In [1]:
```
%matplotlib inline

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
```

In [4]:
```
conn = sqlite3.connect('final.sqlite')
```

In [ ]:
```
c = conn.cursor()
```

In [ ]:
```
final = pd.read_sql_query("""SELECT * FROM Reviews;""", conn)
final.head()
```

In [ ]:
```
## Encoding the labels

# Label encoding the "Score"
label = LabelEncoder()
score_labels = label.fit_transform(final['Score'])
# Adding new column to the dataset that are label - encoded.
final['score_labels'] = score_labels
```

In [ ]:
```
neg = final[final['score_labels'] == 0].reset_index().drop('index', axis = 1)
pos = final[final['score_labels'] == 1].reset_index().drop('index', axis = 1)
```

In [ ]:
```
print(neg.shape, pos.shape)
```

In [ ]:
```
np.random.seed(100)
indices = np.random.choice(np.arange(57110),replace=False, size= 5000)
neg = neg.loc[indices]
print("Negative reviews shape:", neg.shape)
indices = np.random.choice(np.arange(307061),replace=False, size = 5000)
pos = pos.loc[indices]
print("Positive reviews shape", pos.shape)
fil_df = pd.concat([neg, pos])
```

In [ ]:
```
fil_df.sort_values(by='Time', inplace=True)
```

In [ ]:
```
final_df = fil_df.drop('level_0', axis=1)
```

In [ ]:
```
final_df.to_csv("final_df.csv", index=False)
```

In [2]:
```
final_df = pd.read_csv('final_df.csv', encoding='cp1252')
```

In [3]:
```
final_df.head()
```

Out[3]:

| | Id | ProductId | UserId | ProfileName | HelpfulnessNumerator | H |
|---|---|---|---|---|---|---|
| **0** | 361319 | B00005IX96 | AGUF1WPEG4GSM | "lchang44" | 5 | 8 |
| **1** | 193108 | B0000DJDJR | A3F6UNXVI9LSMA | Samuel H. Wheeler "bigdaddysam" | 7 | 7 |
| **2** | 30629 | B00008RCMI | A19E94CF5O1LY7 | Andrew Arnold | 0 | 0 |
| **3** | 434425 | B0000CA4TK | A5VIGE8EO86RI | captmorgan1670 "captmorgan1670" | 4 | 4 |
| **4** | 333669 | B0000UBTYG | A6BS5D5YPF2HW | MT | 3 | 3 |

# BOW

In [14]:
```
## Computing binary BOW.
count_vect = CountVectorizer()
final_counts = count_vect.fit_transform(final_df['Text'].values)
final_counts = final_counts.toarray()
```

In [15]:
```
np.save('final_bow.npy',final_counts)
```

## Bi-grams and n-grams

In [16]:
```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
count_vect = CountVectorizer(ngram_range=(1,2) ) #in scikit-learn
final_bigram_counts = count_vect.fit_transform(final_df['Text'].values)
```

In [17]:
```
final_bigram_counts = final_bigram_counts.toarray()
```

In [18]:
```
np.save("bigram.npy", final_bigram_counts)
```

## TF - IDF(Unigram)

In [4]:
```
tf_idf_vect = TfidfVectorizer()
final_tf_idf = tf_idf_vect.fit_transform(final_df['Text'].values)
final_tf_idf = final_tf_idf.toarray()
```

In [5]:
```
np.save('final-unigram-tfidf.npy', final_tf_idf)
```

In [21]:
```
tf_idf_vect = np.load('Word Vectors//final-tfidf.npy', mmap_mode='r+')
```

## TF-IDF(Unigram + Bigram)

In [ ]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(final_df['Text'].values)
final_tf_idf = final_tf_idf.toarray()
```

In [ ]:
```
np.save('final-tfidf.npy', final_tf_idf)
```

## word2vec

In [6]:
```
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

stop = set(stopwords.words('english')) #set of stopwords
sno = nltk.stem.SnowballStemmer('english') #initialising the snowball stemmer

def cleanhtml(sentence): #function to clean the word of any html-tags
    cleanr = re.compile('<.*?>')
    cleantext = re.sub(cleanr, ' ', sentence)
    return cleantext
def cleanpunc(sentence): #function to clean the word of any punctuation or spe
cial characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

In [7]:
```
# Train your own Word2Vec model using your own text corpus
import gensim
i=0
list_of_sent=[]
for sent in final_df['Text'].values:
    filtered_sentence=[]
    sent=cleanhtml(sent)
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if(cleaned_words.isalpha()):
                filtered_sentence.append(cleaned_words.lower())
            else:
                continue
    list_of_sent.append(filtered_sentence)
```

```
C:\Users\rahul\AppData\Local\conda\conda\envs\my_root\lib\site-packages\gensi
m\utils.py:860: UserWarning: detected Windows; aliasing chunkize to chunkize_
serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [23]:
```
w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
```

In [24]:
```
words = list(w2v_model.wv.vocab)
print(len(words))
```

```
6472
```

In [9]:
```
np.save('list_of_sent.npy', list_of_sent)
```

# Avg W2V, TFIDF - W2V

In [25]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this li
st
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
10000
50
```

In [26]:
```
np.save('avg-W2V.npy', sent_vectors)
```

In [27]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val
 = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
 this list
row=0;
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        try:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass
    sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [28]:
```
np.save('tf-idf-w2v.npy', tfidf_sent_vectors)
```

In [29]:
```
tf_idf_w2v = np.load('tf-idf-w2v.npy')
```

# Google pre trained W2V and tfidf weighted W2V.

In [10]:
```
# Using Google News Word2Vectors
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


model = KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin'
, binary=True)
```

In [15]:
```
len(model.wv['computer'])
```

Out[15]:
```
300
```

### Google AVG - W2V.

In [16]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this li
st
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        try:
            vec = model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
10000
300
```

In [17]:
```
np.save('final-google-avg-w2v.npy',sent_vectors)
```

### Google TFIDF weighted AVG - W2V.

In [28]:
```
# TF-IDF weighted Word2Vec
tfidf_feat = tf_idf_vect.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val
 = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in
 this list
row=0;
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(300) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        try:
            vec = model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass
    sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [29]:
```
np.save('final-google-tfidf-avg-w2v.npy', tfidf_sent_vectors)
```