```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import spacy
import re,string,unicodedata
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import LancasterStemmer,WordNetLemmatizer
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from textblob import TextBlob
from textblob import Word
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
```

```
#importing the training data
imdb_data=pd.read_csv('IMDB Dataset.csv')
print(imdb_data.shape)
imdb_data.head(10)
```

```
    (50000, 2)
```

```
#Summary of the dataset
imdb_data.describe()
```

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	positive
freq	5	25000

6 I sure would like to see a resurrection of a u... positive

```
#sentiment count
imdb_data['sentiment'].value_counts()
```

```
     positive    25000
     negative    25000
     Name: sentiment, dtype: int64
```

```
#split the dataset
#train dataset
train_reviews=imdb_data.review[:40000]
train_sentiments=imdb_data.sentiment[:40000]
#test dataset
test_reviews=imdb_data.review[40000:]
test_sentiments=imdb_data.sentiment[40000:]
print(train_reviews.shape,train_sentiments.shape)
print(test_reviews.shape,test_sentiments.shape)
```

```
     (40000,) (40000,)
     (10000,) (10000,)
```

```
#Tokenization of text
tokenizer=ToktokTokenizer()
#Setting English stopwords
import nltk
nltk.download('stopwords')
stopword_list=nltk.corpus.stopwords.words('english')
```

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

```
#Define function for removing special characters
def remove_special_characters(text, remove_digits=True):
    pattern=r'[^a-zA-z0-9\s]'
    text=re.sub(pattern,'',text)
    return text
```

```
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_special_characters)

#Stemming the text
def simple_stemmer(text):
    ps=nltk.porter.PorterStemmer()
    text= ' '.join([ps.stem(word) for word in text.split()])
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(simple_stemmer)


#set stopwords to english
stop=set(stopwords.words('english'))
print(stop)

#removing the stopwords
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_stopwords)
```

```
{'and', "wouldn't", 'are', 'do', 'at', 'most', 'too', 'have', 'all', 'her', 'no', 'very
```

```
#normalized train reviews
norm_train_reviews=imdb_data.review[:40000]
norm_train_reviews[0]
#convert dataframe to string
#norm_train_string=norm_train_reviews.to_string()
#Spelling correction using Textblob
#norm_train_spelling=TextBlob(norm_train_string)
#norm_train_spelling.correct()
#Tokenization using Textblob
#norm_train_words=norm_train_spelling.words
#norm_train_words
```

```
'one review ha mention watch 1 oz episod youll hook right thi exactli happen mebr
br first thing struck oz wa brutal unflinch scene violenc set right word go trust
thi show faint heart timid thi show pull punch regard drug sex violenc hardcor cl
assic use wordbr br call oz nicknam given oswald maximum secur state penitentari
focus mainli emerald citi experiment section prison cell glass front face inward
privaci high agenda em citi home manyaryan muslim gangsta latino christian italia
n irish moreso scuffl death stare dodgi deal shadi agreement never far awaybr br
would say main appeal show due fact goe show wouldnt dare forget pretti pictur pa
```

```
#Normalized test reviews
norm_test_reviews=imdb_data.review[40000:]
norm_test_reviews[45005]
##convert dataframe to string
#norm_test_string=norm_test_reviews.to_string()
#spelling correction using Textblob
#norm_test_spelling=TextBlob(norm_test_string)
#print(norm_test_spelling.correct())
#Tokenization using Textblob
#norm_test_words=norm_test_spelling.words
#norm_test_words
```

```
'read review watch thi piec cinemat garbag took least 2 page find somebodi els di
dnt think thi appallingli unfunni montag wasnt acm humour 70 inde ani era thi isn
t least funni set sketch comedi ive ever seen itll till come along half skit alre
adi done infinit better act monti python woodi allen wa say nice piec anim last 9
0 second highlight thi film would still get close sum mindless drivelridden thi w
ast 75 minut semin comedi onli world semin realli doe mean semen scatolog humour
onli world scat actual fece precursor joke onli mean thi handbook comedi tit bum
odd beaver niceif pubesc boy least one hand free havent found playboy exist give
```

```
#Count vectorizer for bag of words
cv=CountVectorizer(min_df=0,max_df=1,binary=False,ngram_range=(1,3))
#transformed train reviews
cv_train_reviews=cv.fit_transform(norm_train_reviews)
#transformed test reviews
cv_test_reviews=cv.transform(norm_test_reviews)

print('BOW_cv_train:',cv_train_reviews.shape)
print('BOW_cv_test:',cv_test_reviews.shape)
#vocab=cv.get_feature_names()-toget feature names
```

```
BOW_cv_train: (40000, 6147537)
BOW_cv_test: (10000, 6147537)
```

```
#Tfidf vectorizer
tv=TfidfVectorizer(min_df=0,max_df=1,use_idf=True,ngram_range=(1,3))
#transformed train reviews
tv_train_reviews=tv.fit_transform(norm_train_reviews)
#transformed test reviews
tv_test_reviews=tv.transform(norm_test_reviews)
print('Tfidf_train:',tv_train_reviews.shape)
print('Tfidf_test:',tv_test_reviews.shape)
```

```
Tfidf_train: (40000, 6147537)
Tfidf_test: (10000, 6147537)
```

```
#labeling the sentient data
lb=LabelBinarizer()
#transformed sentiment data
```

```
sentiment_data=lb.fit_transform(imdb_data['sentiment'])
print(sentiment_data.shape)
```

```
(50000, 1)
```

```
#Spliting the sentiment data
train_sentiments=sentiment_data[:40000]
test_sentiments=sentiment_data[40000:]
print(train_sentiments)
print(test_sentiments)
```

```
[[1]
 [1]
 [1]
 ...
