```
In [1]: #to ignore warnings
import warnings
warnings.filterwarnings("ignore")
#to use sqlite3 database
import sqlite3
import numpy as np
import pandas as pd
import string
import nltk
import matplotlib.pyplot as plt
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import cross_validation
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cross_validation import cross_val_score
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
C:\Users\krush\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: T
his module was deprecated in version 0.18 in favor of the model_selection module into which all
the refactored classes and functions are moved. Also note that the interface of the new CV itera
tors are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [2]: con = sqlite3.connect('database.sqlite')

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, con)


# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rating.
def partition(x):
    if x < 3:
        return 'negative'
    return 'positive'

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
```

Text Preprocessing on all data points

```
In [3]: 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 special characters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
```

```
In [4]: #Code for implementing step-by-step the checks mentioned in the pre-processing phase
# this code takes a while to run as it needs to run on 500k sentences.
i=0
str1=' '
final_string=[]
all_positive_words=[] # store words from +ve reviews here
all_negative_words=[] # store words from -ve reviews here.
s=''
for sent in filtered_data['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTMl tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (filtered_data['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to describe positive re
views
                    if(filtered_data['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describe negative re
views reviews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("***********************************************************************")

    final_string.append(str1)
    i+=1
```

```
In [5]: filtered_data['CleanedText']=final_string #adding a column of CleanedText which displays the data af
ter pre-processing of the review
filtered_data['CleanedText']=filtered_data['CleanedText'].str.decode("utf-8")
```

Sort the datapoints according to time and take first 50000 points

```
In [6]: sorted_data=filtered_data.sort_values(by=['Time'])
sampledata = sorted_data.head(50000)#

S = sorted_data['Score']
Score = S.head(50000)
```

Splitting

```
In [7]: # HERE WE ARE SPLITTING THE DATA POINTS IN TO 80% TRAIN AND 20% FOR TEST
X_1, X_test, y_1, y_test = cross_validation.train_test_split(sampledata, Score, test_size=0.2, rando
m_state=0)
#HERE WE ARE AGAIN SPLITTING THE TRAIN DATA IN EARLIER LINE X_1 IN TO 75% TRAINING AND 25% CROSS VAL
IDATION DATA so we have 60% 20% 20% ratio
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.25)
```

BAG OF WORDS

```
In [8]: count_vect = CountVectorizer() #in scikit-learn
vec = count_vect.fit(X_tr['CleanedText'].values)
```

```
In [9]: X_trvec = vec.transform(X_tr['CleanedText'].values)
X_cvvec = vec.transform(X_cv['CleanedText'].values)
X_testvec = vec.transform(X_test['CleanedText'].values)
```

```
In [10]: #As you mentioned no need of K-fold as i did in last submission
listacc=[]
myList = list(range(0,20))
neighbors = list(filter(lambda x: x % 2 != 0, myList))
for i in neighbors:
    # instantiate learning model (k = 30)
    knn = KNeighborsClassifier(n_neighbors=i)

    # fitting the model on crossvalidation train
    knn.fit(X_trvec, y_tr)

    # predict the response on the crossvalidation train
    pred = knn.predict(X_cvvec)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))
    listacc.append(acc)
MSE = [100 - x for x in listacc]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)
```

```
CV accuracy for k = 1 is 87%

CV accuracy for k = 3 is 89%

CV accuracy for k = 7 is 89%

CV accuracy for k = 9 is 89%

CV accuracy for k = 11 is 89%

CV accuracy for k = 13 is 88%

CV accuracy for k = 15 is 88%

CV accuracy for k = 17 is 88%

CV accuracy for k = 19 is 88%

The optimal number of neighbors is 5.
```

```
In [11]:   print('\nCV error for k = %d is %d%%' % (optimal_k, min(MSE)))
```

```
CV error for k = 5 is 10%
```

```
In [12]: knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='brute')

# fitting the model
knn_optimal.fit(X_trvec, y_tr)

# predict the response
pred = knn_optimal.predict(X_testvec)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
print('\nThe Test error of the knn classifier for k = %d is %f%%' % (optimal_k, 100-acc))
```

```
The accuracy of the knn classifier for k = 5 is 90.240000%

The Test error of the knn classifier for k = 5 is 9.760000%
```

previously with less points i was not able to decide is kd_tree is faster or not now i am able to find that kdtree is faster

I am doing dimentionality reduction to avoid "curse of dimentionality" in kdtree

```
In [13]: from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2, n_iter=7, random_state=999)
X_trvec1=svd.fit_transform(X_trvec)
X_testvec1=svd.fit_transform(X_testvec)
```

```
In [14]: knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_tree')

# fitting the model
knn_optimal.fit(X_trvec1, y_tr)

# predict the response
pred = knn_optimal.predict(X_testvec1)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
print('\nThe Test error of the knn classifier for k = %d is %f%%' % (optimal_k, 100-acc))
```

```
The accuracy of the knn classifier for k = 5 is 87.010000%

The Test error of the knn classifier for k = 5 is 12.990000%
```

Confusion matrix

It consists 4 things • True Positive (TP) : Observation is positive, and is predicted to be positive.[0,0] • False Negative (FN) : Observation is positive, but is predicted negative.[0,1] • True Negative (TN) : Observation is negative, and is predicted to be negative.[1,1] • False Positive (FP) : Observation is negative, but is predicted positive.[1,0] indexes

If the data is imbalanced , then we will use these things as accuracy is not a correct measure and we will also have other metrix to measure using confusion matrix like 1)precision = TP/TP+FP 2)Recall = TP/TP+FN 3)F1 score = 2*recall*precision/recall + precision

```
    These can also be calculated directly from sklearn...
```

```
In [15]: from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, pred))
```

```
[[  65 1010]
 [ 289 8636]]
```

```
In [16]: from sklearn.datasets import make_classification
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_
report, confusion_matrix

print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.5105124700810639
0.5394546025342316
0.5140420819490586
```

TF-IDF

```
In [17]: tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit(X_tr['CleanedText'].values)
```

```
In [18]: X_tr_tf_idf = final_tf_idf.transform(X_tr['CleanedText'].values)
X_cv_tf_idf = final_tf_idf.transform(X_cv['CleanedText'].values)
X_test_tf_idf = final_tf_idf.transform(X_test['CleanedText'].values)
```

```
In [19]: listacc=[]
myList = list(range(0,20))
neighbors = list(filter(lambda x: x % 2 != 0, myList))
for i in neighbors:
    # instantiate learning model (k = 30)
    knn = KNeighborsClassifier(n_neighbors=i)

    # fitting the model on crossvalidation train
    knn.fit(X_tr_tf_idf, y_tr)

    # predict the response on the crossvalidation train
    pred = knn.predict(X_cv_tf_idf)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))
    listacc.append(acc)
MSE = [100 - x for x in listacc]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)
```

```
CV accuracy for k = 1 is 90%

CV accuracy for k = 3 is 90%

CV accuracy for k = 5 is 90%

CV accuracy for k = 7 is 90%

CV accuracy for k = 9 is 90%

CV accuracy for k = 11 is 90%

CV accuracy for k = 13 is 89%

CV accuracy for k = 15 is 89%

CV accuracy for k = 17 is 89%

CV accuracy for k = 19 is 89%

The optimal number of neighbors is 3.
```

```
In [20]:   print('\nCV error for k = %d is %d%%' % (optimal_k, min(MSE)))
```

```
CV error for k = 3 is 9%
```

```
In [21]: knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='brute')

# fitting the model
knn_optimal.fit(X_tr_tf_idf, y_tr)

# predict the response
pred = knn_optimal.predict(X_test_tf_idf)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
print('\nThe Test error of the knn classifier for k = %d is %f%%' % (optimal_k, 100-acc))
```

```
The accuracy of the knn classifier for k = 3 is 91.570000%

The Test error of the knn classifier for k = 3 is 8.430000%
```

```
In [22]: X_tr_tf_idf1=svd.fit_transform(X_tr_tf_idf)
X_test_tf_idf1=svd.fit_transform(X_test_tf_idf)
```

```
In [23]: knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_tree')

# fitting the model
knn_optimal.fit(X_tr_tf_idf1, y_tr)

# predict the response
pred = knn_optimal.predict(X_test_tf_idf1)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
print('\nThe Test error of the knn classifier for k = %d is %f%%' % (optimal_k, 100-acc))
```

```
The accuracy of the knn classifier for k = 3 is 85.850000%

The Test error of the knn classifier for k = 3 is 14.150000%
```

```
In [24]: from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, pred))
```

```
[[  51 1024]
 [ 391 8534]]
```

```
In [25]: from sklearn.datasets import make_classification
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_
report, confusion_matrix

print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.495340566832762
0.5041246157065365
0.5018161683277962
```

Word to VEC

```
In [26]: from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
i=0
list_of_sent=[]
for sent in X_tr['CleanedText'].values:
    list_of_sent.append(sent.split())
w2v_model=Word2Vec(list_of_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
```

AVG WORD 2 VEC

```
In [27]: from tqdm import tqdm
import os
sent_vectorstr = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(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
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectorstr.append(sent_vec)
```

```
100%|██████████████████████████████████████████████████| 30000/30000 [0
0:26<00:00, 1132.97it/s]
```

```
In [28]: list_of_sent1 = []
for sent in X_test['CleanedText'].values:
    list_of_sent1.append(sent.split())
sent_vectorstest = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent1): # 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
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectorstest.append(sent_vec)
```

```
100%|██████████████████████████████████████████████████| 10000/10000 [0
0:09<00:00, 1075.95it/s]
```

```
In [29]: list_of_sent2 = []
for sent in X_cv['CleanedText'].values:
    list_of_sent2.append(sent.split())
sent_vectorscv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent2): # 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
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectorscv.append(sent_vec)
```

```
100%|██████████████████████████████████████████████████| 10000/10000 [0
0:09<00:00, 1086.55it/s]
```

```
In [30]: listacc=[]
myList = list(range(0,20))
neighbors = list(filter(lambda x: x % 2 != 0, myList))
for i in neighbors:
    # instantiate learning model (k = 30)
    knn = KNeighborsClassifier(n_neighbors=i)

    # fitting the model on crossvalidation train
    knn.fit(sent_vectorstr, y_tr)

    # predict the response on the crossvalidation train
    pred = knn.predict(sent_vectorscv)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))
    listacc.append(acc)
MSE = [100 - x for x in listacc]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)
```

```
CV accuracy for k = 1 is 90%

CV accuracy for k = 3 is 91%

CV accuracy for k = 5 is 90%

CV accuracy for k = 7 is 90%

CV accuracy for k = 9 is 90%

CV accuracy for k = 11 is 90%

CV accuracy for k = 13 is 90%

CV accuracy for k = 15 is 90%

CV accuracy for k = 17 is 89%

CV accuracy for k = 19 is 89%

The optimal number of neighbors is 3.
```

```
In [31]: knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='brute')

# fitting the model
knn_optimal.fit(sent_vectorstr, y_tr)

# predict the response
pred = knn_optimal.predict(sent_vectorstest)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
print('\nThe Test error of the knn classifier for k = %d is %f%%' % (optimal_k, 100-acc))
```

```
The accuracy of the knn classifier for k = 3 is 91.700000%

The Test error of the knn classifier for k = 3 is 8.300000%
```

```
In [32]: sent_vectorstr1=svd.fit_transform(sent_vectorstr)
sent_vectorstest1=svd.fit_transform(sent_vectorstest)
```

```
In [33]: knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_tree')

# fitting the model
knn_optimal.fit(sent_vectorstr1, y_tr)

# predict the response
pred = knn_optimal.predict(sent_vectorstest1)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
print('\nThe Test error of the knn classifier for k = %d is %f%%' % (optimal_k, 100-acc))
```

```
The accuracy of the knn classifier for k = 3 is 85.670000%

The Test error of the knn classifier for k = 3 is 14.330000%
```

```
In [34]: from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2, n_iter=7, random_state=999)
X_tr_tf_idf1=svd.fit_transform(X_tr_tf_idf)
```

```
In [35]: from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, pred))
```

```
[[ 118  957]
 [ 476 8449]]
```

```
In [36]: from sklearn.datasets import make_classification
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_
report, confusion_matrix

print(f1_score(y_test, pred, average="macro"))
print(precision_score(y_test, pred, average="macro"))
print(recall_score(y_test, pred, average="macro"))
```

```
0.53161422570945
0.5484548153589192
0.5282170542635659
```

TF-IDF WEIGHTED W2V

```
In [37]: from tqdm import tqdm
import os
# TF-IDF weighted Word2Vec

tfidf_feat = tf_idf_vect.get_feature_names()
dictionary = dict(zip(tf_idf_vect.get_feature_names(), list(tf_idf_vect.idf_)))# tfidf words/col-nam
es
# 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 tqdm(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
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████| 30000/30000 [0
0:36<00:00, 832.78it/s]
```

```
In [38]: # 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_test = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent1): # 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
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████| 10000/10000 [0
0:11<00:00, 838.84it/s]
```

```
In [39]: # 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_cv = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sent2): # 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
        if word in w2v_words:
            vec = w2v_model.wv[word]
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_cv.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████| 10000/10000 [0
0:11<00:00, 891.54it/s]
```

```
In [40]: listacc=[]
myList = list(range(0,20))
neighbors = list(filter(lambda x: x % 2 != 0, myList))
for i in neighbors:
    # instantiate learning model (k = 30)
    knn = KNeighborsClassifier(n_neighbors=i)

    # fitting the model on crossvalidation train
    knn.fit(tfidf_sent_vectors, y_tr)

    # predict the response on the crossvalidation train
    pred = knn.predict(tfidf_sent_vectors_cv)

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
    print('\nCV accuracy for k = %d is %d%%' % (i, acc))
    listacc.append(acc)
MSE = [100 - x for x in listacc]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('\nThe optimal number of neighbors is %d.' % optimal_k)
```

```
CV accuracy for k = 1 is 89%

CV accuracy for k = 3 is 90%

CV accuracy for k = 5 is 90%

CV accuracy for k = 7 is 90%

CV accuracy for k = 9 is 90%

CV accuracy for k = 11 is 90%

CV accuracy for k = 13 is 89%

CV accuracy for k = 15 is 89%

CV accuracy for k = 17 is 89%

CV accuracy for k = 19 is 89%

The optimal number of neighbors is 5.
```

```
In [41]: knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='brute')

# fitting the model
knn_optimal.fit(tfidf_sent_vectors, y_tr)

# predict the response
pred = knn_optimal.predict(tfidf_sent_vectors_test)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
print('\nThe Test error of the knn classifier for k = %d is %f%%' % (optimal_k, 100-acc))
```

```
The accuracy of the knn classifier for k = 5 is 91.150000%

The Test error of the knn classifier for k = 5 is 8.850000%
```

```
In [42]: tfidf_sent_vectors1=svd.fit_transform(tfidf_sent_vectors)
tfidf_sent_vectors_test1=svd.fit_transform(tfidf_sent_vectors_test)
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_tree')

# fitting the model
knn_optimal.fit(tfidf_sent_vectors1, y_tr)

# predict the response
pred = knn_optimal.predict(tfidf_sent_vectors_test1)

# evaluate accuracy
acc = accuracy_score(y_test, pred) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc))
print('\nThe Test error of the knn classifier for k = %d is %f%%' % (optimal_k, 100-acc))
```

```
The accuracy of the knn classifier for k = 5 is 86.780000%

The Test error of the knn classifier for k = 5 is 13.220000%
```

```
In [43]: from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, pred))
```

```
[[  84  991]
 [ 331 8594]]
```