Importing packages

In [191]:
```
import os
import shutil
import pandas as pd
import numpy as np
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
import operator
```

Data Conversion and loading

In [62]:
```
data_load_train_pos = pd.read_csv('./dataset/train_dataset/train_pos.csv')
data_load_train_neg = pd.read_csv('./dataset/train_dataset/train_neg.csv')
data_load_test_pos = pd.read_csv('./dataset/test_dataset/test_pos.csv')
data_load_test_neg = pd.read_csv('./dataset/test_dataset/test_neg.csv')
```

In [63]:
```
frames_train = [data_load_train_pos,data_load_train_neg]
frames_test = [data_load_test_pos,data_load_test_neg]
```

In [64]:
```
dataset_1 = pd.concat(frames_train)
dataset_2 = pd.concat(frames_test)
```

In [65]:
```
dataset_1 = dataset_1.rename(columns = {'Reviews':0,'Class':1})
dataset_2 = dataset_2.rename(columns = {'Reviews':0,'Class':1})
```

In [66]:
```
dataset_1[1] = dataset_1[1].replace('positive',1)
dataset_1[1] = dataset_1[1].replace('negative',0)
dataset_2[1] = dataset_2[1].replace('positive',1)
dataset_2[1] = dataset_2[1].replace('negative',0)
```

In [67]:
```
dataset_1.to_csv('./dataset/final_pre_train_data.csv')
dataset_2.to_csv('./dataset/final_pre_test_data.csv')
```

In [68]:
```
df_train = pd.read_csv('./dataset/final_pre_train_data.csv')
df_train = df_train.drop(['Unnamed: 0'], axis= 1)
df_test = pd.read_csv('./dataset/final_pre_test_data.csv')
df_test = df_test.drop(['Unnamed: 0'], axis= 1)
```

In [69]:
```
x = df_train.iloc[:,0].values
y = df_train.iloc[:,-1].values
```

In [70]:
```
X_test=df_test.iloc[:,0].values
Y_test=df_test.iloc[:,1].values
```

Cleaning data

In [71]:
```
def cleanText(text):

    text = re.sub(r'<.*?>', ' ', text)
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "can not", text)
    text = re.sub(r"n't", " not", text)
    text = re.sub(r"'ve", " have", text)
    text = re.sub(r"'ll", " will", text)
    text = re.sub(r"'re", " are", text)

    text = re.sub(r"[0-9]+", ' ', text)
    text = re.sub(r"-", ' ', text)


    text = text.strip().lower()

    text = re.sub(r"'", ' ', text)


    filters='!"\'#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n'
    translate_dict = dict((i, " ") for i in filters)
    translate_map = str.maketrans(translate_dict)
    text = text.translate(translate_map)


    text = ' '.join([w for w in text.split() if len(w)>1])

    # Replace multiple space with one space
    text = re.sub(' +', ' ', text)

    text = ''.join(text)

    return text
```

In [72]:
```
for i in range(len(x)):
    x[i]=cleanText(x[i])

for i in range(len(X_test)):
    X_test[i]=cleanText(X_test[i])
```

Division of Data into Train, Development and Test

X train, Y train, X dev and Y dev

In [73]:
```
X_train, X_dev, Y_train, Y_dev = train_test_split(x, y, test_size = 0.25)
```

