In [4]:
```
#Load the libraries
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,accura

import os
print(os.listdir("C:/Users/admin/Desktop/Untitled Folder"))
import warnings
warnings.filterwarnings('ignore')

```

```
['.ipynb_checkpoints', 'Assignment 2 DL.ipynb', 'desktop.ini', 'DL Binary Cla
ssification.ipynb', 'DL Binary.ipynb', 'DL Linear Regression.ipynb', 'IMDB Da
taset.csv']
```

**Sentiment Analysis of IMDB Movie Reviews**

**Problem Statement:**

In this, we have to predict the number of positive and negative reviews based on sentiments by using different classification models.

**Import necessary libraries**

**Import the training dataset**

In [5]:
```
#importing the training data
imdb_data=pd.read_csv('IMDB Dataset.csv')
print(imdb_data.shape)
imdb_data.head(10)
```

```
(50000, 2)
```

Out[5]:

| | review | sentiment |
|---|---|---|
| **0** | One of the other reviewers has mentioned that ... | positive |
| **1** | A wonderful little production. <br /><br />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 |

### Exploratery data analysis

In [6]:
```
#Summary of the dataset
imdb_data.describe()
```

Out[6]:

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

### Sentiment count

In [7]:
```
#sentiment count
imdb_data['sentiment'].value_counts()
```

Out[7]:
```
negative    25000
positive    25000
Name: sentiment, dtype: int64
```

We can see that the dataset is balanced.

### Spliting the training dataset

In [8]:
```
#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**

In [9]:
```
#Tokenization of text


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

**Removing html strips and noise text**

In [10]:
```
#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**

In [11]:
```
#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)
```

*Text stemming *

In [14]:
```
#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 stopwords**

In [15]:
```
#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 stopw
    else:
        filtered_tokens = [token for token in tokens if token.lower() not
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_stopwords)
```

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

**Normalized train reviews**

In [16]:
```
#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
```

Out[16]:
```
'one review ha mention watch 1 oz episod youll hook right thi exactli happen
meth 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 classic use wordit call oz nicknam given oswald maximum secur state p
enitentari focus mainli emerald citi experiment section prison cell glass fro
nt face inward privaci high agenda em citi home manyaryan muslim gangsta lati
no christian italian irish moreso scuffl death stare dodgi deal shadi agreeme
nt never far awayi would say main appeal show due fact goe show wouldnt dare
forget pretti pictur paint mainstream audienc forget charm forget romanceoz d
oesnt mess around first episod ever saw struck nasti wa surreal couldnt say w
a readi watch develop tast oz got accustom high level graphic violenc violenc
injustic crook guard wholl sold nickel inmat wholl kill order get away well m
anner middl class inmat turn prison bitch due lack street skill prison experi
watch oz may becom comfort uncomfort viewingthat get touch darker side'
```

**Normalized test reviews**

In [17]:
```
#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
```

Out[17]:
```
'read review watch thi piec cinemat garbag took least 2 page find somebodi el
s didnt think thi appallingli unfunni montag wasnt acm humour 70 inde ani era
thi isnt least funni set sketch comedi ive ever seen itll till come along hal
f skit alreadi done infinit better act monti python woodi allen wa say nice p
iec anim last 90 second highlight thi film would still get close sum mindless
drivelridden thi wast 75 minut semin comedi onli world semin realli doe mean
semen scatolog humour onli world scat actual fece precursor joke onli mean th
i handbook comedi tit bum odd beaver niceif pubesc boy least one hand free ha
vent found playboy exist give break becaus wa earli 70 way sketch comedi go b
ack least ten year prior onli way could even forgiv thi film even made wa gun
point retro hardli sketch clown subtli pervert children may cut edg circl cou
ld actual funni come realli quit sad kept go throughout entir 75 minut sheer
belief may save genuin funni skit end gave film 1 becaus wa lower scoreand on
li recommend insomniac coma patientsor perhap peopl suffer lockjawtheir jaw w
ould final drop open disbelief'
```

**Bags of words model **

It is used to convert text documents to numerical vectors or bag of words.

