as the sigmoid function. Logistic Regression produces interpretable coefficients for each feature, making it easy to understand the impact of individual features on the predicted probabilities. It is computationally efficient, scales well to large datasets, and is less prone to overfitting compared to more complex models. Logistic Regression is commonly used in binary classification tasks such as spam detection, medical diagnosis, and customer churn prediction.

In [41]:
```
lr = LogisticRegression()
lr.fit(X_train,y_train)
y_train_pred = lr.predict(X_train)
y_test_pred = lr.predict(X_test)
metrics(y_train,y_train_pred,y_test,y_test_pred)
```

```
training accuracy =  83.0
              precision    recall  f1-score   support

           0       0.84      0.81      0.82    166545
           1       0.82      0.84      0.83    166546

    accuracy                           0.83    333091
   macro avg       0.83      0.83      0.83    333091
weighted avg       0.83      0.83      0.83    333091
```




```
testing accuracy =  78.0
              precision    recall  f1-score   support

           0       0.79      0.75      0.77     82031
           1       0.77      0.80      0.78     82030

    accuracy                           0.78    164061
   macro avg       0.78      0.78      0.78    164061
weighted avg       0.78      0.78      0.78    164061
```



Conclusion

Model	Training Accuracy	Testing Accuracy
Naive Bayes	86%	76%
Multinomial NB	85%	76%
linear SVC	90%	77%
Logistic	83%	78%

We see that Logistic regression model performs best with least overfitting as compared to other models and has better performance in testing dataset as well

REAL TIME ANALYSIS OF TEXTS AS INPUTS

In [42]:

```
def preprocessing(text):
    text = re.sub('[^a-zA-Z]', ' ', text)
    text = text.lower()
    text = re.sub("&lt;/?.*?&gt;", " &lt;&gt; ", text)
    text = re.sub("(\\d|\\W)+", " ", text)
    text = text.split()
    lem = WordNetLemmatizer()
    # Adjust stop word removal to include more words
    text = [lem.lemmatize(word) for word in text if word.lower() not ir
    return " ".join(text)
```

In [43]:
```
stuff_to_be_removed = list(stopwords.words('english')) + list(punctuati
corpus = df['text'].tolist()
final_corpus = []
final_corpus_joined = []
```

In [44]:
```
for i in df.index:
    text = preprocessing(df['text'][i])
    final_corpus.append(text)
    final_corpus_joined.append(" ".join(text))
```

In [78]:
```
def prediction(comment):
    preprocessed_comment = preprocessing(comment)
    comment_list = [preprocessed_comment]
    comment_vector = tfidf.transform(comment_list)
    prediction = lr.predict(comment_vector)[0]
    return prediction
# Enter your Statment for Analysis
statement=input("Enter a statement")
prediction = prediction(statement)
print(statement)
if prediction == 1:
    print("positive comment")
else:
    print("negative comment")
```

```
i like this subject , it is good
positive comment
```

In [ ]: