```
#Load the libraries

import os
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
from nltk.tokenize import sent_tokenize
from bs4 import BeautifulSoup
import spacy
import re
import string
import 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



import warnings
warnings.filterwarnings('ignore')

# import the training dataset
imdb_data = pd.read_csv('/content/IMDB Dataset.csv.zip')
print(imdb_data.shape)
imdb_data.head(10)
```

```
(50000, 2)
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
5  Probably my all-time favorite movie, a story o...  positive
6  I sure would like to see a resurrection of a u...  positive
7  This show was an amazing, fresh & innovative i...  negative
8  Encouraged by the positive comments about this...  negative
9  If you like original gut wrenching laughter yo...  positive
```

Exploratory data analysis

```
#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
```

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

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

We can see that the data is balanced

Splitting the training dataset

```
#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,)
```

Text normalization: This is the process of transforming a text into a single canonical form.

```
import nltk
nltk.download('stopwords')

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

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Removing html strips and noise text from the dataset

```
#Removing the html strips
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Removing the square brackets
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)

#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(denoise_text)
```

Removing special characters from the dataset

```
# Defining the function for removing the 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_character
s)
```

Stemming:Stemming is important in natural language understanding (NLU) and natural language processing (NLP). Stemming is the process of reducing a word to its stem that affixes to suffixes and prefixes or to the roots of words known as "lemmas".

```
#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)
```

Removing the stopwords

```
#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)
```

```
{'hers', "she's", 'has', 'to', 'other', 'above', 'at', "wouldn't",
'having', 'in', 'here', 'an', 'because', 'too', 'hasn', "you're",
'shan', 'again', "you'd", 'll', 'own', 'but', 'i', 'whom', 'now',
'no', 'be', "mightn't", 'her', 'while', 'their', 'doing', 'my', 'it',
'ourselves', 'his', 'when', 't', 'isn', 'those', "wasn't", 'further',
'should', 'its', 'been', 'being', "don't", 'haven', 'mightn',
'during', 'all', "doesn't", 'as', "haven't", 'few', 'before', 'nor',
'weren', 'or', 'mustn', 'me', 'had', 're', 'with', 'once', 'both',
's', 'there', 'y', 'wouldn', 'couldn', 'are', 'over', 'why', 'just',
'from', 'between', 'didn', "didn't", 'itself', 'that', 'am',
'themselves', 'theirs', 'and', 'than', 'about', 'will', 've',
"mustn't", 'o', 'they', 'don', 'each', 'ma', 'yourself', 'ours', 'm',
"shan't", 'off', 'out', 'up', 'ain', 'until', 'for', 'we', 'who',
'very', 'wasn', 'them', "you'll", "shouldn't", 'through', 'doesn',
'under', "hasn't", 'himself', 'was', 'can', 'our', "aren't", 'if',
'any', 'aren', 'do', 'most', 'have', 'after', 'did', 'down', "won't",
'she', 'such', 'not', 'him', "needn't", 'yours', 'on', 'how', 'which',
'does', "that'll", "it's", 'of', 'same', 'he', 'you', 'won', 'only',
'your', 'is', 'by', 'against', 'more', 'shouldn', 'so', "couldn't",
"hadn't", 'd', 'this', 'what', 'myself', 'were', 'a', 'yourselves',
'where', 'needn', 'some', 'these', "isn't", "you've", 'into', 'then',
"weren't", 'the', 'below', 'hadn', 'herself', "should've"}
```

Normalized train reviews In general Normalization is a pre processing stage of any type of problem statementit is used in scaling the data to be analyzed to a specific range such as [0.0, 1.0] to provide better results.

```
#normalized train reviews
norm_train_reviews=imdb_data.review[:40000]
norm_train_reviews[0]
```

```
#converting the dataframe to string
#norm_train_string=norm_train_reviews.to_string()
#Spelling the correction using Textblob
#norm_train_spelling=TextBlob(norm_train_string)
#norm_train_spelling.correct()
#Tokenization using the Textblob
#norm_train_words=norm_train_spelling.words
#norm_train_words
```

```
{"type":"string"}
```

Normalized test reviews

```
#Normalized test reviews
norm_test_reviews=imdb_data.review[40000:]
norm_test_reviews[45005]
#converting the dataframe to string
#norm_test_string=norm_test_reviews.to_string()
#spelling the correction using the 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
```

```
{"type":"string"}
```

Bags of words model

```
#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, 6209089)
BOW_cv_test: (10000, 6209089)
```

Term Frequency-Inverse Document Frequency model (TFIDF) :It is used to convert text documents to matrix of tfidf features.

```
#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, 6209089)
Tfidf_test: (10000, 6209089)
```

Labeling the sentiment text

```
#labeling the sentiment data
lb=LabelBinarizer()
#transformed sentiment data
sentiment_data=lb.fit_transform(imdb_data['sentiment'])
print(sentiment_data.shape)
```

```
(50000, 1)
```

Split the sentiment data

```
#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]]
```

Modelling of the dataset

```
# Build the logistic regression model for both bag of words & tfidf features
#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)
```

```
LogisticRegression(C=1, max_iter=500, random_state=42)
LogisticRegression(C=1, max_iter=500, random_state=42)
```

Performing the Logistic regression model on test dataset

```
#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 for the lr model

```
#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 : 0.7512
lr_tfidf_score : 0.75
```

Print the Classification report

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

#Classification report for tfidf features
lr_tfidf_report=classification_report(test_sentiments,lr_tfidf_predict
,target_names=['Positive','Negative'])
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
```

```
              precision    recall  f1-score   support

    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      0.73      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
```

Built the Confusion matrix

```
#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)
```

```
[[3768 1239]
 [1249 3744]]
[[3663 1344]
 [1156 3837]]
```

Stochastic gradient descent or Linear support vector machines for bag of words and tfidf features

Performance of model on the test data

```
#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)
```

```
SGDClassifier(max_iter=500, random_state=42)
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 of the svm model

```
#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 : 0.5829
svm_tfidf_score : 0.5112
```

Print the Classification report

```
#Classification report for bag of words
svm_bow_report=classification_report(test_sentiments,svm_bow_predict,t
arget_names=['Positive','Negative'])
print(svm_bow_report)
#Classification report for tfidf features
svm_tfidf_report=classification_report(test_sentiments,svm_tfidf_predi
ct,target_names=['Positive','Negative'])
print(svm_tfidf_report)
```

```
              precision    recall  f1-score   support

    Positive       0.94      0.18      0.30      4993
    Negative       0.55      0.99      0.70      5007

    accuracy                           0.58     10000
   macro avg       0.74      0.58      0.50     10000
weighted avg       0.74      0.58      0.50     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
```

Plot the confusion matrix

```
#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)
```

```
[[4948   59]
 [4112  881]]
[[5007    0]
 [4888  105]]
```

Multinomial Naviebayes for bag of words and tfidf features

```
#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)
```

```
MultinomialNB()
MultinomialNB()
```

MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

Model performance on test data

```
#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 of the model

```
#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 : 0.751
mnb_tfidf_score : 0.7509
```

Print the Classification report

```
#Classification report for bag of words
mnb_bow_report=classification_report(test_sentiments,mnb_bow_predict,t
arget_names=['Positive','Negative'])
print(mnb_bow_report)
#Classification report for tfidf features
mnb_tfidf_report=classification_report(test_sentiments,mnb_tfidf_predi
ct,target_names=['Positive','Negative'])
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      0.74      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
```

Plot the confusion matrix

```
#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)
```

```
[[3736 1271]
 [1219 3774]]
```

```
[[3729 1278]
 [1213 3780]]
```

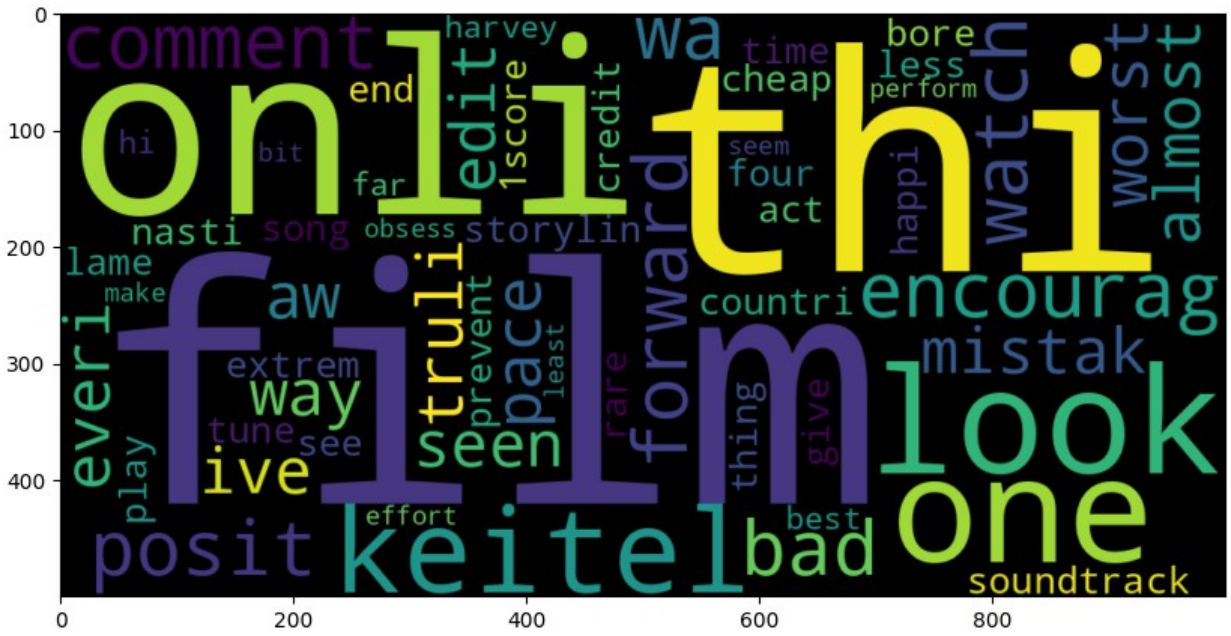
****Print the positive and negative review words by using the WordCloud**

```
#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(close=None, block=None)>
```


```
#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(close=None, block=None)>
```



Conclusion: 1.It can be seen that the both logistic regression model (with 75%) and multinomial naive bayes model(with 75%) are performing well compared to the linear(or)support vector machines(svm). 2.We can still improve the accuracy of the models by preprocessing the data and by using the lexicon models like Textblob.