Amazon Reviews Sentiment Prediction using KNN

We are given here amazon reviews dataset. We will first convert all reviews text to vector using different techniques like(bow, tfidf, average w2v, tfidf weighted average w2v). Our task here is to give Generalization score of our classifier (KNN using 'bruteforce' & 'kd_tree') on different text to vector converted data. We also need to get optimal k(nearest neigbors). After getting vector form of reviews we will fit these data in KNN classifier using two different algorithms 'bruteforce' and 'kd_tree'

In [ ]:
```
#importing necessary packages
import sqlite3
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import pickle
import gensim
import sklearn.cross_validation
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim')
```

In [2]:
```
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
```

Reading already Cleaned, Preprocessed data from database

After removing stopwords, punctuations, meaningless characters, HTML tags from Text and done stemming. Using it directly as it was alredy done in prevoius assignment

In [3]:
```
#Reading
conn= sqlite3.connect('cleanedTextData.sqlite')
data= pd.read_sql_query('''
SELECT * FROM Reviews
''',conn)
data=data.drop('index',axis=1)
data.shape
```

Out[3]:
```
(364171, 11)
```

In [4]:
```
data.columns
```

Out[4]:
```
Index(['Id', 'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerato
r',
       'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text',
       'CleanedText'],
      dtype='object')
```

In [5]:
```
data['CleanedText'].head(3)
```

Out[5]:
```
0    witti littl book make son laugh loud recit car...
1    rememb see show air televis year ago child sis...
2    beetlejuic well written movi everyth act speci...
Name: CleanedText, dtype: object
```

Sorting on the basis of 'Time' and taking top 60k pts

This data has time attribute so it will be reasonable to do time based splitting instead of random splitting.

So, before splitting we have to sort our data according to time and here we are taking 100k points from our dataset(population)

In [6]:
```
data["Time"] = pd.to_datetime(data["Time"], unit = "ms")
data = data.sort_values(by = "Time")
data.head(3)
```

Out[6]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0
1	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2
2	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0

In [7]:
```
#latest 60k points according to time
```

```
data= data[:60000]
len(data)
```

Out[7]: 60000

Splitting data into train60% cv20% test20%

Splitting our data into train, cross validation data and test data.

- train data will train our ML model
- cross validataion data will be for determining our hyperparameter
- test data will tell how Generalized our model is
- dataframes after splitting:- traindata, cvdata, testdata

In [8]:
```
tr, testdata= train_test_split(data, test_size= 0.2, shuffle= False,str
atify= None)
traindata, cvdata= train_test_split(tr, test_size= 0.25, shuffle=False,
stratify= None)
print(len(traindata),len(testdata),len(cvdata))
```

36000 12000 12000

Taking Text and score(class) as sequences

- traindata -> Xtrain, Ytrain
- cvdata -> Xcv, Ycv
- testdata -> Xtest, Ytest

In [9]:
```
Xtrain,Xcv,Xtest= traindata['CleanedText'],cvdata['CleanedText'],testda
ta['CleanedText']
Ytrain,Ycv,Ytest= traindata['Score'],cvdata['Score'],testdata['Score']
```

BOW Vectorization

Bow vectorization is basic technique to convert a text into numerical vector.

- We will build a model on train text using fit-transform
- Then transform (test and cv) text on model build by train text
- Transformed data will be in the form of sparse matrix

In [10]:
```
# vectorizing X and transforming
bowModel=CountVectorizer()
XtrainV=bowModel.fit_transform(Xtrain.values)
```

In [11]:
```
XcvV= bowModel.transform(Xcv)
XtestV= bowModel.transform(Xtest)
XtestV.shape
```

Out[11]: (12000, 27530)

Dumping sparse vectors for sustainable use

In [12]:
```
import pickle
with open('BowVectors.pkl','wb') as i:
    pickle.dump(XtrainV,i)
    pickle.dump(XcvV,i)
    pickle.dump(XtestV,i)
i.close()
```

Standardizing the vectors

In [ ]:
```
#Standardizing the vector
std = StandardScaler(with_mean=False).fit(XtrainV)
XtrainV = std.transform(XtrainV)
XtestV = std.transform(XtestV)
XcvV = std.transform(XcvV)
```

So our our Bow vectors are XtrainV, XtestV, XcvV

Converting Class of 'Positive','Negative' to 1 0 numerical represention

In [14]:
```
Ytrain=Ytrain.map(lambda x:1 if x=='Positive' else 0)
Ycv=Ycv.map(lambda x:1 if x=='Positive' else 0)
Ytest=Ytest.map(lambda x:1 if x=='Positive' else 0)
```

KNN - Bruteforce on BOW vector

We will apply KNN's bruteforce algorithm for classifying our dataset

- First we will make function to determine optimum 'k' using bruteforce
- this function will be used frequently.
- For determining 'k' we need to have a metric on which we will calculate k
- So to choose a metric first we have to check if our dataset is *balanced or imbalanced*

In [15]:
```
print(len(Ytrain[Ytrain==1])/len(Ytrain[Ytrain==0]))
```

```
8.125475285171103
```

So we can clearly see that 'Positive' points are 8 times larger in number than 'Negative' points. Hence this is *highly imbalanced dataset*

In this case simple acuracy can't be good metric as it wont work on imbalanced dataset. Fundamental metrics:-

- **TP** True positive points count **TN** True negative points count
- **FP** False positive points count **FN** False negative points count

For imbalanced datasets we can use advanced metrics like:-

- **Precision** TP/(TP+FP)
- **Recall** TP/(TP+FN)
- **F1 Score** - harmonic mean of precision and recall

- **ROC-AUC Score** - reciver operating characteristic
- **Confusion Matrix** ### Taking F1 Score as our Metric When there are more positive points are there in dataset then **F1 Score** metric is more preferable. Because it mantains good *balance* between precision and recall

Now we will make function **k_classifier_brute** which will return optimal k when passed in arguments are train and cross validate data. Optimal k will be calculated on **F1 Score** metric.

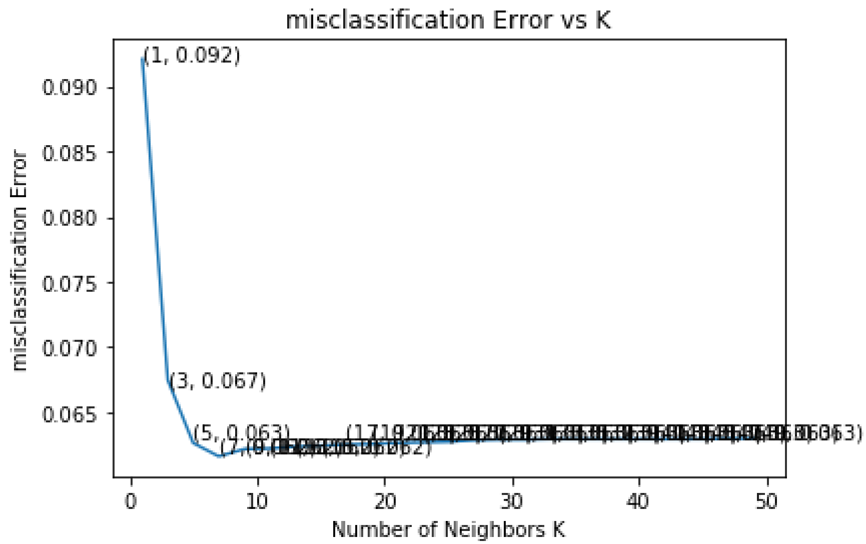
In [29]:
```
from sklearn.metrics import f1_score
def k_classifier_brute(xtrain, ytrain, xcv, ycv):
    '''
    This function will return optimal k along with misclassification er
ror vs
    K curve. k will be calculated on cv data
    '''
    # creating odd list of K for KNN
    myList = list(range(0,50))
    neighbors = list(filter(lambda x: x % 2 != 0, myList))

    f1_scores=[]
    # performing for different k
    for k in neighbors:
        knn = KNeighborsClassifier(n_neighbors=k, algorithm = "brute")
        knn.fit(xtrain,ytrain)
        pred= knn.predict(xcv)
        acc = f1_score(ycv, pred)
        f1_scores.append(acc)

    # changing to misclassification error
    MSE = [1 - x for x in f1_scores]

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

    # plot misclassification error vs k
    plt.plot(neighbors, MSE)
```

```
    for xy in zip(neighbors, np.round(MSE,3)):
        plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
    plt.title("misclassification Error vs K")
    plt.xlabel('Number of Neighbors K')
    plt.ylabel('misclassification Error')
    plt.show()
    print("the misclassification error for each k value is : ", np.roun
d(MSE,3))
    print('With f1_score as ',max(f1_scores))
    return optimal_k
```

Finding Optimal 'k' using CV data

In [30]:
```
# Calling function for optimum k and assigning it to k
k=k_classifier_brute(XtrainV,Ytrain,XcvV,Ycv)
```

```
The optimal number of neighbors is 7.
```




```
the misclassification error for each k value is :  [0.092 0.067 0.063
0.062 0.062 0.062 0.062 0.062 0.063 0.063 0.063 0.063
 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.06
3
```

```
 0.063]
With f1_score as  0.9383780899875511
```

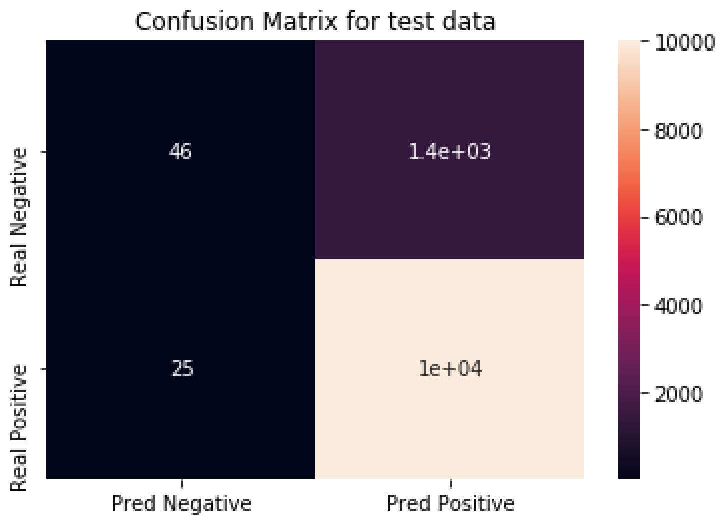
Using this optimal 'k' to train our model.

using brute algorithm. predicting out test datapoints.

