Objective: To determine k value(number of nearest neighbors)and analysing the metrics of each model by applying simple cross validation for Amazon food review dataset using feature to vectorization techniques namely BoW, TFIDF, Average w2v and tf_idf weighted w2v. Algorithms used: Kd_tree and brute

Note: Sampled 60000 datapoints for Bow, Average w2v and tf_idf weighted w2v. For TFIDF, due to memory issue, sampled 10000 points only.

In [8]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

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.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


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

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os



# ============================== loading libraries ===========================================
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
# =============================================================================================
```

In [30]:

```
# using the SQLite Table to read data.
con = sqlite3.connect('database.sqlite')
#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
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):
```

```
def partition(x):
    if x < 3:
        return 0
    return 1

#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
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
Number of data points in our data (525814, 10)
```

Out[30]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Ti
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	1	1303862
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	0	1346976
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	1	1219017

In [31]:

```
#Sorting data according to Time in ascending order
sorted_data=filtered_data.sort_values('Time', axis=0, ascending=True, inplace=False, kind='quicksort',
na_position='last')
#sorted_data.head
```

In [32]:

```
sorted_data['Score'].value_counts()
```

Out[32]:

```
1    443777
0     82037
Name: Score, dtype: int64
```

In [33]:

```
#Selecting top 60k points
final_data = sorted_data[0:60000:]
y = final_data['Score']
#final_data.head
```

In [34]:

```
# find sentences containing HTML tags
import re
i=0;
for sent in final_data['Text'].values:
```

```
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;
```

```
14
What happens when you say his name three times? Michael Keaten stars in this comedy about two couples t
hat live in an old two story house.  While coming back from a supply store, the couple suddenly get cau
ght inside of a  &quot;broken-up&quot; bridge and then just before they start to tumble down  into the
lake, a board catches them.  But just when they've got their hopes  up, and small dog steps on the boar
d and the car starts to slide off the  bridge and into the lake waters.  A few minutes later...<p>They
find  themselves back into their home, they find that somehow somehad light the  fireplace, as if done
by magic.  From then on, they find a weird-looking  dead guy known as Bettlejuice.  The only way they c
an get him for help is  to call him by his name three times and he will appear at their survice.  But t
hey soon wish that they have never called his name, because  Bettlejuice was once a troublemaker but he
is the only one who can save  them, on the account that they said his name three times.  They can't lea
ve  their houses or else they will find theirselves in another world with giant  sandworms.  This is a
stellar comedy that you should see! Michael Keaton is  awesome as he plays the leading role of Bettleju
ice.
```

In [35]:

```
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
print(stop)
print('************************************')
print(sno.stem('beautiful'))
```

```
{"don't", "weren't", 'more', 'why', 't', 'have', 'and', 'me', 'each', 'during', 'himself', 'mightn', "h
adn't", 'of', 'how', 'which', "you're", 'll', 'am', 'wouldn', 'doing', "that'll", 'while', 'so', 'that'
, 'then', "should've", 'does', 'ours', 'this', 'had', 'both', 'him', "shan't", 'all', 'yours', 'few', '
them', 'too', 're', 'being', 'your', 'did', 'for', 'the', 'when', 'those', 'hadn', "it's", 'haven', 'hi
s', 'a', 'from', "you've", "you'd", 'down', 'now', "hasn't", 'once', 'off', 'no', "couldn't", 'shouldn'
, 'my', 'm', 'any', 'not', 'were', 'do', 'until', 'there', 'ain', 'aren', 'other', "shouldn't", "you'll
", 'yourself', 'hasn', 'who', 'will', 'with', 'o', 'under', 'some', 'here', 's', 'on', 'just', "haven't
", 'as', 'whom', 'these', 'if', "needn't", 'above', 'itself', 'was', 'about', 'its', "aren't", 'at', 'b
een', 'wasn', 'an', 'needn', 'he', 'you', 'couldn', 'or', 'don', 'again', 'weren', 'because', 'she', 'f
urther', 'over', 'by', 'ma', 'before', 'such', 'they', 'i', 'than', 'what', 'didn', 'own', "isn't", "wo
uldn't", 'against', 'to', 'their', 'won', "she's", 'be', 'theirs', 'her', 'hers', 'up', "didn't", 'most
', "won't", 'but', 'into', 'themselves', 'very', 'has', 've', 'isn', 'herself', "wasn't", 'd', 'through
', 'yourselves', 'mustn', 'ourselves', 'is', 'same', 'can', 'it', 'shan', 'doesn', 'are', "doesn't", 'b
elow', 'myself', 'nor', 'in', 'between', 'we', 'having', 'after', 'out', 'should', 'our', 'only', 'wher
e', "mightn't", 'y', "mustn't"}
************************************
beauti
```

In [36]:

```
if os.path.isfile('final.sqlite'):
    i=0
    str1=' '
    final_string=[]
    all_positive_words=[] # store words from +ve reviews here
    all_negative_words=[] # store words from -ve reviews here.
    s=''
    #print(final['Text'].head)
    #print(final['Text'].shape)
    for sent in tqdm(final_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():
```

```
            for cleaned_words in cleanpunc(w).split():
                if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                    #print("First If condition Passed")
                    if(cleaned_words.lower() not in stop):
                        #print("Word is not a stopword")
                        s=(sno.stem(cleaned_words.lower())).encode('utf8')
                        #print(s)
                        filtered_sentence.append(s)
                        # (final['Score'].values)[i] == 'positive':
                        if (final_data['Score'].values)[i] == 1:
                            #print("Positive word found")
                            all_positive_words.append(s) #list of all words used to describe positive r
eviews
                        #if(final['Score'].values)[i] == 'negative':
                        if(final_data['Score'].values)[i] == 0:
                            #print("Negative word found")
                            all_negative_words.append(s) #list of all words used to describe negative r
eviews reviews
                    else:
                        continue
                else:
                    continue
        #print(filtered_sentence)
        str1 = b" ".join(filtered_sentence) #final string of cleaned words
        #str1
        #print("***********************************************************************")

        final_string.append(str1)
        i+=1

    #############---- storing the data into .sqlite file ------########################
    final_data['CleanedText']=final_string #adding a column of CleanedText which displays the data afte
r pre-processing of the review
    final_data['CleanedText']=final_data['CleanedText'].str.decode("utf-8")
    #print(final['CleanedText'])
    #print(final.shape)
    print(final_data.columns.values)


        # store final table into an SQlLite table for future.
    conn = sqlite3.connect('final.sqlite')
    c=conn.cursor()
    conn.text_factory = str
    sorted_data.to_sql('Reviews', conn,  schema=None, if_exists='replace', \
                 index=True, index_label=None, chunksize=None, dtype=None)
    conn.close()



    with open('positive_words.pkl', 'wb') as f:
        pickle.dump(all_positive_words, f)
    with open('negitive_words.pkl', 'wb') as f:
        pickle.dump(all_negative_words, f)

#print(all_positive_words)
#print(all_negative_words)
```

```
100%|██████████████████████████████| 60000/60000 [01:36<00:00, 623.67it/s]
```

```
['Id' 'ProductId' 'UserId' 'ProfileName' 'HelpfulnessNumerator'
 'HelpfulnessDenominator' 'Score' 'Time' 'Summary' 'Text' 'CleanedText']
```

In [37]:

```
#Splitting into train and test (80:20)
X_train, X_test, y_train, y_test = train_test_split(final_data, y, test_size=0.2)
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

#import pickle as pkl

#to save it
#with open("train.pkl", "w") as f:
#    pkl.dump([X_train], f)
```

```
#Splitting train data into train and cv(60:20)
X_tr, X_cv, y_tr, y_cv = train_test_split(X_train, y_train, test_size=0.2)
print(X_tr.shape, y_tr.shape)
print(X_cv.shape, y_cv.shape)
```

```
(48000, 11) (48000,)
(12000, 11) (12000,)
(38400, 11) (38400,)
(9600, 11) (9600,)
```

In [38]:

```
#Applying BoW
#X_tr['CleanedText'].head
model = CountVectorizer()
model.fit(X_tr['CleanedText'])
train_bow = model.transform(X_tr['CleanedText'])
cv_bow = model.transform(X_cv['CleanedText'])
test_bow = model.transform(X_test['CleanedText'])
print(test_bow.shape)
print(cv_bow.shape)
print(train_bow.shape)
```

```
(12000, 21093)
(9600, 21093)
(38400, 21093)
```

Applying Truncated SVD for train dataset by choosing 300/500/800 features and calculating explained variance ratio. Fixing 800 features since it has high variance 84%

In [26]:

```
#Applying Truncated SVD:
#choosing 300 features and calculating explained variance ration sum
from sklearn.decomposition import TruncatedSVD
from sklearn.random_projection import sparse_random_matrix
svd = TruncatedSVD(n_components=300)
svd.fit(train_bow)
TruncatedSVD(algorithm='randomized', n_components=300, n_iter=7,
        random_state=42, tol=0.0)
train_bow_svd = svd.fit(train_bow).transform(train_bow)
print(train_bow_svd.shape)
print(svd.explained_variance_ratio_.sum())
```

```
(64000, 300)
0.6564238793755369
```

In [27]:

```
##Applying Truncated SVD:
#choosing 500 features and calculating explained variance ration sum
from sklearn.decomposition import TruncatedSVD
from sklearn.random_projection import sparse_random_matrix
svd = TruncatedSVD(n_components=500)
svd.fit(train_bow)
TruncatedSVD(algorithm='randomized', n_components=500, n_iter=7,
        random_state=42, tol=0.0)
train_bow_svd = svd.fit(train_bow).transform(train_bow)
print(train_bow_svd.shape)
print(svd.explained_variance_ratio_.sum())
```

```
(64000, 500)
0.7516773877215094
```

In [39]:

```
##Applying Truncated SVD:
#choosing 800 features and calculating explained variance ration sum
```

```
from sklearn.decomposition import TruncatedSVD
from sklearn.random_projection import sparse_random_matrix
svd = TruncatedSVD(n_components=800)
svd.fit(train_bow)
TruncatedSVD(algorithm='randomized', n_components=800, n_iter=7,
        random_state=42, tol=0.0)
train_bow_svd = svd.fit(train_bow).transform(train_bow)
print(train_bow_svd.shape)
print(svd.explained_variance_ratio_.sum())
```

```
(38400, 800)
0.8403366301260714
```

In [ ]:

```
Applying Truncated SVD for test and cross validation dataset with 800 features
```

In [40]:

```
##Applying Truncated SVD:
#choosing 800 features and calculating explained variance ration sum
from sklearn.decomposition import TruncatedSVD
from sklearn.random_projection import sparse_random_matrix
svd = TruncatedSVD(n_components=800)
svd.fit(test_bow)
TruncatedSVD(algorithm='randomized', n_components=800, n_iter=7,
        random_state=42, tol=0.0)
test_bow_svd = svd.fit(test_bow).transform(test_bow)
print(test_bow_svd.shape)
print(svd.explained_variance_ratio_.sum())
```

```
(12000, 800)
0.8683241165174521
```

In [41]:

```
##Applying Truncated SVD:
#choosing 800 features and calculating explained variance ration sum
from sklearn.decomposition import TruncatedSVD
from sklearn.random_projection import sparse_random_matrix
svd = TruncatedSVD(n_components=800)
svd.fit(cv_bow)
TruncatedSVD(algorithm='randomized', n_components=800, n_iter=7,
        random_state=42, tol=0.0)
cv_bow_svd = svd.fit(cv_bow).transform(cv_bow)
print(cv_bow_svd.shape)
print(svd.explained_variance_ratio_.sum())
```

```
(9600, 800)
0.86971031672507
```

As the number of (features) increases(300/500/800), the explained variance ratio also increases. More the variance better the result is. Hence, choosing 800 features for train/test and CV dataset.

Explained variance ratio is approximately equal to 85% for train,test and CV dataset with 800 features.

In [11]:

```
#function to save sparse matrices to a file
from scipy.sparse import csr_matrix

def save_sparse_csr(filename, array):
    # note that .npz extension is added automatically
    np.savez(filename, data=array.data, indices=array.indices,
             indptr=array.indptr, shape=array.shape)
```

In [12]:

```
save_sparse_csr('Train_sparse',train_bow)
save_sparse_csr('Test_sparse',test_bow)
save_sparse_csr('CV_sparse',cv_bow)
```

In [42]:

```
def knnbrute(Train,CV,Test):


    for i in range(1,30,2):
    # instantiate learning model (k = 30)
        knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'brute')

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

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

    # evaluate CV accuracy
        acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
        print('\nCV accuracy for k = %d is %d%%' % (i, acc))

    knn = KNeighborsClassifier(1)
    knn.fit(Train,y_tr)
    pred = knn.predict(Test)
    acc = accuracy_score(y_test, pred, normalize=True) * float(100)
    print('\n****Test accuracy for k = 1 is %d%%' % (acc))

    print('\n***********************Metrics calculation********************************')
    Confusion_mat = confusion_matrix(y_test,pred)
    class_label = ['0', '1']
    con_mat = pd.DataFrame(Confusion_mat, index = class_label, columns = class_label)
    sns.heatmap(con_mat, annot = True, fmt = "d")
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
    print("Confusion_mat:\n",Confusion_mat)
    #F1 Score
    print("F1 score:\n",metrics.f1_score(y_test,pred,labels=None, pos_label=1, average='binary', sample
_weight=None))
    #AUC score
    print("ROC AUC score:\n",metrics.roc_auc_score(y_test, pred, sample_weight=None))
    #Classification Report
    print("Classification Report:\n",metrics.classification_report(y_test, pred, labels=None, target_na
mes=None))
```

In [43]:

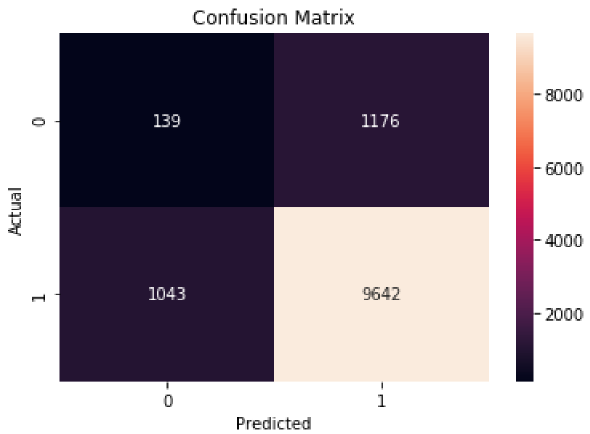
```
def knnkd_tree(Train,CV,Test):


    for i in range(1,30,2):
    # instantiate learning model (k = 30)
        knn = KNeighborsClassifier(n_neighbors=i,algorithm = 'kd_tree')

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

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

    # evaluate CV accuracy
        acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
        print('\nCV accuracy for k = %d is %d%%' % (i, acc))

    knn = KNeighborsClassifier(1)
    knn.fit(Train,y_tr)
    pred = knn.predict(Test)
    acc = accuracy_score(y_test, pred, normalize=True) * float(100)
    print('\n****Test accuracy for k = 1 is %d%%' % (acc))

    print('\n***********************Metrics calculation********************************')
```

```
    Confusion_mat = confusion_matrix(y_test,pred)
    class_label = ['0', '1']
    con_mat = pd.DataFrame(Confusion_mat, index = class_label, columns = class_label)
    sns.heatmap(con_mat, annot = True, fmt = "d")
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
    print("Confusion_mat:\n",Confusion_mat)
    #F1 Score
    print("F1 score:\n",metrics.f1_score(y_test,pred,labels=None, pos_label=1, average='binary', sample
_weight=None))
    #AUC score
    print("ROC AUC score:\n",metrics.roc_auc_score(y_test, pred, sample_weight=None))
    #Classification Report
    print("Classification Report:\n",metrics.classification_report(y_test, pred, labels=None, target_na
mes=None))
```

In [28]:

```
knnbrute(train_bow_svd,cv_bow_svd,test_bow_svd)
```

```
CV accuracy for k = 1 is 80%

CV accuracy for k = 3 is 79%

CV accuracy for k = 5 is 81%

CV accuracy for k = 7 is 80%

CV accuracy for k = 9 is 78%

CV accuracy for k = 11 is 76%

CV accuracy for k = 13 is 80%

CV accuracy for k = 15 is 82%

CV accuracy for k = 17 is 84%

CV accuracy for k = 19 is 86%

CV accuracy for k = 21 is 87%

CV accuracy for k = 23 is 87%

CV accuracy for k = 25 is 87%

CV accuracy for k = 27 is 87%

CV accuracy for k = 29 is 87%

****Test accuracy for k = 1 is 81%

***********************Metrics calculation*******************************
```




```
Confusion_mat:
```

```
 [[ 139 1176]
 [1043 9642]]
F1 score:
 0.8968050969632144
ROC AUC score:
 0.5040449726082724
Classification Report:
             precision    recall  f1-score   support

          0       0.12      0.11      0.11      1315
          1       0.89      0.90      0.90     10685

avg / total       0.81      0.82      0.81     12000
```

Observation: KNN with BoW(brute algorithm)

1. Of the 12000 test data points, there are 1315 negative reviews and 10685 positive reviews.
2. From the confusion matrix, it can be found that out of 1315 negative reviews, 139 reviews are predicted as negative(True negative) and the remaining 1176 reviews are predicted as positive(False Positive).
3. Similarly, out of 10685 positive reviews, 9642 reviews are predicted correctly as positive(True Positive) and the rest 1043 are classified as negative reviews.
4. It can be said that KNN with BoW predicted positive reviews(class 1) better than negative reviews(class 0).
5. ROC metric is 0.5 which is better. ROC value lies between 0 and 1. If it is 0.5, the model is better.
6. CV Accuracy is 87% when the number of neighbours is 17. Test accuracy is 81% for k=1.

In [ ]:

```
knnkd_tree(train_bow_svd,cv_bow_svd,test_bow_svd)
```

In [12]:

```
# Word2Vec model for train
i=0
list_of_sent=[]
for sent in X_tr['CleanedText'].values:
    list_of_sent.append(sent.split())

print(X_tr['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent[0])


# Word2Vec model for test and CV
i=0
list_of_sent_cv=[]
for sent in X_cv['CleanedText'].values:
    list_of_sent_cv.append(sent.split())

print(X_cv['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent_cv[0])


i=0
list_of_sent_test=[]
for sent in X_test['CleanedText'].values:
    list_of_sent_test.append(sent.split())

print(X_test['CleanedText'].values[0])
print("*****************************************************************")
print(list_of_sent_test[0])
```

```
introduc cavend greek season sever year ago greek festiv versatil use blacken steak grill pork chop sea
son gravi season hamburg potato give gift friend love much
*****************************************************************
['introduc', 'cavend', 'greek', 'season', 'sever', 'year', 'ago', 'greek', 'festiv', 'versatil', 'use',
'blacken', 'steak', 'grill', 'pork', 'chop', 'season', 'gravi', 'season', 'hamburg', 'potato', 'give',
'gift', 'friend', 'love', 'much']
brother bought last christma best brat ever eaten need search elsewher order pounder order thanksgiv th
ank bro
*****************************************************************
```

```
['brother', 'bought', 'last', 'christma', 'best', 'brat', 'ever', 'eaten', 'need', 'search', 'elsewher'
, 'order', 'pounder', 'order', 'thanksgiv', 'thank', 'bro']
tri tasimo coffe maker love select coffe creamer great servic receiv amazon fast stay date deliveri tha
nk amazon
*****************************************************************
['tri', 'tasimo', 'coffe', 'maker', 'love', 'select', 'coffe', 'creamer', 'great', 'servic', 'receiv',
'amazon', 'fast', 'stay', 'date', 'deliveri', 'thank', 'amazon']
```

In [13]:

```
w2v_model_train=Word2Vec(list_of_sent,min_count=5,size=50, workers=5)
w2v_model_test=Word2Vec(list_of_sent_test,min_count=5,size=50, workers=5)
w2v_model_cv=Word2Vec(list_of_sent_cv,min_count=5,size=50, workers=5)
```

In [14]:

```
w2v_words = list(w2v_model_train.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  8416
sample words  ['introduc', 'cavend', 'greek', 'season', 'sever', 'year', 'ago', 'festiv', 'versatil', '
use', 'blacken', 'steak', 'grill', 'pork', 'chop', 'gravi', 'hamburg', 'potato', 'give', 'gift', 'frien
d', 'love', 'much', 'order', 'lean', 'pupperoni', 'case', 'amazon', 'tina', 'treat', 'break', 'one', 'f
ood', 'morn', 'make', 'palat', 'put', 'medicin', 'deter', 'eat', 'alway', 'fix', 'finiki', 'type', 'fav
orit', 'weve', 'even', 'modifi', 'song', 'famili']
```

In [15]:

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # 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_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))


# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent_test): # for each review/sentence
    sent_vec_test = 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_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))


# average Word2Vec
# compute average word2vec for each review.
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sent_cv): # for each review/sentence
    sent_vec_cv = 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_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv.append(sent_vec)
print(len(sent_vectors_cv))
print(len(sent_vectors_cv[0]))
```

```
100%|██████████| 38400/38400 [00:39<00:00, 961.16it/s]
```

```
38400
50
```

```
100%|██████████| 12000/12000 [00:12<00:00, 976.18it/s]
```

```
12000
50
```

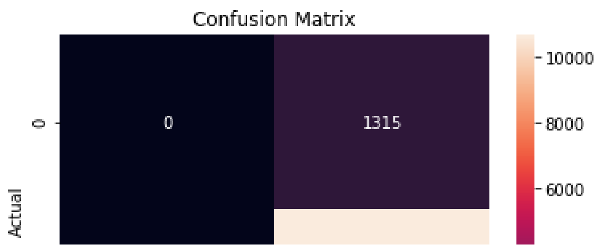
```
100%|██████████| 9600/9600 [00:09<00:00, 972.17it/s]
```

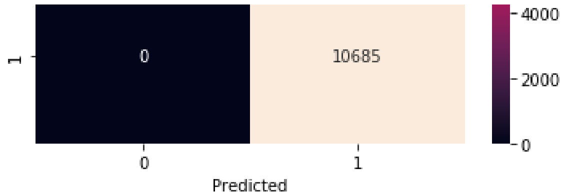
```
9600
50
```

In [16]:

```
knnbrute(sent_vectors,sent_vectors_cv,sent_vectors_test)
```

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

CV accuracy for k = 3 is 10%

CV accuracy for k = 5 is 10%

CV accuracy for k = 7 is 10%

CV accuracy for k = 9 is 10%

CV accuracy for k = 11 is 10%

CV accuracy for k = 13 is 10%

CV accuracy for k = 15 is 10%

CV accuracy for k = 17 is 10%

CV accuracy for k = 19 is 10%

CV accuracy for k = 21 is 10%

CV accuracy for k = 23 is 10%

CV accuracy for k = 25 is 10%

CV accuracy for k = 27 is 10%

CV accuracy for k = 29 is 10%

****Test accuracy for k = 1 is 89%

***********************Metrics calculation*******************************
```






```
Confusion_mat:
 [[    0  1315]
 [    0 10685]]
F1 score:
 0.9420321798545294
ROC AUC score:
 0.5
```

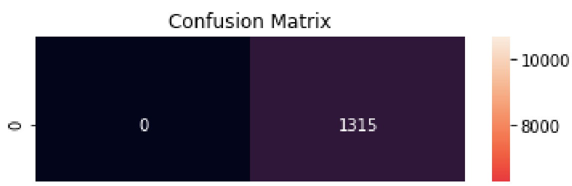
```
D:\AAnaconda\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

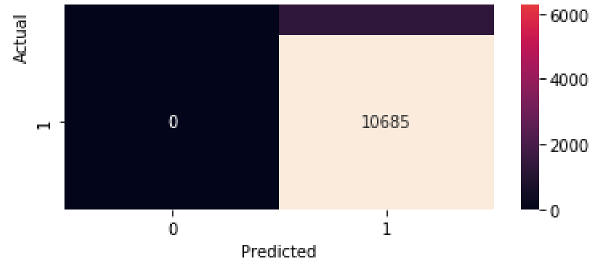
```
Classification Report:
              precision    recall  f1-score   support

          0       0.00      0.00      0.00      1315
          1       0.89      1.00      0.94     10685

avg / total       0.79      0.89      0.84     12000
```

In [18]:

```
knnkd_tree(sent_vectors,sent_vectors_cv,sent_vectors_test)
```

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

CV accuracy for k = 3 is 10%

CV accuracy for k = 5 is 10%

CV accuracy for k = 7 is 10%

CV accuracy for k = 9 is 10%

CV accuracy for k = 11 is 10%

CV accuracy for k = 13 is 10%

CV accuracy for k = 15 is 10%

CV accuracy for k = 17 is 10%

CV accuracy for k = 19 is 10%

CV accuracy for k = 21 is 10%

CV accuracy for k = 23 is 10%

CV accuracy for k = 25 is 10%

CV accuracy for k = 27 is 10%

CV accuracy for k = 29 is 10%

****Test accuracy for k = 1 is 89%

***********************Metrics calculation*******************************
```






```
Confusion_mat:
 [[    0  1315]
 [    0 10685]]
F1 score:
 0.9420321798545294
ROC AUC score:
 0.5
```

```
D:\AAnaconda\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: Precisio
n and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
Classification Report:
              precision    recall  f1-score   support

          0       0.00      0.00      0.00      1315
          1       0.89      1.00      0.94     10685

avg / total       0.79      0.89      0.84     12000
```

In [ ]:

```
Observation: KNN with Average weighted W2V

1. Of the 12000 test data points, there are 1315 negative reviews and 10685 positive reviews.
2. From the confusion matrix, it can be found that all the negative reviews are predicted as positive(F
alse Positive).
3. Similarly, out of 10685 positive reviews, all of them are predicted correctly as positive(True Posit
ive).
4. It can be said that this model predicted positive reviews(class 1) accurately but predicted the nega
tive reviews wrongly.
5. Though the ROC metric and F1 score looks better,this model didnot perform well while predicting Nega
tive reviews.
6. CV Accuracy is 89% when the number of neighbours is 1. As the number of neighbours increases, the ac
curacy is getting reduced.
Since, the number of positive reviews is greater than the negative ones(10685 positive>1315 negative),
this model is biased
towards positive reviews.
Test accuracy is 89% for k=1.(model is overfitted)
```

In [20]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_tr['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [21]:

```
# TF-IDF weighted Word2Vec
tfidf_feat = model.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 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:
```

```
        if word in w2v_words:
            vec = w2v_model_train.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this 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



# TF-IDF weighted Word2Vec for test dataset
tfidf_feat = model.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_sent_test): # 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_train.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this 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


# TF-IDF weighted Word2Vec for cross validation dataset
tfidf_feat = model.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_sent_cv): # 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_train.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this 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
```

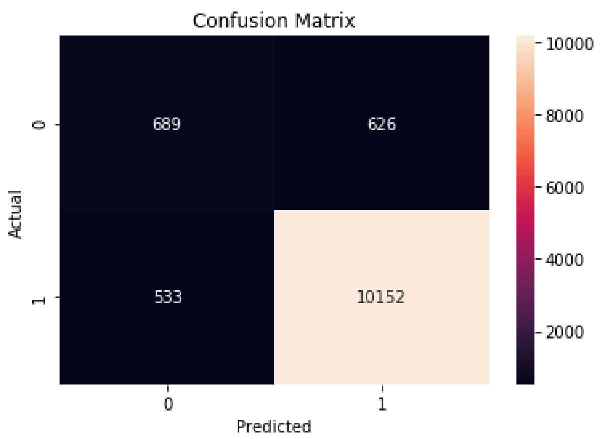
```
100%|██████████| 38400/38400 [00:52<00:00, 733.47it/s]
100%|██████████| 12000/12000 [00:16<00:00, 747.55it/s]
100%|██████████| 9600/9600 [00:12<00:00, 747.73it/s]
```

In [22]:

```
knnbrute(tfidf_sent_vectors,tfidf_sent_vectors_cv,tfidf_sent_vectors_test)
```

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

```
CV accuracy for k = 3 is 91%

CV accuracy for k = 5 is 91%

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 90%

CV accuracy for k = 19 is 90%

CV accuracy for k = 21 is 90%

CV accuracy for k = 23 is 90%

CV accuracy for k = 25 is 90%

CV accuracy for k = 27 is 90%

CV accuracy for k = 29 is 90%

****Test accuracy for k = 1 is 90%

***********************Metrics calculation*******************************
```



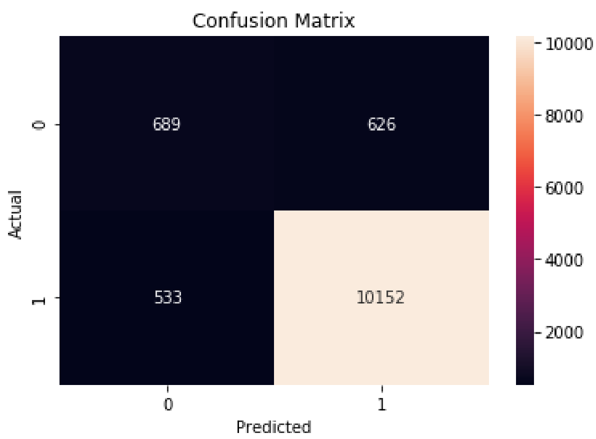

```
Confusion_mat:
 [[  689   626]
 [  533 10152]]
F1 score:
 0.9460000931836183
ROC AUC score:
 0.7370356795265741
Classification Report:
              precision    recall  f1-score   support

          0       0.56      0.52      0.54      1315
          1       0.94      0.95      0.95     10685

avg / total       0.90      0.90      0.90     12000
```

In [23]:

```
knnkd_tree(tfidf_sent_vectors,tfidf_sent_vectors_cv,tfidf_sent_vectors_test)
```

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

CV accuracy for k = 3 is 91%

CV accuracy for k = 5 is 91%
```

```
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 90%

CV accuracy for k = 19 is 90%

CV accuracy for k = 21 is 90%

CV accuracy for k = 23 is 90%

CV accuracy for k = 25 is 90%

CV accuracy for k = 27 is 90%

CV accuracy for k = 29 is 90%

****Test accuracy for k = 1 is 90%

************************Metrics calculation********************************
```




```
Confusion_mat:
 [[  689   626]
 [  533 10152]]
F1 score:
 0.9460000931836183
ROC AUC score:
 0.7370356795265741
Classification Report:
              precision    recall  f1-score   support

          0       0.56      0.52      0.54      1315
          1       0.94      0.95      0.95     10685

avg / total       0.90      0.90      0.90     12000
```

In [ ]:

```
Observation: KNN with TFIDF weighted W2V.

1. Of the 12000 test data points, there are 1315 negative reviews and 10685 positive reviews.
2. From the confusion matrix, it can be found that out of 1315 negative reviews, 689 are predicted corr
ectly as negative
reviews and the remaining 626 are predicted incorrectly as positive(False Positive).
3. Similarly, out of 10685 positive reviews, 10152 reviews are predicted correctly as positive(True Pos
itive) and the rest
533 predicted incorrectly as negative(False negative)
4. It can be said that this model predicted both positive and negative reviews to an acceptable extent.
```

```
5. F1 score is 0.54 for negative reviews and 0.95 for positive reviews.
6. CV Accuracy is getting stable (90%) when the number of neighbours is increasing from 7.
Test accuracy is 90% for k=1.
```

In [11]:

```
from prettytable import PrettyTable
table = PrettyTable(["model","k","Train accuracy","Test accuracy","F1 score","ROC"])
table.add_row(["KNN with BoW", "17", "87%","81%","0.89","0.5"])
table.add_row(["KNN with TFIDF", "7", "88%","82","0.90","0.49"])
table.add_row(["KNN with Avg w2v", "1", "89%","89","0.94","0.5"])
table.add_row(["KNN with TFIDF weighted w2v","7","90%","90%","0.94","0.73"])
print(table)
```

```
+-----------------------------+----+----------------+---------------+----------+------+
|            model            | k  | Train accuracy | Test accuracy | F1 score | ROC  |
+-----------------------------+----+----------------+---------------+----------+------+
|         KNN with BoW        | 17 |      87%       |      81%      |   0.89   | 0.5  |
|        KNN with TFIDF       | 7  |      88%       |       82      |   0.90   | 0.49 |
|       KNN with Avg w2v      | 1  |      89%       |       89      |   0.94   | 0.5  |
| KNN with TFIDF weighted w2v | 7  |      90%       |      90%      |   0.94   | 0.73 |
+-----------------------------+----+----------------+---------------+----------+------+
```

Conclusion:

```
  Test accuracy is close to 80% for all the three vectorization techniques. Both kd_tree and bru
te algorithm shows similar results in terms of accuracy.
  F1 score is considered as a single metric for precision and recall. F1 score(>0.8) for all the
models looks good. That is all the model perform well on predicting positive reviews.
  Considering ROC AUC score and negative class prediction into account, TFIDF w2v performs well.
```

Performance of the model in descending order:

```
1.TFIDF weighted w2v
2.BoW
3.TFIDF
4.Average w2v
```

BoW: 1.KNN with BoW predicted positive reviews(class 1) better than negative reviews(class 0). 2.ROC metric is 0.5 which is better. ROC value lies between 0 and 1. If it is 0.5, the model is better. 3.CV Accuracy is 87% when the number of neighbours is 17. Test accuracy is 81% for k=1.

Average W2V:

```
1.Average w2v model cannot be considered since it is trying to overfit the datapoints(number of
neighbors is 1).
2.Though the ROC metric and F1 score looks better,this model didnot perform well while predictin
g Negative reviews.
3.CV Accuracy is 89% when the number of neighbours is 1. As the number of neighbours increases,
the accuracy is getting reduced. Since, the number of positive reviews is greater than the negat
ive ones(10685 positive>1315 negative), this model is biased towards positive reviews. Test accu
racy is 89% for k=1.(model is overfitted)
```

TFIDF w2V:

```
Upon considering ROC metric, TFIDF weighted w2v KNN model is giving good result. (ROC is 0.73)Nu
mber of nearest neighbors is also 7.
F1 score is 0.54 for negative reviews and 0.95 for positive reviews.
CV Accuracy is getting stable (90%) when the number of neighbours is increasing from 7.
```

TFIDF:

1. Of the 2000 test data points, there are 266 negative reviews and 1734 positive reviews.
2. Out of 1734 positive reviews, 1642 reviews are predicted correctly as positive and the rest are incorrectly predicted as negative. And out of 266 negative reviews, 14 are correctly predicted and the rest are incorrectly predicted.

3. ROC metric(0.49) and F1 score(0.90). Predicted positive class better.
4. CV Accuracy is 88% when the number of neighbours is 7.