X test and Y test

In [74]:
```
X_test = df_test.iloc[0:,0]
Y_test = df_test.iloc[0:,-1]
```

Building Vocab as a list

In [79]:
```
stopwords = ["a","about","above","after","again","against","ain","all","am","an","and","any"
,"are","aren","aren't","as","at","be","because","been","before","being","below","between","b
oth","but","by","can","couldn","couldn't","d","did","didn","didn't","do","does","doesn","doe
sn't","doing","don","don't","down","during","each","few","for","from","further","had","hadn"
,"hadn't","has","hasn","hasn't","have","haven","haven't","having","he","her","here","hers",
"herself","him","himself","his","how","i","if","in","into","is","isn","isn't","it","it's","i
ts","itself","just","ll","m","ma","me","mightn","mightn't","more","most","mustn","mustn't",
"my","myself","needn","needn't","no","nor","not","now","o","of","off","on","once","only","o
r","other","our","ours","ourselves","out","over","own","re","s","same","shan","shan't","she"
,"she's","should","should've","shouldn","shouldn't","so","some","such","t","than","that","th
at'll","their","theirs","them","themselves","then","there","these","they","this","those","th
rough","to","too","under","until","up","ve","very","was","wasn","wasn't","we","were","weren"
,"weren't","what","when","where","which","while","who","whom","why","will","with","won","wo
n't","wouldn","wouldn't","y","you","you'd","you'll","you're","you've","your","yours","yourse
lf","yourselves","could","he'd","he'll","he's","here's","how's","i'd","i'll","i'm","i've","l
et's","ought","she'd","she'll","that's","there's","they'd","they'll","they're","they've","w
e'd","we'll","we're","we've","what's","when's","where's","who's","why's","would","able","abs
t","accordance","according","accordingly","across","act","actually","added","adj","affected"
,"affecting","affects","afterwards","ah","almost","alone","along","already","also","althoug
h","always","among","amongst","announce","another","anybody","anyhow","anymore","anyone","an
ything","anyway","anyways","anywhere","apparently","approximately","arent","arise","around",
"aside","ask","asking","auth","available","away","awfully","b","back","became","become","bec
omes","becoming","beforehand","begin","beginning","beginnings","begins","behind","believe",
"beside","besides","beyond","biol","brief","briefly","c","ca","came","cannot","can't","caus
e","causes","certain","certainly","co","com","come","comes","contain","containing","contain
s","couldnt","date","different","done","downwards","due","e","ed","edu","effect","eg","eigh
t","eighty","either","else","elsewhere","end","ending","enough","especially","et","etc","eve
n","ever","every","everybody","everyone","everything","everywhere","ex","except","f","far",
"ff","fifth","first","five","fix","followed","following","follows","former","formerly","fort
h","found","four","furthermore","g","gave","get","gets","getting","give","given","gives","gi
ving","go","goes","gone","got","gotten","h","happens","hardly","hed","hence","hereafter","he
reby","herein","heres","hereupon","hes","hi","hid","hither","home","howbeit","however","hund
red","id","ie","im","immediate","immediately","importance","important","inc","indeed","inde
x","information","instead","invention","inward","itd","it'll","j","k","keep","keeps","kept",
"kg","km","know","known","knows","l","largely","last","lately","later","latter","latterly",
"least","less","lest","let","lets","like","liked","likely","line","little","'ll","look","loo
king","looks","ltd","made","mainly","make","makes","many","may","maybe","mean","means","mean
time","meanwhile","merely","mg","might","million","miss","ml","moreover","mostly","mr","mrs"
,"much","mug","must","n","na","name","namely","nay","nd","near","nearly","necessarily","nece
ssary","need","needs","neither","never","nevertheless","new","next","nine","ninety","nobody"
,"non","none","nonetheless","noone","normally","nos","noted","nothing","nowhere","obtain","o
btained","obviously","often","oh","ok","okay","old","omitted","one","ones","onto","ord","oth
ers","otherwise","outside","overall","owing","p","page","pages","part","particular","particu
larly","past","per","perhaps","placed","please","plus","poorly","possible","possibly","poten
tially","pp","predominantly","present","previously","primarily","probably","promptly","prou
d","provides","put","q","que","quickly","quite","qv","r","ran","rather","rd","readily","real
ly","recent","recently","ref","refs","regarding","regardless","regards","related","relativel
y","research","respectively","resulted","resulting","results","right","run","said","saw","sa
y","saying","says","sec","section","see","seeing","seem","seemed","seeming","seems","seen",
"self","selves","sent","seven","several","shall","shed","shes","show","showed","shown","show
ns","shows","significant","significantly","similar","similarly","since","six","slightly","so
mebody","somehow","someone","somethan","something","sometime","sometimes","somewhat","somewh
ere","soon","sorry","specifically","specified","specify","specifying","still","stop","strong
ly","sub","substantially","successfully","sufficiently","suggest","sup","sure","take","take
n","taking","tell","tends","th","thank","thanks","thanx","thats","that've","thence","thereaf
ter","thereby","thered","therefore","therein","there'll","thereof","therere","theres","there
to","thereupon","there've","theyd","theyre","think","thou","though","thoughh","thousand","th
roug","throughout","thru","thus","til","tip","together","took","toward","towards","tried","t
```

In [80]:
```
len(stopwords)
```

Out[80]:
```
783
```

In [81]:
```
%%time
word_frequency = dict()
for reviews in X_train:
    words = reviews.split()
    for word in words:
        if word not in word_frequency.keys():
            word_frequency[word]=1
        else:
            word_frequency[word]+=1
```

```
Wall time: 1.68 s
```

