IMDB Dataset Classifier: ML Algorithms

Load Dependencies

In [1]:

```
%%capture
!pip install nltk
```

In [2]:

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

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\chung\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\chung\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[2]:

```
True
```

In [3]:

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import pos_tag, word_tokenize
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

In [4]:

```
import pandas as pd
import numpy as np
```

Data Exploration

In [5]:

```
df = pd.read_csv("IMDB_Dataset.csv")
```

In [6]:

```
df.head()
```

Out[6]:

	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

In [7]:

```
df.groupby('sentiment').count()
```

Out[7]:

	review
sentiment	
negative	25000
positive	25000

The dataset is balanced with 25K each of positive and negative reviews

Data Pre-Processing

In [8]:

```
# Change lower case

def ChangeLower(msg):
    # converting messages to lowercase
    msg = msg.lower()
    return msg

df['review']=df['review'].apply(ChangeLower)
```

In [9]:

```
df['review'].head()
```

Out[9]:

```
0    one of the other reviewers has mentioned that ...
1    a wonderful little production. <br /><br />the...
2    i thought this was a wonderful way to spend ti...
3    basically there's a family where a little boy ...
4    petter mattei's "love in the time of money" is...
Name: review, dtype: object
```

Remove Stop Words

In [10]:

```
stopwords = set(stopwords.words('english'))
```

In [11]:

```
def RemoveStop (msg):
    msg = [word for word in msg.split() if word not in stopwords]
    return msg

df['review']=df['review'].apply(RemoveStop)
```

In [12]:

```
df['review'].head()
```

Out[12]:

```
0    [one, reviewers, mentioned, watching, 1, oz, e...
1    [wonderful, little, production., <br, /><br, /...
2    [thought, wonderful, way, spend, time, hot, su...
3    [basically, there's, family, little, boy, (jak...
4    [petter, mattei's, "love, time, money", visual...
Name: review, dtype: object
```

Check for Most Common Words & Remove if not Meaninful

In [13]:

```
from collections import Counter
df['review']=df['review'].apply(lambda a: ' '.join(a))
Counter(" ".join(df["review"]).split()).most_common(100)
```

Out[13]:

```
[('/><br', 100974),
 ('movie', 61492),
 ('film', 55086),
 ('one', 44983),
 ('like', 37281),
 ('would', 23807),
 ('even', 23681),
 ('good', 23467),
 ('really', 21805),
 ('see', 20901),
 ('-', 18201),
 ('get', 17689),
 ('much', 17278),
 ('story', 16810),
 ('also', 15743),
 ('time', 15657),
 ('great', 15465),
 ('first', 15455),
 ('make', 15028),
 ('people', 15028),
 ('could', 14927),
 ('/>the', 14702),
 ('made', 13562),
 ('bad', 13494),
 ('think', 13304),
 ('many', 12877),
 ('never', 12621),
 ('two', 12189),
 ('<br', 12028),
 ('little', 11827),
 ('well', 11692),
 ('watch', 11461),
 ('way', 11375),
 ('it.', 11169),
 ('know', 10784),
 ('movie.', 10764),
 ('love', 10748),
 ('best', 10743),
 ('seen', 10611),
 ('characters', 10599),
 ('character', 10386),
 ('movies', 10349),
 ('ever', 10218),
 ('still', 9778),
 ('films', 9578),
 ('plot', 9455),
 ('acting', 9378),
 ('show', 9376),
 ('better', 9045),
 ('film.', 8921),
 ('say', 8824),
 ('go', 8798),
 ('something', 8764),
```

```
 ("i'm", 8262),
 ('scene', 8235),
 ('makes', 8222),
 ('watching', 8146),
 ('film,', 8120),
 ('real', 8041),
 ('movie,', 8040),
 ('find', 8002),
 ('back', 7904),
 ('actually', 7798),
 ('scenes', 7797),
 ('every', 7791),
 ('going', 7659),
 ('man', 7659),
 ('life', 7570),
 ('new', 7502),
 ('/>i', 7498),
 ('nothing', 7417),
 ('look', 7409),
 ('another', 7379),
 ('lot', 7356),
 ('quite', 7120),
 ('thing', 7113),
 ('&', 7063),
 ('want', 7048),
 ('end', 6962),
 ('pretty', 6953),
 ('old', 6946),
 ('seems', 6860),
 ("can't", 6847),
 ('got', 6774),
 ('take', 6632),
 ('actors', 6612),
 ('give', 6556),
 ('years', 6517),
 ('part', 6515),
 ('may', 6390),
 ('young', 6340),
 ("that's", 6301),
 ("i've", 6247),
 ('us', 6241),
 ('without', 6202),
 ('big', 6198),
 ('thought', 6184),
 ('things', 6168),
 ('around', 6085),
 ('it,', 6068)]
```

