Run the Cell to import the packages

In [1]:
```
import pandas as pd
import numpy as np
import csv
```

Fill in the Command to load your CSV dataset "imdb.csv" with pandas

In [2]:
```
#Data Loading
imdb=pd.read_csv("imdb.csv")
imdb.columns = ["index","text","label"]
print(imdb.head(5))
```

```
   index                                               text  label
0      0  A very, very, very slow-moving, aimless movie ...      0
1      1  Not sure who was more lost - the flat characte...      0
2      2  Attempting artiness with black & white and cle...      0
3      3       Very little music or anything to speak of.        0
4      4  The best scene in the movie was when Gerardo i...      1
```

Data Analysis

- Get the shape of the dataset and print it.
- Get the column names in list and print it.
- Group the dataset by **label** and describe the dataset to understand the basic statistics of the dataset.
- Print the first three rows of the dataset

In [3]:
```
data_size = imdb.shape

print(data_size)

imdb_col_names = list(imdb.columns)

print(imdb_col_names)

print(imdb.groupby('label').describe())

print(imdb.head(3))
```

```
(1000, 3)
['index', 'text', 'label']
       index
       count     mean         std  min     25%    50%     75%    max
label
0      500.0  466.418  276.272620  0.0  218.75  462.5  700.25  999.0
1      500.0  532.582  297.457084  4.0  297.75  569.5  787.25  993.0
   index                                               text  label
0      0  A very, very, very slow-moving, aimless movie ...      0
1      1  Not sure who was more lost - the flat characte...      0
2      2  Attempting artiness with black & white and cle...      0
```

Target Identification

Execute the below cell to identify the target variables. If 0 it is a bad review,if it is 1 it is a good review.

In [4]:
```
imdb_target=imdb['label']

print(imdb_target)
```

```
0      0
1      0
2      0
3      0
4      1
5      0
6      0
7      1
8      0
9      1
10     1
11     1
12     1
13     1
14     1
15     0
16     1
17     1
18     1
19     1
20     1
21     1
22     1
23     1
24     1
25     0
26     0
27     1
28     1
29     1
      ..
970    1
971    1
972    0
973    0
974    0
975    1
976    1
977    0
978    1
979    1
980    1
981    1
982    1
983    1
984    1
985    1
986    1
987    1
988    1
989    1
990    1
991    1
992    1
993    1
994    0
995    0
996    0
997    0
998    0
999    0
Name: label, Length: 1000, dtype: int64
```

Tokenization

- Convert the text into lower.
- Tokenize the text using word_tokenize
- Apply the function **split_tokens** for the column **text** in the **imdb** dataset with axis =1

In [5]:
```
from nltk.tokenize import word_tokenize

import nltk

nltk.download('all')

def split_tokens(text):

  text = text.lower()

  word_tokens = word_tokenize(text)

  return word_tokens

imdb['tokenized_message'] = imdb.apply(lambda row: split_tokens(row['text']), axis = 1)
```

```
[nltk_data]    | Downloading package universal_treebanks_v20 to
[nltk_data]    |     /home/user/nltk_data...
[nltk_data]    |   Package universal_treebanks_v20 is already up-to-
[nltk_data]    |       date!
[nltk_data]    | Downloading package verbnet to
[nltk_data]    |     /home/user/nltk_data...
[nltk_data]    |   Package verbnet is already up-to-date!
[nltk_data]    | Downloading package verbnet3 to
[nltk_data]    |     /home/user/nltk_data...
[nltk_data]    |   Package verbnet3 is already up-to-date!
[nltk_data]    | Downloading package webtext to
[nltk_data]    |     /home/user/nltk_data...
[nltk_data]    |   Package webtext is already up-to-date!
[nltk_data]    | Downloading package wordnet to
[nltk_data]    |     /home/user/nltk_data...
[nltk_data]    |   Package wordnet is already up-to-date!
[nltk_data]    | Downloading package wordnet_ic to
[nltk_data]    |     /home/user/nltk_data...
[nltk_data]    |   Package wordnet_ic is already up-to-date!
[nltk_data]    | Downloading package words to /home/user/nltk_data...
```

Lemmatization

- Apply the function **split_into_lemmas** for the column **tokenized_message** with axis=1
- Print the 55th row from the column **tokenized_message**.
- Print the 55th row from the column **lemmatized_message**