In [83]:
```
word_frequency['the']
```

Out[83]:
```
251004
```

Remove words with frequency less than 5

In [84]:
```
words_less_than_5 = list()
for i in word_frequency.keys():
    if word_frequency[i]<5:
        words_less_than_5.append(i)

for i in words_less_than_5:
    del word_frequency[i]
```

Remove stop words from the above list

In [85]:
```
for i in stopwords:
    if i in word_frequency.keys():
        del word_frequency[i]
```

In [86]:
```
len(word_frequency)
```

Out[86]:
```
24499
```

In [87]:
```
print(X_train[526])
len(X_train[526])
```

```
nothing new in this hackneyed romance with characters put into unbelievable situations speaki
ng dialogue that borders on the ridiculous this is an example of another movie put into produ
ction before serious script problems were solved do not waste your time
```

Out[87]:
```
257
```

Calculating number of documents containing a particular word

In [88]:
```
def word_count(x,wordfrequency):
    wordcount = {}
    for word in wordfrequency.keys():
        count = 0
        for review in x:
            if word in review:
                count += 1
        wordcount[word]= count
    return wordcount
```

In [89]:
```
%%time
wordcount =  word_count(X_train,word_frequency)
```

```
Wall time: 8min 21s
```

In [90]:
```
len(wordcount)
```

Out[90]:
```
24499
```

Calculate Probability

Probability of the occurance

In [91]:
```
def probability_occurance(word,dataset):
    count = 0
    if word not in wordcount.keys():
        count
    else:
        count = wordcount[word]
    return count/len(dataset)
```

In [92]:
```
print('P |the| = ',probability_occurance('the',X_train))
```

```
P |the| =  0.9959466666666666
```

Conditional probability based on the sentiment

In [98]:
```
def word_positive(x,y,wordfrequency):
    wordpositive = {}
    for word in wordfrequency.keys():
        count = 0
        for index, reviews in enumerate(x):
            if word in reviews and y[index]==1:
                count+=1
        wordpositive[word] = count
    return wordpositive

def word_negative(x,y,wordfrequency):
    wordnegative = {}
    for word in wordfrequency.keys():
        count = 0
        for index, reviews in enumerate(x):
            if word in reviews and y[index]==0:
                count+=1
        wordnegative[word] = count
    return wordnegative
```

In [94]:
```
%%time

wordpositive=word_positive(X_train,Y_train,word_frequency)
```

```
Wall time: 7min 45s
```

In [95]:
```
len(wordpositive)
```

Out[95]:
```
24499
```

In [101]:
```
%%time

wordnegative=word_negative(X_train,Y_train,word_frequency)
```

```
Wall time: 7min 36s
```

In [103]:
```
len(wordnegative)
```

Out[103]:
```
24499
```

occurance of the 'the' in positive document

In [ ]:
```
wordpositive['the']
```

P[“the” | Positive] = # of positive documents containing “the” / num of all positive review documents

bayes rule:

P|positive/the|=(P|the/positive|* P|positive|)/P|the|

In [207]:
```
def total_sentiment(Y ,sentiment):
    length=0
    for i in Y:
        if i == sentiment:
            length+=1
    return length

def conditional_probability(X,Y,word,sentiment,length):
    if sentiment ==1:
        count=0
        if word not in wordpositive.keys():
            count
        else:
            count = wordpositive[word]
    else:
        count=0
        if word not in wordnegative.keys():
            count
        else:
            count = wordnegative[word]
    return count/length

def sentiment_probability(Y,sentiment):
    count=0
    for i in Y:
        if i==sentiment:
            count+=1
    return count/len(Y)

def calBayes(X,Y,word,sentiment,length):
    a = conditional_probability(X,Y,word,sentiment,length)
    b = length/len(Y)
    c = probability_occurance(word,X)
    if c!= 0:
        c = c
    else:
        c = 0.000001
    return ((a*b)/float(c))

def calsmoothBayes(X,Y,word,sentiment,length):
    count = 0
    a = conditional_probability(X,Y,word,sentiment,length)
    b = length/len(Y)
    c = probability_occurance(word,X)
    if word in wordcount.keys():
        count = wordcount[word]
    else:
        count = 2
    return  (((a*b)+1)/(c+count))

def review_sentiment(X,Y,review,sentiment,smooth):
    summ = 1
    length = total_sentiment(Y_train,1)
    if smooth == True:
        for word in review.split():
            prob = calsmoothBayes(X,Y,word,sentiment,length)
            summ *= prob
    else:
        for word in review.split():
            prob = calBayes(X,Y,word,sentiment,length)
            summ *= prob
    return summ

def naivesBayes(X,Y,review,smooth):
    pos_rev = review_sentiment(X,Y,review,1,smooth)
    neg_rev = review_sentiment(X,Y,review,0,smooth)
    if pos_rev > neg_rev:
        return 1
    else :
        return 0

def naiveBayesGlobal(X,Y,x_test,smooth):
    ypred=[]
    for review in x_test:
        pred=naivesBayes(X,Y,review,smooth)
        ypred.append(pred)
    return ypred

def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

In [208]:
```
length = total_sentiment(Y_train ,1)
print('P|the/positive| = ',conditional_probability(X_train,Y_train,'the',1,length))
```

```
P|the/positive| =  0.995826198630137
```

In [209]:
```
length = total_sentiment(Y_train ,1)
calBayes(X_train,Y_train,'the',1,length)
```

Out[209]:
```
0.4982863874906287
```

Predicting the sentiment for Training Data

In [210]:
```
print('Actual Label: {}'.format(Y_train[0]))
```

```
Actual Label: 1
```

In [211]:
```
review=X_train[0]
pred =naivesBayes(X_train,Y_train,review,smooth=False)
print('Predicted Label :',pred)
```

```
Predicted Label : 0
```

In [212]:
```
y_pred=naiveBayesGlobal(X_train,Y_train,X_dev,smooth=False)
uncleanscore=accuracy_metric(Y_dev, y_pred)
```

In [213]:
```
print('Accuracy on development set : {} %'.format(uncleanscore))
```

```
Accuracy on development set : 82.464 %
```

Effect of smoothing

In [214]:
```
y_pred=naiveBayesGlobal(X_train,Y_train,X_dev,smooth=True)
uncleanscore=accuracy_metric(Y_dev, y_pred)
```

In [215]:
```
print('Accuracy on development set : {} %'.format(uncleanscore))
```

```
Accuracy on development set : 68.06400000000001 %
```

Accuracy for cleaned dataset

In [216]:
```
def remove_extra_words(dataset,wordfreq):
    for index ,review in enumerate(dataset):
        cleanreview=[]
        for word in review.split():
            if word in wordfreq.keys():
                cleanreview.append(word)
        cleanreview=' '.join(cleanreview)
        dataset[index]=cleanreview
    return dataset
```

In [217]:
```
xdev=X_dev.copy()
xdev=remove_extra_words(X_dev,word_frequency)
y_pred=naiveBayesGlobal(X_train,Y_train,xdev,smooth=False)
cleanedscore=accuracy_metric(Y_dev, y_pred)
```

In [218]:
```
print('Accuracy on cleaned development set : {} %'.format(cleanedscore))
```

```
Accuracy on cleaned development set : 82.464 %
```

Derive Top 10 words that predicts positive and negative class

In [219]:
```
totalpositive = total_sentiment(Y_train ,1)
toppositive={}
for word in word_frequency.keys():
    result=calBayes(X_train,Y_train,word,1,totalpositive)
    toppositive[word]=result

totalnegative = total_sentiment(Y_train ,0)
topnegative={}
for word in word_frequency.keys():
    result=calBayes(X_train,Y_train,word,0,totalnegative)
    topnegative[word]=result
```

In [220]:
```
top10pos = dict(sorted(toppositive.items(), key=operator.itemgetter(1), reverse=True)[:10])
top10neg = dict(sorted(topnegative.items(), key=operator.itemgetter(1), reverse=True)[:10])
```

Top 10 positive words

In [221]:
```
for word in top10pos.keys():
    print(word)
```

```
krell
yokai
scola
dickey
gypo
cheadle
visconti
girotti
pappy
waxworks
```

Top 10 negative words

In [222]:
```
for word in top10neg.keys():
    print(word)
```

```
frakes
sponsors
vishal
colagrande
giada
completists
dax
guernsey
greydon
drearily
```

Using the test dataset

Use the optimal hyperparameters you found and use it to calculate the final accuracy.

In [223]:
```
X_test=remove_extra_words(X_test,word_frequency)
y_pred=naiveBayesGlobal(X_train,Y_train,X_test,smooth=True)
Finalscore=accuracy_metric(Y_test, y_pred)
```

In [225]:
```
print('Final Accuracy : {} %'.format(Finalscore))
```

```
Final Accuracy : 69.092 %
```