In [18]:
```
#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.

In [19]:
```
#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**

In [20]:
```
#labeling the sentient data
lb=LabelBinarizer()
#transformed sentiment data
sentiment_data=lb.fit_transform(imdb_data['sentiment'])
print(sentiment_data.shape)
```

```
(50000, 1)
```

**Split the sentiment tdata**

In [21]:
```
#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 the dataset**

Let us build logistic regression model for both bag of words and tfidf features

In [39]:
```
#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)
```

**Logistic regression model performane on test dataset**

In [23]:
```
#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 of the model**

In [24]:
```
#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**

In [25]:
```
#Classification report for bag of words
lr_bow_report=classification_report(test_sentiments,lr_bow_predict,target_
print(lr_bow_report)

#Classification report for tfidf features
lr_tfidf_report=classification_report(test_sentiments,lr_tfidf_predict,tar
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      0.73      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
```

**Confusion matrix**

In [26]:
```
#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**

In [27]:
```
#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)
```

**Model performance on test data**

In [28]:
```
#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 model**

In [29]:
```
#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**

In [30]:
```
#Classification report for bag of words
svm_bow_report=classification_report(test_sentiments,svm_bow_predict,targ
print(svm_bow_report)
#Classification report for tfidf features
svm_tfidf_report=classification_report(test_sentiments,svm_tfidf_predict,
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**

In [31]:
```
#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 Naive Bayes for bag of words and tfidf features**

In [32]:
```
#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()
```

### Model performance on test data

In [33]:
```
#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

In [34]:
```
#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

In [35]:
```
#Classification report for bag of words
mnb_bow_report=classification_report(test_sentiments,mnb_bow_predict,targ
print(mnb_bow_report)
#Classification report for tfidf features
mnb_tfidf_report=classification_report(test_sentiments,mnb_tfidf_predict,
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**

In [36]:
```
#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)
```

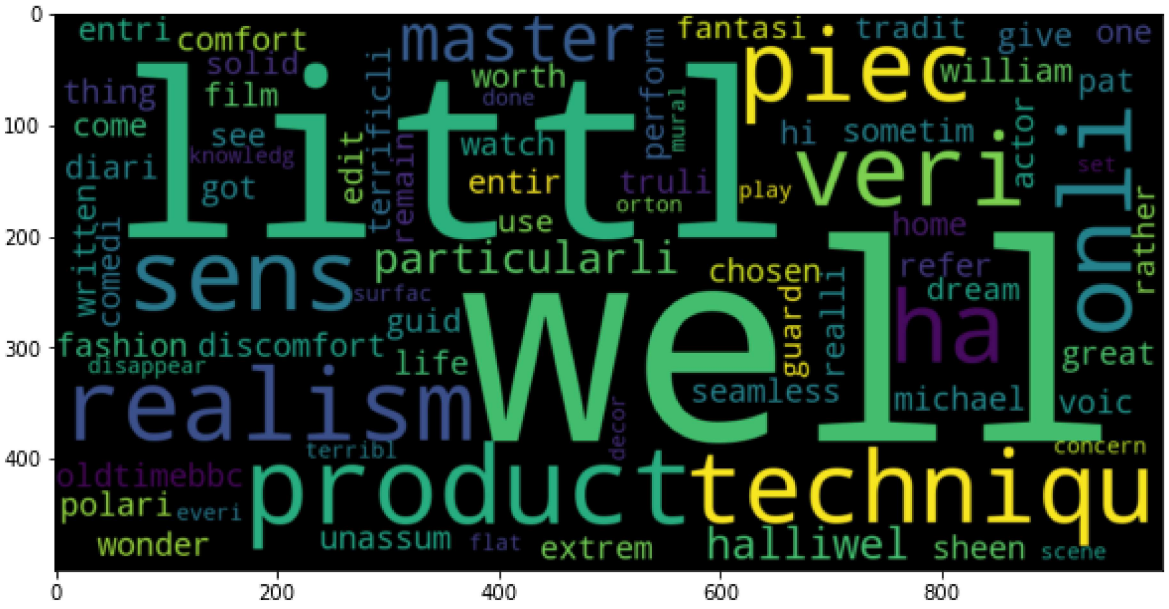
```
[[3736 1271]
 [1219 3774]]
[[3729 1278]
 [1213 3780]]
```

**Let us see positive and negative words by using WordCloud.**

**Word cloud for positive review words**

In [37]:
```
#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
```

Out[37]: <function matplotlib.pyplot.show(close=None, block=None)>




**Word cloud for negative review words**

In [38]:
```
#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
```

Out[38]: <function matplotlib.pyplot.show(close=None, block=None)>