In [6]:
```
from nltk.stem.wordnet import WordNetLemmatizer

def split_into_lemmas(text):

    lemma = []

    lemmatizer = WordNetLemmatizer()

    for word in text:

        a=lemmatizer.lemmatize(word)

        lemma.append(a)

    return lemma

imdb['lemmatized_message'] = imdb.apply(lambda row: split_into_lemmas(row['tokenized_message']),axis=1)

print('Tokenized message:', imdb['tokenized_message'][55])

print('Lemmatized message:', imdb['lemmatized_message'][55])
```

```
Tokenized message: ['but', 'i', 'recommend', 'waiting', 'for', 'their', 'future', 'efforts', ',', 'let', 'this',
'one', 'go', '.']
Lemmatized message: ['but', 'i', 'recommend', 'waiting', 'for', 'their', 'future', 'effort', ',', 'let', 'this',
'one', 'go', '.']
```

Stop Word Removal

- Set the stop words language as english in the variable **stop_words**
- Apply the function **stopword_removal** to the column **lemmatized_message** with axis=1
- Print the 55th row from the column **preprocessed_message**

In [7]:
```
from nltk.corpus import stopwords

def stopword_removal(text):

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

    filtered_sentence = []

    filtered_sentence = ' '.join([word for word in text if word not in stop_words])

    return filtered_sentence

imdb['preprocessed_message'] = imdb.apply(lambda row: stopword_removal(row['lemmatized_message']),axis = 1)

print('Preprocessed message:',imdb['preprocessed_message'])

Training_data=pd.Series(list(imdb['preprocessed_message']))

Training_label=pd.Series(list(imdb['label']))
```

```
Preprocessed message: 0      , , slow-moving , aimless movie distressed , d...
1      sure wa lost - flat character audience , nearl...
2      attempting artiness black & white clever camer...
3                          little music anything speak .
4      best scene movie wa gerardo trying find song k...
5      rest movie lack art , charm , meaning ... 's e...
6                                    wasted two hour .
7      saw movie today thought wa good effort , good ...
8                                    bit predictable .
9           loved casting jimmy buffet science teacher .
10                                 baby owl adorable .
11     movie showed lot florida 's best , made look a...
12                         song best muppets hilarious .
13                                          wa cool .
14     `` right case '' movie delivers everything alm...
15     average acting main person , wa low budget cle...
16     review long overdue , since consider tale two ...
17     'll put gem movie term screenplay , cinematogr...
18     's practically perfect  true masterpiece sea ...
19     `` structure film easily tightly constructed h...
20     think film something vitally important occurs ...
21     word , content level film enough easily fill d...
22                anyone right mind ask anything movie ?
23     's quite simply highest , superlative form cin...
24     yes , film doe require rather significant amou...
25                     short film certainly pull punch .
26                          graphic far best part game .
27                       number one best th game series .
28                                 deserves strong love .
29                                         insane game .
                             ...
970              enough said remarkable animation film .
971    art style ha appearance crayon/pencil drawing ...
972    act film , glad 're gon na drift away earth fa...
973    one want surf small wave space movie 1998 ( de...
974    n't choked vomit end ( cheap drama worthless d...
975    still , make super ending depicts great sea ve...
976    consider excellent story , solid acting look f...
977    instead , got bore fest whiny , spoiled brat b...
978    watched two sunday ago ( march 20th , 2005 ) b...
979                            well acted done tv movie .
980    judith light one favorite actress think doe su...
981                                       keep watching .
982                                 's sad movie , good .
983                  seen movie , definitely recommend !
984                                 lovely usual , cutie !
985     still 's quite interesting entertaining follow .
986                             ; ) recommend confidence !
987    movie well-balanced comedy drama thoroughly en...
988    wa riot see hugo weaving play sex-obsessed gay...
989    : ) anyway , plot flowed smoothly male-bonding...
990    opening sequence gem classic , cat n mouse gam...
991                                    fan genre heaven .
992                          lange become great actress .
993                         looked like wonderful story .
994                           never walked movie faster .
995      got bored watching jessice lange take clothes !
996    unfortunately , virtue film 's production work...
997                                  word , embarrassing .
998                                    exceptionally bad !
```

```
999        insult one 's intelligence huge waste money .
Name: preprocessed_message, Length: 1000, dtype: object
```

