Bag of Words Model with Naive Bayes:

In [1]:
```python
import pandas as pd
from bs4 import BeautifulSoup
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

In [2]:
```python
data = pd.read_csv('IMDB Dataset.csv')
data.head()
```

Out[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

Basic Statistics

In [3]:
```python
print("Number of rows: ", data.shape[0])
print("Number of columns: ", data.shape[1])
```

```
Number of rows:  50000
Number of columns:  2
```

In [4]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

In [5]:
```python
data.sentiment.value_counts()
```

Out[5]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

from the above, we can confirm that the data is equally partioned.

Data Cleaning and preprocessing

In [6]:
```python
data['review'][1]
```

Out[6]:
```
'A wonderful little production. <br /><br />The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece. <br /><br />The actors are extremely well chosen- Michael Sheen not only "has got all the polari" but he has all the voices down pat too! You can truly see the seamless editing guided by the references to Williams\' diary entries, not only is it well worth the watching but it is a terrificly written and performed piece. A masterful production about one of the great master\'s of comedy and his life. <br /><br />The realism really comes home with the little things: the fantasy of the guard which, rather than use the traditional \'dream\' techniques remains solid then disappears. It plays on our knowledge and our senses, particularly with the scenes concerning Orton and Halliwell and the sets (particularly of their flat with Halliwell\'s murals decorating every surface) are terribly well done.'
```

In the above data we can see \

\ break tags. We need to remove them before using this data.

In [7]:
```python
cleantext = BeautifulSoup(data["review"][1], 'lxml').text
```

We need to remove the slash

In [8]:
```python
import re
cleantext = re.sub(r'[^\w\s]', '', cleantext)
cleantext
```

Out[8]:
```
'A wonderful little production The filming technique is very unassuming very oldtimeBBC fashion and gives a comforting and sometimes discomforting sense of realism to the entire piece The actors are extremely well chosen Michael Sheen not only has got all the polari but he has all the voices down pat too You can truly see the seamless editing guided by the references to Williams diary entries not only is it well worth the watching but it is a terrificly written and performed piece A masterful production about one of the great masters of comedy and his life The realism really comes home with the little things the fantasy of the guard which rather than use the traditional dream techniques remains solid then disappears It plays on our knowledge and our senses particularly with the scenes concerning Orton and Halliwell and the sets particularly of their flat with Halliwells murals decorating every surface are terribly well done'
```

In [9]:
```python
import nltk
from nltk.corpus import stopwords
```

In [10]:
```python
nltk.download('stopwords')
stopwords.words('english')
```

```
[nltk_data] Error loading stopwords: <urlopen error [WinError 10060] A
[nltk_data]     connection attempt failed because the connected party
[nltk_data]     did not properly respond after a period of time, or
[nltk_data]     established connection failed because connected host
[nltk_data]     has failed to respond>
```

Out[10]:
```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
 'himself',
 'she',
 "she's",
 'her',
 'hers',
 'herself',
 'it',
 "it's",
 'its',
 'itself',
 'they',
 'them',
 'their',
 'theirs',
 'themselves',
 'what',
 'which',
 'who',
 'whom',
 'this',
 'that',
 "that'll",
 'these',
 'those',
 'am',
 'is',
 'are',
 'was',
 'were',
 'be',
 'been',
 'being',
 'have',
 'has',
 'had',
 'having',
 'do',
 'does',
 'did',
 'doing',
 'a',
 'an',
 'the',
 'and',
 'but',
 'if',
 'or',
 'because',
 'as',
 'until',
 'while',
 'of',
 'at',
 'by',
 'for',
 'with',
 'about',
 'against',
 'between',
 'into',
 'through',
 'during',
 'before',
 'after',
 'above',
 'below',
 'to',
 'from',
 'up',
 'down',
 'in',
 'out',
 'on',
 'off',
 'over',
 'under',
 'again',
 'further',
 'then',
 'once',
 'here',
 'there',
 'when',
 'where',
 'why',
 'how',
 'all',
 'any',
 'both',
 'each',
 'few',
 'more',
 'most',
 'other',
 'some',
 'such',
 'no',
 'nor',
 'not',
 'only',
 'own',
 'same',
 'so',
 'than',
 'too',
 'very',
 's',
 't',
 'can',
 'will',
 'just',
 'don',
 "don't",
 'should',
 "should've",
 'now',
 'd',
 'll',
 'm',
 'o',
 're',
 've',
 'y',
 'ain',
 'aren',
 "aren't",
 'couldn',
 "couldn't",
 'didn',
 "didn't",
 'doesn',
 "doesn't",
 'hadn',
 "hadn't",
 'hasn',
 "hasn't",
 'haven',
 "haven't",
 'isn',
 "isn't",
 'ma',
 'mightn',
 "mightn't",
 'mustn',
 "mustn't",
 'needn',
 "needn't",
 'shan',
 "shan't",
 'shouldn',
 "shouldn't",
 'wasn',
 "wasn't",
 'weren',
 "weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't"]
```

In [11]:
```python
token = cleantext.lower().split()
stopword = set(stopwords.words('english'))
token_list = [ word for word in token if word.lower() not in stopword ]
```

In [12]:
```python
" ".join(token_list)
```

Out[12]:
```
'wonderful little production filming technique unassuming oldtimebbc fashion gives comforting sometimes discomforting sense realism entire piece actors extremely well chosen michael sheen got polari voices pat truly see seamless editing guided references williams diary entries well worth watching terrificly written performed piece masterful production one great masters comedy life realism really comes home little things fantasy guard rather use traditional dream techniques remains solid disappears plays knowledge senses particularly scenes concerning orton halliwell sets particularly flat halliwells murals decorating every surface terribly well done'
```

In [13]:
```python
lemmatizer = WordNetLemmatizer()
```

In [14]:
```python
lemmatizer.lemmatize(" ".join(token_list))
```

Out[14]:
```
'wonderful little production filming technique unassuming oldtimebbc fashion gives comforting sometimes discomforting sense realism entire piece actors extremely well chosen michael sheen got polari voices pat truly see seamless editing guided references williams diary entries well worth watching terrificly written performed piece masterful production one great masters comedy life realism really comes home little things fantasy guard rather use traditional dream techniques remains solid disappears plays knowledge senses particularly scenes concerning orton halliwell sets particularly flat halliwells murals decorating every surface terribly well done'
```

In [15]:
```python
data.keys()
```

Out[15]:
```
Index(['review', 'sentiment'], dtype='object')
```

In [16]:
```python
from tqdm import tqdm
def data_cleaner(data):
    clean_data = []
    for review in tqdm(data):
        cleantext = BeautifulSoup(review, "lxml").text
        cleantext = re.sub(r'[^\w\s]', '', cleantext)
        cleantext = [ token for token in cleantext.lower().split() if token not in stopword ]
        cleantext = lemmatizer.lemmatize(" ".join(cleantext))
        clean_data.append(cleantext.strip())
    return clean_data
```

In [ ]:
```python
clean_data = data_cleaner(data.review.values)
```

In [18]:
```python
clean_data[0]
```

Out[18]:
```
'one reviewers mentioned watching 1 oz episode youll hooked right exactly happened methe first thing struck oz brutality unflinching scenes violence set right word go trust show faint hearted timid show pulls punches regards drugs sex violence hardcore classic use wordit called oz nickname given oswald maximum security state penitentary focuses mainly emerald city experimental section prison cells glass fronts face inwards privacy high agenda em city home manyaryans muslims gangstas latinos christians italians irish moreso scuffles death stares dodgy dealings shady agreements never far awayi would say main appeal show due fact goes shows wouldnt dare forget pretty pictures painted mainstream audiences forget charm forget romanceoz doesnt mess around first episode ever saw struck nasty surreal couldnt say ready watched developed taste oz got accustomed high levels graphic violence violence injustice crooked guards wholl sold nickel inmates wholl kill order get away well mannered middle class inmates turned prison bitches due lack street skills prison experience watching oz may become comfortable uncomfortable viewingthats get touch darker side'
```

Train test split

In [19]:
```python
X_train, X_test, y_train, y_test = train_test_split(data, data.sentiment, test_size=0.2, random_state=42, stratify=data.sentiment)
```

In [20]:
```python
le = LabelEncoder()
y_train = le.fit_transform(y_train)
le_test = LabelEncoder()
y_test = le_test.fit_transform(y_test)
```

In [21]:
```python
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(40000, 2) (40000,)
(10000, 2) (10000,)
```

In [ ]:
```python
clean_data_train_data = data_cleaner(X_train.review.values)
```

In [23]:
```python
X_train['cleaned_text'] = clean_data_train_data
X_train.head()
```

Out[23]:

	review	sentiment	cleaned_text
47808	I caught this little gem totally by accident b...	positive	caught little gem totally accident back 1980 8...
20154	I can't believe that I let myself into this mo...	negative	cant believe let movie accomplish favor friend...
43069	*spoiler alert!* it just gets to me the nerve ...	negative	spoiler alert gets nerve people remake use ter...
19413	If there's one thing I've learnt from watching...	negative	theres one thing ive learnt watching george ro...
13673	I remember when this was in theaters, reviews ...	negative	remember theaters reviews said horrible well d...

In [ ]:
```python
clean_data_test_data = data_cleaner(X_test.review.values)
X_test['cleaned_text'] = clean_data_test_data
X_test.head()
```

Vectorizer

In [25]:
```python
vec = CountVectorizer()
vec = vec.fit(X_train.cleaned_text)
train_x_bow = vec.transform(X_train.cleaned_text)
test_x_bow = vec.transform(X_test.cleaned_text)
```

In [26]:
```python
print(train_x_bow.shape)
print(test_x_bow.shape)
```

```
(40000, 192139)
(10000, 192139)
```

Naive Bayes with Hyperparameter Tuning

In [27]:
```python
classifier = MultinomialNB()
```

In [28]:
```python
alpha_ranges = {"alpha": [0.001, 0.01, 0.1, 1, 10.0, 100]}
```

In [ ]:
```python
grid_search = GridSearchCV(classifier, param_grid=alpha_ranges, scoring='accuracy', cv=3, return_train_score=True)
grid_search.fit(train_x_bow, y_train)
```

In [30]:
```python
alpha = [0.001, 0.01, 0.1, 1, 10.0, 100]
train_acc = grid_search.cv_results_['mean_train_score']
train_std = grid_search.cv_results_['std_train_score']

test_acc = grid_search.cv_results_['mean_test_score']
test_std = grid_search.cv_results_['std_test_score']
```

In [ ]:
```python
grid_search.best_estimator_
```

In [ ]:
```python
classifier = MultinomialNB(alpha=1)
classifier.fit(train_x_bow, y_train)
```

In [33]:
```python
predict = classifier.predict(test_x_bow)
```

In [34]:
```python
print("Accuracy is ", accuracy_score(y_test, predict))
```

```
Accuracy is  0.8599
```

In [35]:
```python
print("Accuracy is ", classification_report(y_test, predict))
```

```
Accuracy is                precision    recall  f1-score   support

           0       0.85      0.88      0.86      5000
           1       0.87      0.84      0.86      5000

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

TF-IDF Model with Naive Bayes:

In [36]:
```python
# Vectorize the text using TF-IDF model
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train.cleaned_text)
X_test_tfidf = tfidf_vectorizer.transform(X_test.cleaned_text)

# Train a Naive Bayes classifier on the TF-IDF features
nb_classifier_tfidf = MultinomialNB()
nb_classifier_tfidf.fit(X_train_tfidf, y_train)

# Predict and calculate accuracy
predictions_tfidf = nb_classifier_tfidf.predict(X_test_tfidf)
accuracy_tfidf = accuracy_score(y_test, predictions_tfidf)
print("Accuracy using TF-IDF model:", accuracy_tfidf)
```

```
Accuracy using TF-IDF model: 0.867
```