 [1]
 [0]
 [0]]
[[0]
 [0]
 [0]
 ...
 [0]
 [0]
 [0]]
```

```
#training the model
lr=LogisticRegression(penalty='l2',max_iter=500,C=1,random_state=42)
#Fitting the model for Bag of words
lr_bow=lr.fit(cv_train_reviews,train_sentiments)
print(lr_bow)
#Fitting the model for tfidf features
lr_tfidf=lr.fit(tv_train_reviews,train_sentiments)
print(lr_tfidf)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWa
  y = column_or_1d(y, warn=True)
LogisticRegression(C=1, max_iter=500, random_state=42)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWa
  y = column_or_1d(y, warn=True)
LogisticRegression(C=1, max_iter=500, random_state=42)
```

```
#Predicting the model for bag of words
lr_bow_predict=lr.predict(cv_test_reviews)
print(lr_bow_predict)
##Predicting the model for tfidf features
lr_tfidf_predict=lr.predict(tv_test_reviews)
print(lr_tfidf_predict)
```

```
[0 0 0 ... 0 1 1]
[0 0 0 ... 0 1 1]
```

```
#Accuracy score for bag of words
lr_bow_score=accuracy_score(test_sentiments,lr_bow_predict)
print("lr_bow_score :",lr_bow_score)
#Accuracy score for tfidf features
lr_tfidf_score=accuracy_score(test_sentiments,lr_tfidf_predict)
print("lr_tfidf_score :",lr_tfidf_score)
```

```
lr_bow_score : 0.7503
lr_tfidf_score : 0.7513
```

```
#Classification report for bag of words
lr_bow_report=classification_report(test_sentiments,lr_bow_predict,target_names=['Positive','
print(lr_bow_report)

#Classification report for tfidf features
lr_tfidf_report=classification_report(test_sentiments,lr_tfidf_predict,target_names=['Positiv
print(lr_tfidf_report)
```

```
              precision    recall  f1-score   support

    Positive       0.75      0.75      0.75      4993
    Negative       0.75      0.75      0.75      5007

    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000

              precision    recall  f1-score   support

    Positive       0.74      0.77      0.75      4993
    Negative       0.76      0.74      0.75      5007

    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000
```

```
#confusion matrix for bag of words
cm_bow=confusion_matrix(test_sentiments,lr_bow_predict,labels=[1,0])
print(cm_bow)
#confusion matrix for tfidf features
cm_tfidf=confusion_matrix(test_sentiments,lr_tfidf_predict,labels=[1,0])
print(cm_tfidf)
```

```
[[3760 1247]
 [1250 3743]]
[[3685 1322]
 [1165 3828]]
```

```
#training the linear svm
```

```
svm=SGDClassifier(loss='hinge',max_iter=500,random_state=42)
#fitting the svm for bag of words
svm_bow=svm.fit(cv_train_reviews,train_sentiments)
print(svm_bow)
#fitting the svm for tfidf features
svm_tfidf=svm.fit(tv_train_reviews,train_sentiments)
print(svm_tfidf)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWa
  y = column_or_1d(y, warn=True)
SGDClassifier(max_iter=500, random_state=42)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWa
  y = column_or_1d(y, warn=True)
SGDClassifier(max_iter=500, random_state=42)
```

```
#Predicting the model for bag of words
svm_bow_predict=svm.predict(cv_test_reviews)
print(svm_bow_predict)
#Predicting the model for tfidf features
svm_tfidf_predict=svm.predict(tv_test_reviews)
print(svm_tfidf_predict)
```

```
[1 1 0 ... 1 1 1]
[1 1 1 ... 1 1 1]
```

```
#Accuracy score for bag of words
svm_bow_score=accuracy_score(test_sentiments,svm_bow_predict)
print("svm_bow_score :",svm_bow_score)
#Accuracy score for tfidf features
svm_tfidf_score=accuracy_score(test_sentiments,svm_tfidf_predict)
print("svm_tfidf_score :",svm_tfidf_score)
```

```
svm_bow_score : 0.5859
svm_tfidf_score : 0.5112
```

```
#Classification report for bag of words
svm_bow_report=classification_report(test_sentiments,svm_bow_predict,target_names=['Positive'
print(svm_bow_report)
#Classification report for tfidf features
svm_tfidf_report=classification_report(test_sentiments,svm_tfidf_predict,target_names=['Posit
print(svm_tfidf_report)
```

```
              precision    recall  f1-score   support

    Positive       0.94      0.18      0.31      4993
    Negative       0.55      0.99      0.70      5007

    accuracy                           0.59     10000
   macro avg       0.74      0.59      0.51     10000
weighted avg       0.74      0.59      0.51     10000
```

```
              precision    recall  f1-score   support

    Positive       1.00      0.02      0.04      4993
    Negative       0.51      1.00      0.67      5007

    accuracy                           0.51     10000
   macro avg       0.75      0.51      0.36     10000
weighted avg       0.75      0.51      0.36     10000
```

```
#confusion matrix for bag of words
cm_bow=confusion_matrix(test_sentiments,svm_bow_predict,labels=[1,0])
print(cm_bow)
#confusion matrix for tfidf features
cm_tfidf=confusion_matrix(test_sentiments,svm_tfidf_predict,labels=[1,0])
print(cm_tfidf)
```

```
[[4944   63]
 [4078  915]]
[[5007    0]
 [4888  105]]
```

```
#training the model
mnb=MultinomialNB()
#fitting the svm for bag of words
mnb_bow=mnb.fit(cv_train_reviews,train_sentiments)
print(mnb_bow)
#fitting the svm for tfidf features
mnb_tfidf=mnb.fit(tv_train_reviews,train_sentiments)
print(mnb_tfidf)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWa
  y = column_or_1d(y, warn=True)
MultinomialNB()
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWa
  y = column_or_1d(y, warn=True)
MultinomialNB()
```

```
#Predicting the model for bag of words
mnb_bow_predict=mnb.predict(cv_test_reviews)
print(mnb_bow_predict)
#Predicting the model for tfidf features
mnb_tfidf_predict=mnb.predict(tv_test_reviews)
print(mnb_tfidf_predict)
```

```
[0 0 0 ... 0 1 1]
[0 0 0 ... 0 1 1]
```

```
#Accuracy score for bag of words
mnb_bow_score=accuracy_score(test_sentiments,mnb_bow_predict)
print("mnb_bow_score :",mnb_bow_score)
#Accuracy score for tfidf features
mnb_tfidf_score=accuracy_score(test_sentiments,mnb_tfidf_predict)
print("mnb_tfidf_score :",mnb_tfidf_score)
```

```
mnb_bow_score : 0.7515
mnb_tfidf_score : 0.7506
```

```
#Classification report for bag of words
mnb_bow_report=classification_report(test_sentiments,mnb_bow_predict,target_names=['Positive'
print(mnb_bow_report)
#Classification report for tfidf features
mnb_tfidf_report=classification_report(test_sentiments,mnb_tfidf_predict,target_names=['Posit
print(mnb_tfidf_report)
```

```
              precision    recall  f1-score   support

    Positive       0.75      0.76      0.75      4993
    Negative       0.75      0.75      0.75      5007

    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000

              precision    recall  f1-score   support

    Positive       0.75      0.76      0.75      4993
    Negative       0.75      0.75      0.75      5007

    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000
```

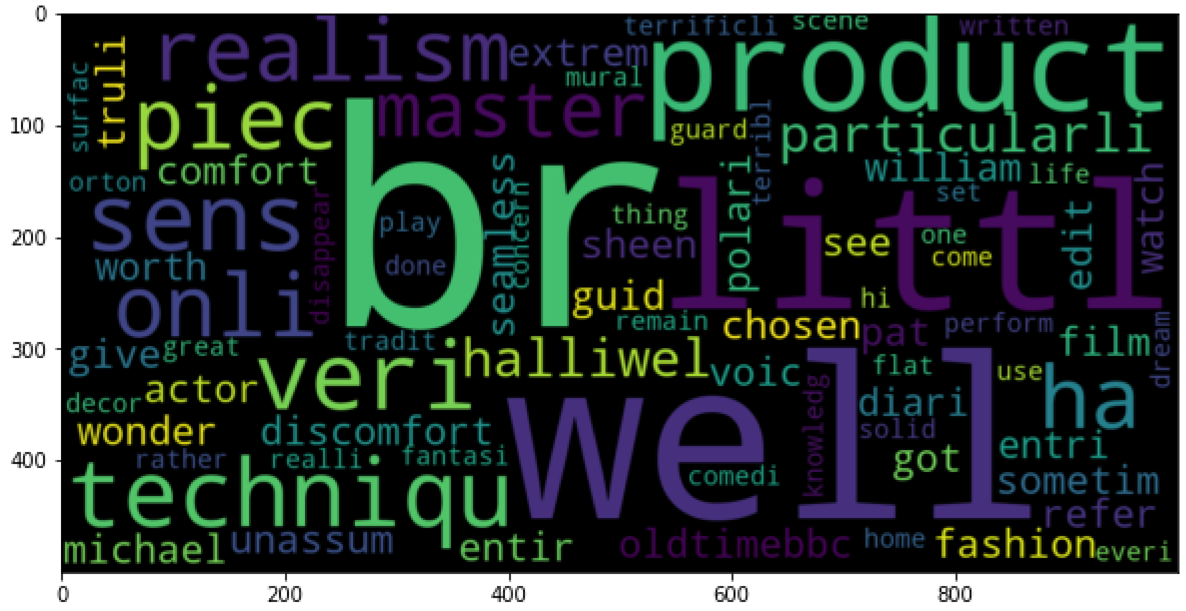
```
#confusion matrix for bag of words
cm_bow=confusion_matrix(test_sentiments,mnb_bow_predict,labels=[1,0])
print(cm_bow)
#confusion matrix for tfidf features
cm_tfidf=confusion_matrix(test_sentiments,mnb_tfidf_predict,labels=[1,0])
print(cm_tfidf)
```

```
[[3744 1263]
 [1222 3771]]
[[3734 1273]
 [1221 3772]]
```

```
#word cloud for positive review words
plt.figure(figsize=(10,10))
positive_text=norm_train_reviews[1]
```

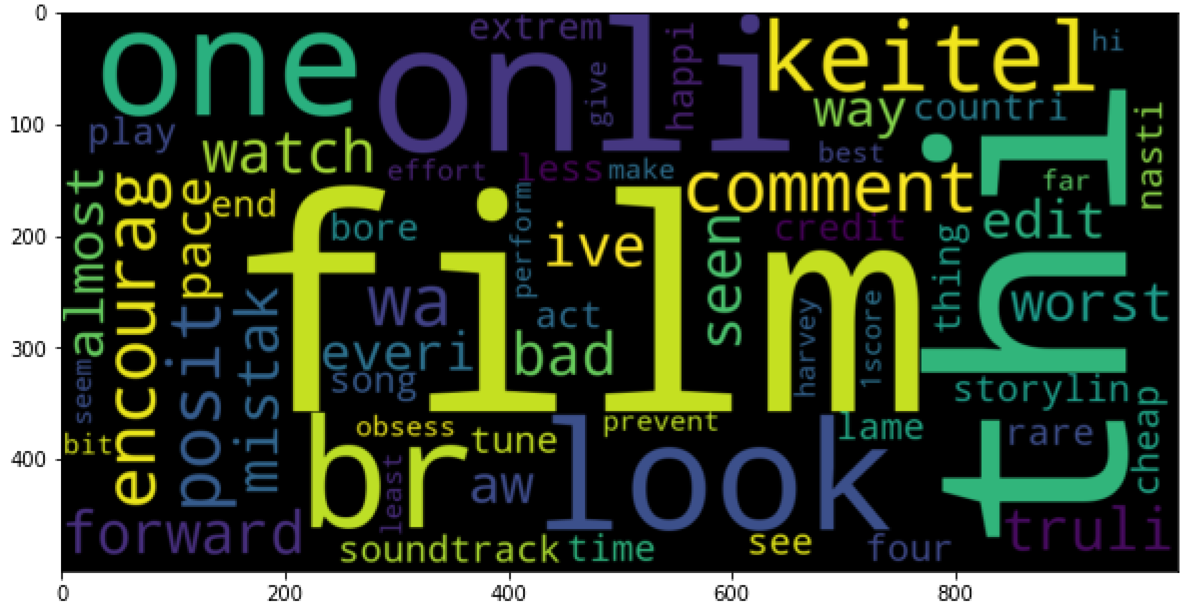
```
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
positive_words=WC.generate(positive_text)
plt.imshow(positive_words,interpolation='bilinear')
plt.show
```

```
<function matplotlib.pyplot.show>
```



```
#Word cloud for negative review words
plt.figure(figsize=(10,10))
negative_text=norm_train_reviews[8]
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
negative_words=WC.generate(negative_text)
plt.imshow(negative_words,interpolation='bilinear')
plt.show
```

```
<function matplotlib.pyplot.show>
```



✓ 2s completed at 18:13 ● ×