In [14]:

```
morewords ={"i've",'&','/>i',"i'm",'it.','<br','two','/>the','-','one','film','movie','/><b
```

In [15]:

```
stopwords.update(morewords)
```

In [16]:

```
df.head()
```

Out[16]:

	review	sentiment
0	one reviewers mentioned watching 1 oz episode ...	positive
1	wonderful little production. the f...	positive
2	thought wonderful way spend time hot summer we...	positive
3	basically there's family little boy (jake) thi...	negative
4	petter mattei's "love time money" visually stu...	positive

In [17]:

```
def RemoveStop (msg):
    msg = [word for word in msg.split() if word not in stopwords]
    return msg

df['review']=df['review'].apply(RemoveStop)
```

Carry out a second scan of most common words & remove

In [18]:

```
df['review']=df['review'].apply(lambda a: ' '.join(a))
Counter(" ".join(df["review"]).split()).most_common(100)
```

Out[18]:

```
[('like', 37281),
 ('would', 23807),
 ('even', 23681),
 ('good', 23467),
 ('really', 21805),
 ('see', 20901),
 ('get', 17689),
 ('much', 17278),
 ('story', 16810),
 ('also', 15743),
 ('time', 15657),
 ('great', 15465),
 ('first', 15455),
 ('make', 15028),
 ('people', 15028),
 ('could', 14927),
 ('made', 13562),
 ('bad', 13494),
 ('think', 13304),
 ('many', 12877),
 ('never', 12621),
 ('little', 11827),
 ('well', 11692),
 ('watch', 11461),
 ('way', 11375),
 ('know', 10784),
 ('movie.', 10764),
 ('love', 10748),
 ('best', 10743),
 ('seen', 10611),
 ('characters', 10599),
 ('character', 10386),
 ('movies', 10349),
 ('ever', 10218),
 ('still', 9778),
 ('films', 9578),
 ('plot', 9455),
 ('acting', 9378),
 ('show', 9376),
 ('better', 9045),
 ('film.', 8921),
 ('say', 8824),
 ('go', 8798),
 ('something', 8764),
 ('scene', 8235),
 ('makes', 8222),
 ('watching', 8146),
 ('film,', 8120),
 ('real', 8041),
 ('movie,', 8040),
 ('find', 8002),
 ('back', 7904),
 ('actually', 7798),
 ('scenes', 7797),
```

```
 ('every', 7791),
 ('going', 7659),
 ('man', 7659),
 ('life', 7570),
 ('new', 7502),
 ('nothing', 7417),
 ('look', 7409),
 ('another', 7379),
 ('lot', 7356),
 ('quite', 7120),
 ('thing', 7113),
 ('want', 7048),
 ('end', 6962),
 ('pretty', 6953),
 ('old', 6946),
 ('seems', 6860),
 ("can't", 6847),
 ('got', 6774),
 ('take', 6632),
 ('actors', 6612),
 ('give', 6556),
 ('years', 6517),
 ('part', 6515),
 ('may', 6390),
 ('young', 6340),
 ("that's", 6301),
 ('us', 6241),
 ('without', 6202),
 ('big', 6198),
 ('thought', 6184),
 ('things', 6168),
 ('around', 6085),
 ('it,', 6068),
 ('saw', 6051),
 ('gets', 6049),
 ('almost', 6020),
 ('must', 6004),
 ('though', 6000),
 ('director', 5960),
 ('always', 5898),
 ('whole', 5796),
 ('horror', 5766),
 ('come', 5765),
 ('work', 5689),
 ('might', 5653),
 ("there's", 5615)]
```

In [19]:

```
morewords={"there's",'it,''things','may',"can't",'seems','quite','thing','movie,','film,','
```

In [20]:

```
stopwords.update(morewords)
```

In [21]:

```
def RemoveStop (msg):
    msg = [word for word in msg.split() if word not in stopwords]
    return msg

df['review']=df['review'].apply(RemoveStop)
```

Final scan of most common words

In [22]:

```
df['review']=df['review'].apply(lambda a: ' '.join(a))
Counter(" ".join(df["review"]).split()).most_common(100)
```

Out[22]:

```
[('like', 37281),
 ('even', 23681),
 ('good', 23467),
 ('really', 21805),
 ('see', 20901),
 ('get', 17689),
 ('much', 17278),
 ('story', 16810),
 ('also', 15743),
 ('time', 15657),
 ('great', 15465),
 ('first', 15455),
 ('make', 15028),
 ('people', 15028),
 ('could', 14927),
 ('made', 13562),
 ('bad', 13494),
 ('think', 13304),
 ('many', 12877),
 ('never', 12621),
 ('little', 11827),
 ('well', 11692),
 ('watch', 11461),
 ('way', 11375),
 ('know', 10784),
 ('love', 10748),
 ('best', 10743),
 ('seen', 10611),
 ('ever', 10218),
 ('still', 9778),
 ('films', 9578),
 ('plot', 9455),
 ('acting', 9378),
 ('show', 9376),
 ('better', 9045),
 ('say', 8824),
 ('go', 8798),
 ('scene', 8235),
 ('makes', 8222),
 ('watching', 8146),
 ('real', 8041),
 ('find', 8002),
 ('back', 7904),
 ('actually', 7798),
 ('scenes', 7797),
 ('every', 7791),
 ('going', 7659),
 ('man', 7659),
 ('life', 7570),
 ('new', 7502),
 ('nothing', 7417),
 ('look', 7409),
 ('another', 7379),
 ('lot', 7356),
```

```
 ('want', 7048),
 ('end', 6962),
 ('pretty', 6953),
 ('old', 6946),
 ('got', 6774),
 ('take', 6632),
 ('actors', 6612),
 ('give', 6556),
 ('years', 6517),
 ('part', 6515),
 ('young', 6340),
 ("that's", 6301),
 ('us', 6241),
 ('without', 6202),
 ('big', 6198),
 ('thought', 6184),
 ('things', 6168),
 ('around', 6085),
 ('it,', 6068),
 ('saw', 6051),
 ('gets', 6049),
 ('almost', 6020),
 ('must', 6004),
 ('though', 6000),
 ('director', 5960),
 ('always', 5898),
 ('whole', 5796),
 ('horror', 5766),
 ('come', 5765),
 ('work', 5689),
 ('might', 5653),
 ('"the', 5592),
 ('cast', 5545),
 ("he's", 5476),
 ('enough', 5430),
 ('bit', 5404),
 ('probably', 5365),
 ('least', 5359),
 ('feel', 5316),
 ('last', 5277),
 ('since', 5267),
 ('long', 5254),
 ('far', 5205),
 ('funny', 5163),
 ('kind', 5094),
 ('rather', 5063)]
```

Final Scan of Data before next stage

In [23]:

```
df.head()
```

Out[23]:

	review	sentiment
0	reviewers mentioned watching 1 oz episode hook...	positive
1	wonderful little production. filming technique...	positive
2	thought wonderful way spend time hot summer we...	positive
3	basically family little boy (jake) thinks zomb...	negative
4	petter mattei's "love time money" visually stu...	positive

Set up Data

In [24]:

```
X_train_raw, X_test_raw, y_train, y_test = train_test_split(df['review'], df['sentiment'],
```

In [25]:

```
### STEP 2 - Declare Features Vectors to use
### Create TfidfVectorizer.

vectorizer = TfidfVectorizer()

### STEP 3 - Fit and transform.
### Note there's no need to create TF-IDF vectors for y - labels

X_train = vectorizer.fit_transform(X_train_raw)

### Note vectorizer was fitted prior in X_train process
X_test = vectorizer.transform(X_test_raw)
```

Training & Evaluation

1. Logistic Regression

In [26]:

```
from sklearn.linear_model import LogisticRegression

### STEP 4 - Prediction
### Create and run Classifier

classifier = LogisticRegression()

### Fitting requires training TF_IDF vectors and labels
classifier.fit(X_train, y_train)

### X_test is the transformed test TF-IDF vectors
predictions = classifier.predict(X_test)
```

In [27]:

```
# testing against testing set

y_pred = classifier.predict(X_test)
print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test,y_pred))
```

```
[[4409  590]
 [ 414 4587]]
              precision    recall  f1-score   support

    negative       0.91      0.88      0.90      4999
    positive       0.89      0.92      0.90      5001

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

2. Naive Bayes

In [28]:

```
from sklearn.naive_bayes import MultinomialNB

### STEP 4 - Prediction
### Create and run Classifier

classifierNB = MultinomialNB()

### Fitting requires training TF_IDF vectors and labels
classifierNB.fit(X_train, y_train)

### X_test is the transformed test TF-IDF vectors
predictions = classifierNB.predict(X_test)
```

In [29]:

```
y_pred = classifierNB.predict(X_test)
print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test,y_pred))
```

```
[[4382  617]
 [ 714 4287]]
              precision    recall  f1-score   support

    negative       0.86      0.88      0.87      4999
    positive       0.87      0.86      0.87      5001

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000
```

3. Support Vector Machine

In [30]:

```
from sklearn.svm import LinearSVC

### STEP 4 - Prediction
### Create and run Classifier

classifierSVC = LinearSVC()

### Fitting requires training TF_IDF vectors and labels
classifierSVC.fit(X_train, y_train)

### X_test is the transformed test TF-IDF vectors
predictions = classifierSVC.predict(X_test)
```

In [31]:

```
y_pred = classifierSVC.predict(X_test)
print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test,y_pred))
```

```
[[4419  580]
 [ 444 4557]]
              precision    recall  f1-score   support

    negative       0.91      0.88      0.90      4999
    positive       0.89      0.91      0.90      5001

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

SVM gives the best results at 0.9 accuracy, precision & recall

Explore Prediction Output of Best Algorithm of SVM

In [32]:

```
X_test_raw.head()
```

Out[32]:

```
46331    hey guys girls! ever rent, god forbid, buy pie...
13679    wonderful "revenge" everyone wakes morning bel...
11177    timeless proverb reverberates heart. many year...
14515    hate show poorly leads written. women self-res...
46263    okay, guess pretty much fan spindled, mutilate...
Name: review, dtype: object
```

In [69]:

```
X_test_raw.shape
```

Out[69]:

```
(10000,)
```

In [34]:

```
y_test.head()
```

Out[34]:

```
46331    negative
13679    negative
11177    positive
14515    negative
46263    negative
Name: sentiment, dtype: object
```

In [70]:

```
y_pred.shape
```

Out[70]:

```
(10000,)
```

In [71]:

```
y_test.shape
```

Out[71]:

```
(10000,)
```

Create new dataframe with review, actual & predicted labels

In [77]:

```
df2 = pd.DataFrame(data = X_test_raw)
```

In [78]:

```
df2['truth']=pd.DataFrame(data=y_test)
```

In [79]:

```
c = y_pred.reshape((10000,1))
df2['predict']=c
```

In [80]:

```
df2.head()
```

Out[80]:

	review	truth	predict
46331	hey guys girls! ever rent, god forbid, buy pie...	negative	negative
13679	wonderful "revenge" everyone wakes morning bel...	negative	negative
11177	timeless proverb reverberates heart. many year...	positive	positive
14515	hate show poorly leads written. women self-res...	negative	negative
46263	okay, guess pretty much fan spindled, mutilate...	negative	negative

Create Dataframe of mis-labelled items

In [81]:

```
df3 = df2[df2['truth']!=df2['predict']]
```

In [82]:

```
df3.head()
```

Out[82]:

	review	truth	predict
14351	think great movie!! fun, maybe little unrealis...	negative	positive
9768	viewing, please make sure seen night living de...	positive	negative
23280	...out movie. sorry say, showed cleveland...	negative	positive
44259	ok start? saw screening couple weeks ago shock...	negative	positive
10914	gets score 3 dared different. features cast ac...	negative	positive

Total Mis-labelled items 1024 is consistent with Confusion Matrix items

In [83]:

```
df3.count()
```

Out[83]:

```
review     1024
truth      1024
predict    1024
dtype: int64
```

First Item: Actual Negative, Predict Positive

In [84]:

```
df3.iloc[0,0]
```

Out[84]:

```
'think great movie!! fun, maybe little unrealistic, fun dramatic!! like see
again, showing tv!! 1 question: still talking movie???'
```

Second Item: Actual Positive, Predict Negative

In [86]:

```
df3.iloc[1,0]
```

Out[86]:

```
"viewing, please make sure seen night living dead... might well best 7 minut
e parody ever seen! absurd, crappy 'special effects' (the rope, rope!!!), ma
neating slices bread... need???<br />(do watch eating bread... might get sca
red!)"
```

Third Item: Actual Negative, Predict Positive

In [87]:

```
df3.iloc[2,0]
```

Out[87]:

```
'...out movie.<br />sorry say, showed cleveland international festival. copy
subtitles, asked festival crew problem print received. "not so..." told. "th
e director wants way". />again, sorry say, french barely high school electiv
e level (more 3 decades ago). much initial dialog french, sure missed nuance
many details understanding key words. />i\'ve rated "1", primarily irony dir
ector worked subtitles refusing put subtitles seen american audience. excuse
me, even americans know europe map, even festival audience assumed know "the
native language" given even us know finnish, still expect subtitles "dolts"
sophisticated enough expertise 37 different languages presented. i\'ll put e
go david lynch, litvack.'
```

In [ ]:

```

```