Term Document Matrix

- Apply CountVectorizer with following parameters
  - ngram_range = (1,2)
  - min_df = (1/len(Training_label))
  - max_df = 0.7
- Fit the **tf_vectorizer** with the **Training_data**
- Transform the **Total_Dictionary_TDM** with the **Training_data**

In [8]:
```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer

tf_vectorizer = CountVectorizer(ngram_range = (1,2), min_df = (1/len(Training_label)),max_df = 0.7)

Total_Dictionary_TDM = tf_vectorizer.fit(Training_data)

message_data_TDM = Total_Dictionary_TDM.transform(Training_data)
```

Term Frequency Inverse Document Frequency (TFIDF)

- Apply TfidfVectorizer with following parameters
  - ngram_range = (1,2)
  - min_df = (1/len(Training_label))
  - max_df = 0.7
- Fit the **tfidf_vectorizer** with the **Training_data**
- Transform the **Total_Dictionary_TFIDF** with the **Training_data**

In [9]:
```
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(ngram_range = (1,2), min_df = (1/len(Training_label)),max_df = 0.7)

Total_Dictionary_TFIDF = tfidf_vectorizer.fit(Training_data)

message_data_TFIDF = Total_Dictionary_TFIDF.transform(Training_data)
```

Train and Test Data

Splitting the data for training and testing(90% train,10% test)

- Perform train-test split on **message_data_TDM** and **Training_label** with 90% as train data and 10% as test data.

In [10]:
```
from sklearn.model_selection import train_test_split#Splitting the data for training and testing

train_data,test_data, train_label, test_label = train_test_split(message_data_TDM,Training_label,test_size = 0.1)
```

Support Vector Machine

- Get the shape of the train-data and print the same.
- Get the shape of the test-data and print the same.
- Initialize SVM classifier with following parameters
  - kernel = linear
  - C= 0.025
  - random_state=seed
- Train the model with train_data and train_label
- Now predict the output with test_data
- Evaluate the classifier with score from test_data and test_label
- Print the predicted score

In [11]:
```
seed=9

from sklearn.svm import SVC

train_data_shape = train_data.shape

test_data_shape = test_data.shape

print("The shape of train data", train_data_shape)

print("The shape of test data", test_data_shape )

classifier = SVC(kernel="linear",C=0.025,random_state=seed)

classifier = classifier.fit(train_data,train_label)

#target =

score = classifier.fit(train_data,train_label)

print('SVM Classifier : ',score)

with open('output.txt', 'w') as file:

    file.write(str((imdb['tokenized_message'][55],imdb['lemmatized_message'][55])))
```

```
The shape of train data (900, 9051)
The shape of test data (100, 9051)
SVM Classifier :  SVC(C=0.025, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=9, shrinking=True, tol=0.001,
    verbose=False)
```

Stochastic Gradient Descent Classifier

- Perform train-test split on **message_data_TDM** and **Training_label** with this time 80% as train data and 20% as test data.
- Get the shape of the train-data and print the same.
- Get the shape of the test-data and print the same.
- Initialize SVM classifier with following parameters
  - loss = modified_huber
  - shuffle= True
  - random_state=seed
- Train the model with train_data and train_label
- Now predict the output with test_data
- Evaluate the classifier with score from test_data and test_label
- Print the predicted score

In [12]:
```
from sklearn.linear_model import SGDClassifier

train_data,test_data, train_label, test_label = train_test_split( message_data_TDM, Training_label, test_size = 0

train_data_shape = train_data.shape

test_data_shape = test_data.shape

print("The shape of train data", train_data_shape  )

print("The shape of test data", test_data_shape )

classifier =  SGDClassifier( loss='modified_huber',shuffle = True, random_state = seed )

classifier = classifier.fit(train_data,train_label)

#target=

score = classifier.score(test_data,test_label)

print('SGD classifier : ',score)

with open('output1.txt', 'w') as file:

    file.write(str((imdb['preprocessed_message'][55])))
```

```
The shape of train data (800, 9051)
The shape of test data (200, 9051)
SGD classifier :  0.76
```

In [ ]: