In [7]:
```python
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 sklearn.metrics import classification_report,confusion_matrix,accu
racy_score
```

In [8]:
```python
imdb_data=pd.read_csv('IMDB Dataset.csv')
print(imdb_data.shape)
imdb_data.head(10)
```

```
(50000, 2)
```

Out[8]:

	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

In [9]:
```python
imdb_data.describe()
```

Out[9]:

	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 [10]:
```python
imdb_data['sentiment'].value_counts()
```

Out[10]:
```
negative    25000
positive    25000
Name: sentiment, dtype: int64
```

In [11]:
```python
#split the dataset
#train dataset
train_reviews=imdb_data.review[:30000]
train_sentiments=imdb_data.sentiment[:30000]
#test dataset
test_reviews=imdb_data.review[30000:]
test_sentiments=imdb_data.sentiment[30000:]
print(train_reviews.shape,train_sentiments.shape)
print(test_reviews.shape,test_sentiments.shape)
```

```
(30000,) (30000,)
(20000,) (20000,)
```

Text normalization

In [13]:
```python
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
[nltk_data]     C:\Users\Prachi\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
```

Removing html strips and noise text

In [14]:
```python
#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 [15]:
```python
#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
)
```

Removing stopwords

In [16]:
```python
#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 st
opword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() n
ot 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)
```

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

Text stemming

In [17]:
```python
#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)
```

In [18]:
```python
imdb_data.head(10)
```

Out[18]:

	review	sentiment
0	one review mention watch 1 Oz episod youll hoo...	positive
1	wonder littl product film techniqu unassum old...	positive
2	thought wonder way spend time hot summer weeke...	positive
3	basic there famili littl boy jake think there ...	negative
4	petter mattei love time money visual stun film...	positive
5	probabl alltim favorit movi stori selfless sac...	positive
6	sure would like see resurrect date seahunt ser...	positive
7	show amaz fresh innov idea 70 first air first ...	negative
8	encourag posit comment film look forward watch...	negative
9	like origin gut wrench laughter like movi youn...	positive

In [19]:
```python
norm_train_reviews=imdb_data.review[:30000]
norm_test_reviews=imdb_data.review[30000:]
```

Term Frequency-Inverse Document Frequency model (TFIDF)

It is used to convert text documents to matrix of tfidf features.

In [20]:
```python
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: (30000, 4768828)
Tfidf_test: (20000, 4768828)
```

Labeling the sentiment text

In [21]:
```python
#labeling the sentient data
lb=LabelBinarizer()
#transformed sentiment data
sentiment_data=lb.fit_transform(imdb_data['sentiment'])
print(sentiment_data)
print(sentiment_data[:10])
```

```
(50000, 1)
[[1]
 [1]
 [1]
 [0]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]]
```

Split the sentiment data

In [26]:
```python
#Spliting the sentiment data
train_sentiments=sentiment_data[:30000]
test_sentiments=sentiment_data[30000:]
print(train_sentiments.shape)
print(test_sentiments)
```

```
(30000, 1)
[[1]
 [0]
 [0]
 ...
 [0]
 [0]
 [0]]
```

Stochastic gradient descent or Linear support vector machines for tfidf features

In [31]:
```python
import warnings
warnings.filterwarnings('ignore')
#training the linear svm
svm=SGDClassifier(loss='hinge',max_iter=500,random_state=42)
#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)
```

Model performance on test data

In [32]:
```python
#Predicting the model for tfidf features
svm_tfidf_predict=svm.predict(tv_test_reviews)
print(svm_tfidf_predict)
```

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

Accuracy of the model

In [33]:
```python
#Accuracy score for tfidf features
svm_tfidf_score=accuracy_score(test_sentiments,svm_tfidf_predict)
print("svm_tfidf_score :",svm_tfidf_score)
```

```
svm_tfidf_score : 0.5074
```

Print the classification report

In [34]:
```python
#Classification report for tfidf features
svm_tfidf_report=classification_report(test_sentiments,svm_tfidf_predic
t,target_names=['Positive','Negative'])
print(svm_tfidf_report)
```

```
              precision    recall  f1-score   support

    Positive       0.99      0.02      0.03     10015
    Negative       0.50      1.00      0.67      9985

    accuracy                           0.51     20000
   macro avg       0.75      0.51      0.35     20000
weighted avg       0.75      0.51      0.35     20000
```