In [31]:
```python
knn= KNeighborsClassifier(n_neighbors=k,algorithm='brute')
knn.fit(XtrainV,Ytrain)
pred= knn.predict(XtestV)
```

In [3]:
```python
from sklearn.metrics import confusion_matrix
cfm= confusion_matrix(Ytest,pred)
df_cm = pd.DataFrame(cfm,columns = ['Pred Negative','Pred Positive']\
                     ,index = ['Real Negative','Real Positive'])
plt.title('Confusion Matrix for test data')
sns.heatmap(df_cm, annot=True)
print('f1_score of Test data is ',f1_score(Ytest,pred))
```

```
f1_score of Test data is  0.9347235218108095
```



KNN kd-tree on BOW vector

Kd-tree becomes very time expensive when the dimensionality of data increases. So before feeding our data in kd-tree we will:-

- Reduce the dimensionality of data, using TruncatedSVD
- Standardize our data

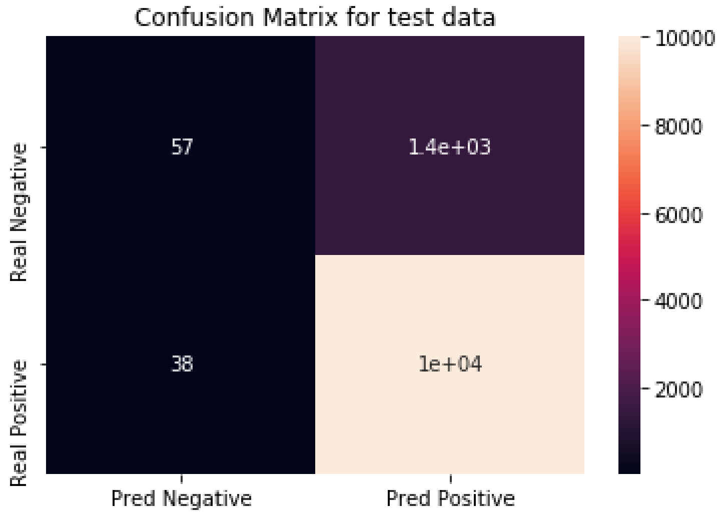
In [34]:
```
from sklearn.decomposition import TruncatedSVD
model=TruncatedSVD(n_components=1500)
XtrainVd=model.fit_transform(XtrainV)
XtestVd= model.transform(XtestV)
XcvVd= model.transform(XcvV)
```

In [35]:
```
from sklearn.preprocessing import StandardScaler
std=StandardScaler(with_mean=False)
XtrainVd = std.fit_transform(XtrainVd)
XtestVd = std.transform(XtestVd)
XcvVd = std.transform(XcvVd)
```

Feeding data in knn kdtree model

In [36]:
```
# training and running kdtree model
knn = KNeighborsClassifier(n_neighbors=k, algorithm = "kd_tree")
knn.fit(XtrainVd,Ytrain)
pred= knn.predict(XtestVd)
```

In [4]:
```
cfm= confusion_matrix(Ytest,pred)
df_cm = pd.DataFrame(cfm,columns = ['Pred Negative','Pred Positive']\
                     ,index = ['Real Negative','Real Positive'])
plt.title('Confusion Matrix for test data')
sns.heatmap(df_cm, annot=True)
print('f1_score of Test data is ',f1_score(Ytest,pred))
```




```
f1_score of Test data is  0.9345644319550381
```

Observation:-

1. Opimal K is 7
2. CV score(F1_score) is 93.83 with CV error of 6.17
3. Testdata's F1_score of bruteforce is 93.47
4. Testdata's F1_score of kd_tree is 93.45
5. From confusion matrix we can say that around 1.4k points from 12k points of test data were wrong classified which implies accuracy of around 88.33

TFIDF vectorization

- We will build a model on train text using fit-transform
- Then transform (test and cv) text on model build by train text
- Transformed data will be in the form of sparse matrix
- Then Standardize our data

In [38]:
```
# generating vetor out of text using tfidf
model=TfidfVectorizer(max_features=2000,min_df=50)
XtrainV= model.fit_transform(Xtrain)
XtestV= model.transform(Xtest)
XcvV= model.transform(Xcv)
```

In [39]:
```
std= StandardScaler(with_mean=False)
XtrainV = std.fit_transform(XtrainV)
XtestV = std.transform(XtestV)
XcvV = std.transform(XcvV)
```

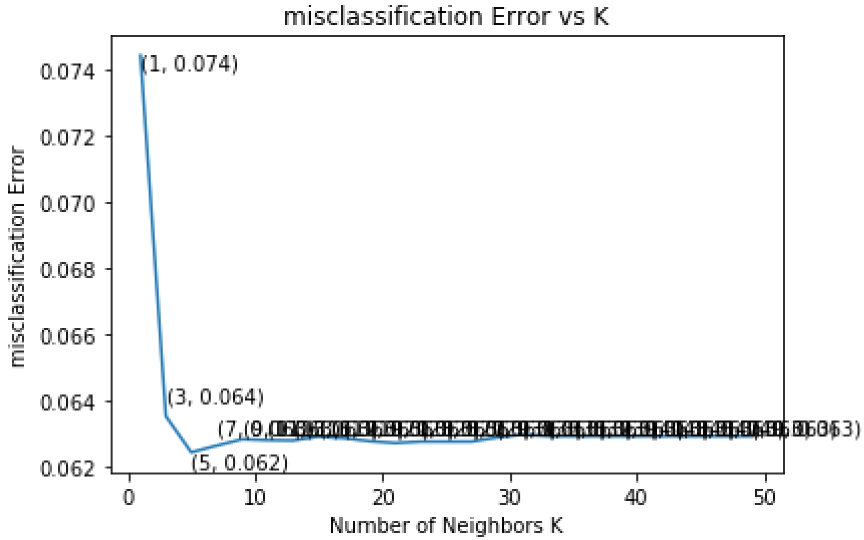
Saving data for sustainable use

In [40]:
```
import pickle
with open('TFIDFVectors.pkl','wb') as i:
    pickle.dump(XtrainV,i)
    pickle.dump(XcvV,i)
    pickle.dump(XtestV,i)
i.close()
```

Feeding data in k_classifier_brute() to get optimal 'k'

In [41]:
```
k= k_classifier_brute(XtrainV,Ytrain,XcvV,Ycv)
```

```
The optimal number of neighbors is 5.
```




```
the misclassification error for each k value is :  [0.074 0.064 0.062
0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063
 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.063 0.06
3
 0.063]
With f1_score as  0.9375499689082349
```

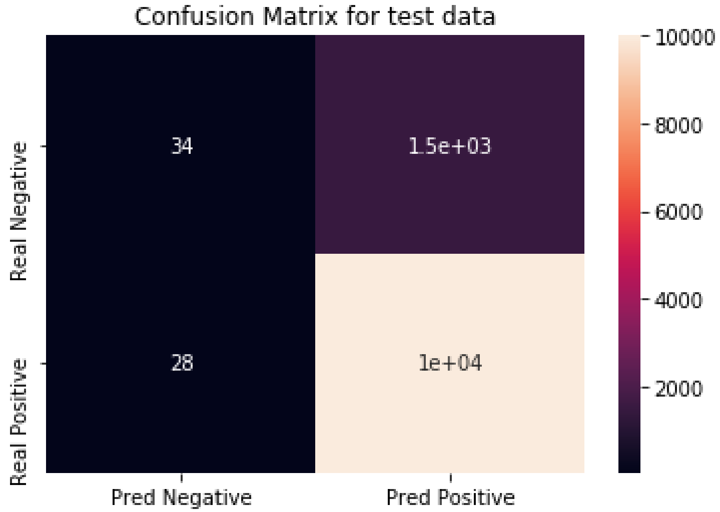
Knn bruteforce on Tfidf vect

Training knn bruteborce on optimal 'k'

In [42]:
```
knn= KNeighborsClassifier(n_neighbors=k,algorithm='brute')
knn.fit(XtrainV,Ytrain)
pred= knn.predict(XtestV)
```

In [5]:
```
cfm= confusion_matrix(Ytest,pred)
df_cm = pd.DataFrame(cfm,columns = ['Pred Negative','Pred Positive']\
                     ,index = ['Real Negative','Real Positive'])
plt.title('Confusion Matrix for test data')
```

```
sns.heatmap(df_cm, annot=True)
print('f1_score of Test data is ',f1_score(Ytest,pred))
```




```
f1_score of Test data is  0.9340815962943168
```

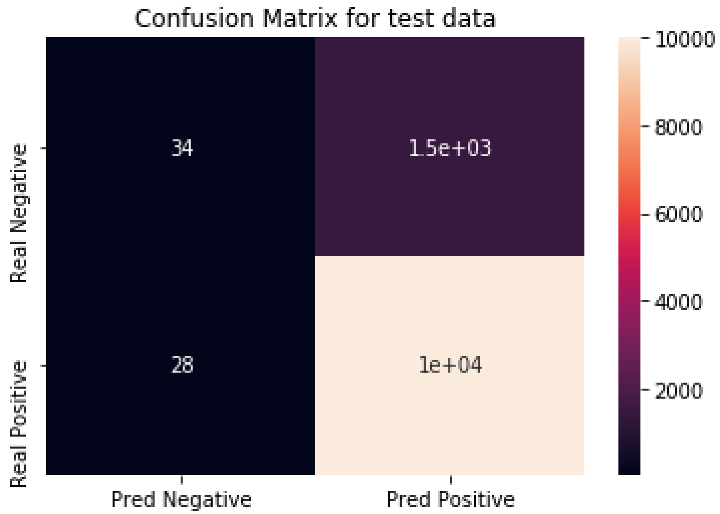
KNN Kdtree on TFIDF vector

In [44]:
```
# converting to dense matrix
XtrainVd= XtrainV.todense()
XtestVd= XtestV.todense()
XcvVd= XcvV.todense()
```

In [45]:
```
# training using kd_tree
knn = KNeighborsClassifier(n_neighbors=k, algorithm = "kd_tree")
knn.fit(XtrainVd,Ytrain)
pred= knn.predict(XtestVd)
```

In [6]:
```
cfm= confusion_matrix(Ytest,pred)
df_cm = pd.DataFrame(cfm,columns = ['Pred Negative','Pred Positive']\
                     ,index = ['Real Negative','Real Positive'])
plt.title('Confusion Matrix for test data')
```

```
sns.heatmap(df_cm, annot=True)
print('f1_score of Test data is ',f1_score(Ytest,pred))
```




```
f1_score of Test data is  0.9340815962943168
```

Observation:-

1. Opimal K is 5
2. CV score(F1_score) is 93.75 with CV error of 6.25
3. Testdata's F1_score of bruteforce is 93.40
4. Testdata's F1_score of kd_tree is 93.40
5. From confusion matrix we can say that around 1.5k points from 12k points of test data were wrong classified which implies accuracy of around 87.5

---

w2v Vectorization

- First we will Train gensim model on traindata text

- Build sentance vectors using average w2v and gensim
- **w2v_model** is our gensim model on train data
- **w2v_words** is our gensim word vocabulary on train data

In [47]:
```
# training our gensim model on our train text
import re
import string
def cleanhtml(sentance): #substitute expression contained in <> with ' '
    cleaned= re.sub(re.compile('<.*?>'),' ',sentance)
    return cleaned
#function for removing punctuations chars
def cleanpunc(sentance):
    cleaned= re.sub(r'[?|!|\'|"|#]',r'',sentance)
    cleaned= re.sub(r'[.|,|)|(|\|/]',r'',sentance)
    return cleaned
i=0
lists=[]

for sent in traindata['CleanedText'].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
    lists.append(filtered_sentence)


w2v_model= gensim.models.Word2Vec(lists,min_count=5,size=50,workers=4)
print(len(list(w2v_model.wv.vocab)))
```

```
7741
```

In [48]:
```
w2v_words = list(w2v_model.wv.vocab)
```

This function(**w2vVect**) will recive list of sentances (reviews here) and convert that into vector using *average w2v* vectorization. Finally return list of numerical vectors corresponding to text sentances

- after this we will feed our train, text, cv data in w2vVect() and get corresponding w2v vectorized sentances
- vocabulary,w2v_model was trained on training data only

In [52]:
```
# converting list of sentance into list of list of words
# then to vector using avg w2v
# function to convert list of list of words to vect using avg w2v
def w2vVect(X):
    '''
    This function takes list of sentance as input (X) and convert it in
to
    list of list of words and then feed it into our gensim model to get
 vector
    and then take its average, finally returns sent_vectors(vector of s
entance)
    ************GENSIM MODEL WAS TRAINED ON TRAINDATA**************
    '''

    lists=[]
    for sent in X.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
        lists.append(filtered_sentence)

    sent_vectors = [];
    for sent in lists:
        sent_vec = np.zeros(50)
        cnt_words =0;
```

```
        for word in sent:
            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_vectors.append(sent_vec)
    return sent_vectors
```

In [53]:
```
# Vectorizing our data
XtrainV= w2vVect(Xtrain)
XtestV= w2vVect(Xtest)
XcvV= w2vVect(Xcv)
```

Saving our data for sustainable use

In [54]:
```
import pickle
with open('W2VVectors.pkl','wb') as i:
    pickle.dump(XtrainV,i)
    pickle.dump(XcvV,i)
    pickle.dump(XtestV,i)
i.close()
```

In [55]:
```
print(len(XtrainV),len(XtestV),len(XcvV))
print(XtrainV[0])
```

```
36000 12000 12000
[ 0.11496643  0.21575583 -0.10025281  0.40054248 -0.07506769 -0.2481011
5
 -0.01198531  0.24205157  0.25574244 -0.31292871 -0.41015361 -0.1335601
8
 -0.12828043  0.37273438 -0.69870188  0.21488369 -0.07629116 -0.2417644
4
 -0.40725511 -0.24324185 -0.45485452 -0.39509548  0.14335891  0.2837919
4
  0.06037044  0.28245135 -0.18741142 -0.1557784  -0.5022352   0.0521304
5
```

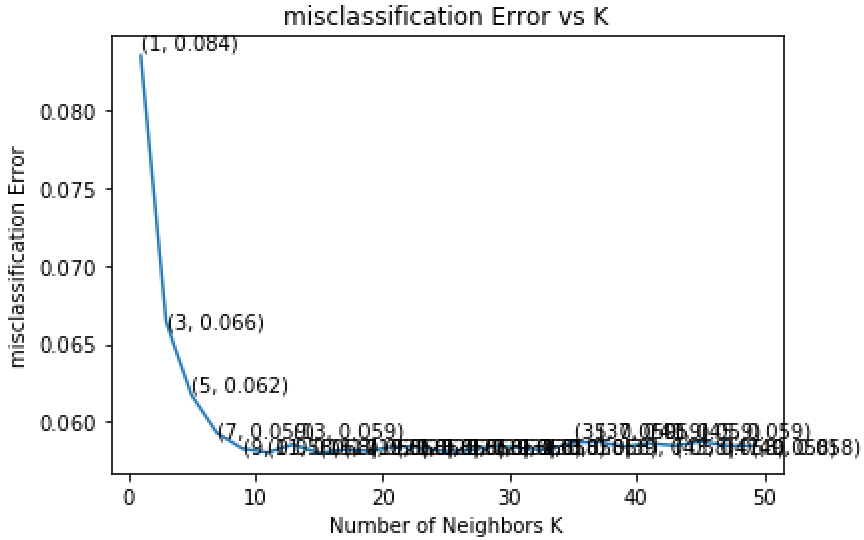
```
 -0.42250781  0.0188486   0.11458139  0.04087626 -0.20831408 -0.3120933
1
  0.41352587  0.19958171  0.18551075  0.50416246 -0.47969187  0.2560323
9
  0.2786916  -0.21661689  0.18634739  0.78883786 -0.00635821  0.4609007
6
 -0.08376825 -0.16584848]
```

KNN bruteforce on w2v

- First get optimal 'k'
- then apply knn bruteforce

In [56]:
```
k= k_classifier_brute(XtrainV,Ytrain,XcvV,Ycv)
```

```
The optimal number of neighbors is 15.
```

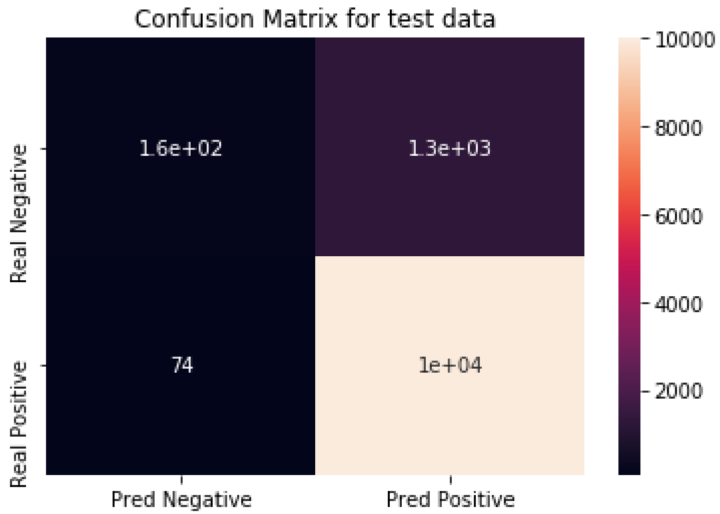



```
the misclassification error for each k value is :  [0.084 0.066 0.062
0.059 0.058 0.058 0.059 0.058 0.058 0.058 0.058 0.058
 0.058 0.058 0.058 0.058 0.058 0.059 0.059 0.058 0.059 0.058 0.059 0.05
8
```

```
 0.058]
With f1_score as  0.9420322392348794
```

In [57]:
```
knn= KNeighborsClassifier(n_neighbors=k,algorithm='brute')
knn.fit(XtrainV,Ytrain)
pred= knn.predict(XtestV)
```

In [7]:
```
cfm= confusion_matrix(Ytest,pred)
df_cm = pd.DataFrame(cfm,columns = ['Pred Negative','Pred Positive']\
                     ,index = ['Real Negative','Real Positive'])
plt.title('Confusion Matrix for test data')
sns.heatmap(df_cm, annot=True)
print('f1_score of Test data is ',f1_score(Ytest,pred))
```

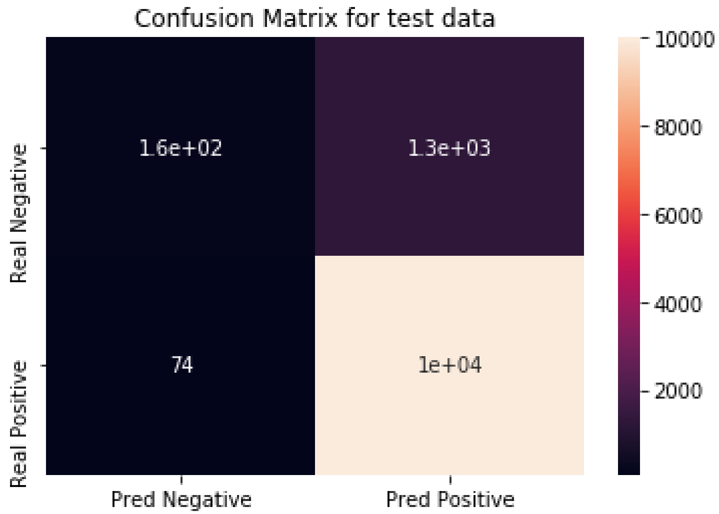



```
f1_score of Test data is  0.9369951534733442
```

KNN Kd-tree on w2v

In [59]:
```
knn= KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
knn.fit(XtrainV,Ytrain)
pred= knn.predict(XtestV)
```

In [8]:
```
cfm= confusion_matrix(Ytest,pred)
df_cm = pd.DataFrame(cfm,columns = ['Pred Negative','Pred Positive']\
                     ,index = ['Real Negative','Real Positive'])
plt.title('Confusion Matrix for test data')
sns.heatmap(df_cm, annot=True)
print('f1_score of Test data is ',f1_score(Ytest,pred))
```




```
f1_score of Test data is  0.9369951534733442
```

Observation:-

1. Opimal K is 15
2. CV score(F1_score) is 94.20 with CV error of 5.80
3. Testdata's F1_score of bruteforce is 93.69
4. Testdata's F1_score of kd_tree is 93.69
5. From confusion matrix we can say that around 1.3k points from 12k points of test data were wrong classified which implies accuracy of around 89.02

Tfidf-avg-w2v Vectorization

This is an advancement to w2v vectorization. In this we multiply the tfidf value and w2v value of each word in sentance and do averaging

- Train tfidf and w2v model on train data
- Make dictionary of vocabulary in which each word is a key and value is tfidf value of that word

In [111]:
```
model=TfidfVectorizer(max_features=2000)
tf_idf_matrix = model.fit_transform(Xtrain.values)
tfidf_feat=model.get_feature_names()
dictionary = {k:v for (k,v) in zip(model.get_feature_names(), list(mode
l.idf_))}
```

The function (**tfidfw2vVect**) will recive list of text sentance and convert it into list of vectors using TFIDF-avg-w2v vectorization process.

- after this we will feed our train, text, cv data in tfidfw2vVect() and get corresponding tfidfw2v vectorized sentances
- vocabulary,w2v_model,tfidf was trained on training data only

In [115]:
```
def tfidfw2vVect(X):
    '''
    This function converts list of sentance into list of list of words
 and then
    finally applies average-tfidf-w2w to get final sentance vector
    w2v model and w2v words already made during w2v vectorization part
    '''
    lists=[]
    for sent in X.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
        lists.append(filtered_sentence)

    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
    row=0;
    for sent in lists: # 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:
                if word in w2v_words:
                    vec = w2v_model.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
            except:
                pass
        if weight_sum != 0:
            sent_vec /= weight_sum
        tfidf_sent_vectors.append(sent_vec)
        row += 1
    # converting nan and infinte values in vect to digit
    tfidf_sent_vectors= np.nan_to_num(tfidf_sent_vectors)
    return tfidf_sent_vectors
```

In [116]:

```
# feeding text data and recieving vectorized data
XtrainV= tfidfw2vVect(Xtrain)
```

```
XtestV= tfidfw2vVect(Xtest)
XcvV= tfidfw2vVect(Xcv)
```

In [117]:
```
XtrainV[1]
```

Out[117]:
```
array([ 0.16363911,  0.32138804,  0.17515039,  0.24090501, -0.00249188,
       -0.49882371, -0.1875781 ,  0.2897003 ,  0.482043  , -0.06329746,
       -0.71696047, -0.2770102 ,  0.12991777,  0.42718397, -1.07288269,
        0.49435643, -0.14185035, -0.12875395, -0.40426827, -0.09652584,
       -0.71185575, -0.56733497,  0.12341449,  0.21259896,  0.05615327,
        0.35141184, -0.80189985, -0.51499443, -0.61201957,  0.33656254,
       -0.6635482 ,  0.37967336, -0.00748498,  0.01036834, -0.27573696,
       -0.07499758,  0.4913544 ,  0.22101287,  0.13950133,  0.44244039,
       -0.53403474,  0.40211393,  0.47660193, -0.35098124,  0.49883436,
        0.53112405,  0.08664968,  0.79542032, -0.18145972,  0.1173344
9])
```

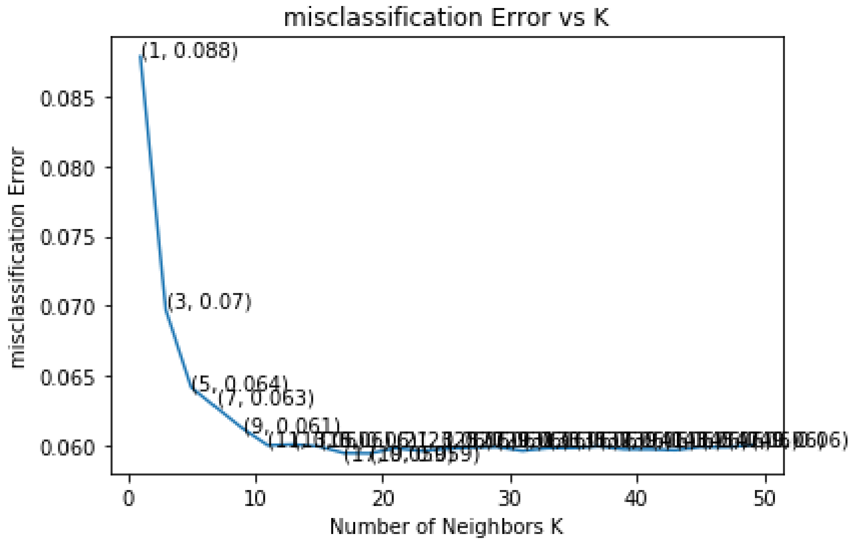
Storing data for sustainable use

In [118]:
```
import pickle
with open('tfidfw2vVectors.pkl','wb') as i:
    pickle.dump(XtrainV,i)
    pickle.dump(XcvV,i)
    pickle.dump(XtestV,i)
i.close()
```

KNN bruteforce on average tfidf-w2v

In [119]:
```
# calling for optimal k
k= k_classifier_brute(XtrainV,Ytrain,XcvV,Ycv)
```

```
The optimal number of neighbors is 19.
```

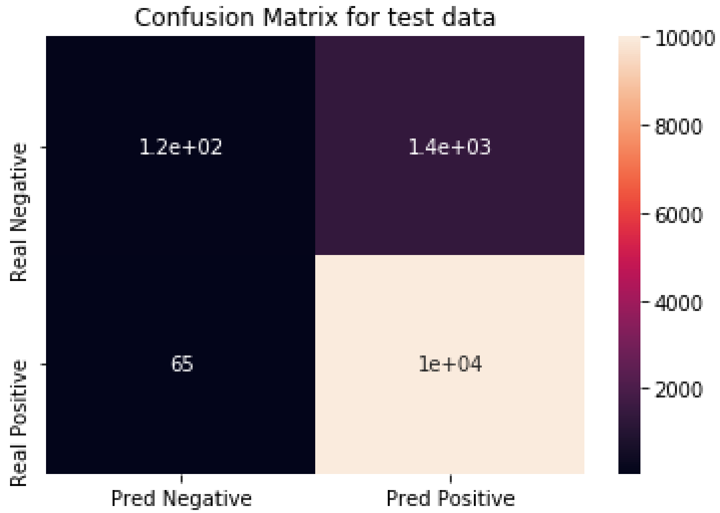



```
the misclassification error for each k value is :  [0.088 0.07  0.064
0.063 0.061 0.06  0.06  0.06  0.059 0.059 0.06  0.06
 0.06  0.06  0.06  0.06  0.06  0.06  0.06  0.06  0.06  0.06  0.06  0.06
 0.06 ]
With f1_score as  0.9406131125531438
```

Training our model on optimal k

In [120]:
```
knn= KNeighborsClassifier(n_neighbors=k,algorithm='brute')
knn.fit(XtrainV,Ytrain)
pred= knn.predict(XtestV)
```

In [9]:
```
cfm= confusion_matrix(Ytest,pred)
df_cm = pd.DataFrame(cfm,columns = ['Pred Negative','Pred Positive']\
                     ,index = ['Real Negative','Real Positive'])
plt.title('Confusion Matrix for test data')
sns.heatmap(df_cm, annot=True)
print('f1_score of Test data is ',f1_score(Ytest,pred))
```




```
f1_score of Test data is  0.9360386992743887
```

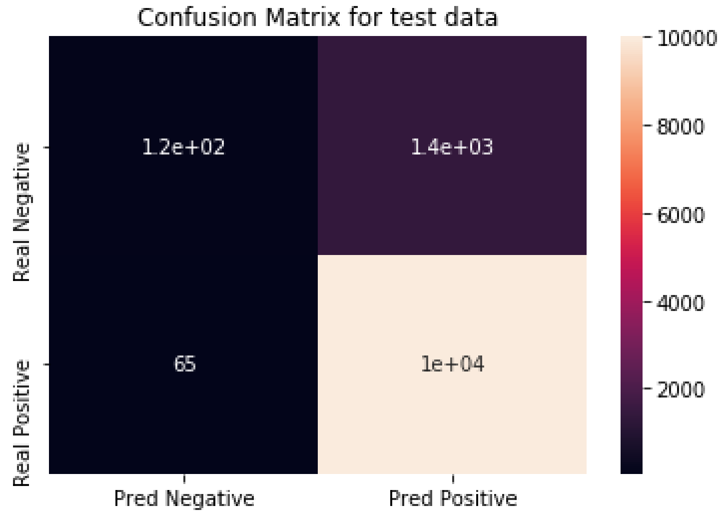
KNN kd-tree on average tfidf-w2v

In [122]:

```
knn= KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
knn.fit(XtrainV,Ytrain)
pred= knn.predict(XtestV)
```

In [10]:

```
cfm= confusion_matrix(Ytest,pred)
df_cm = pd.DataFrame(cfm,columns = ['Pred Negative','Pred Positive']\
                     ,index = ['Real Negative','Real Positive'])
plt.title('Confusion Matrix for test data')
sns.heatmap(df_cm, annot=True)
print('f1_score of Test data is ',f1_score(Ytest,pred))
```




```
f1_score of Test data is  0.9360386992743887
```

Observation:-

1. Opimal K is 19
2. CV score(F1_score) is 94.60 with CV error of 5.40
3. Testdata's F1_score of bruteforce is 93.60
4. Testdata's F1_score of kd_tree is 93.60
5. From confusion matrix we can say that around 1.4k points from 12k points of test data were wrong classified which implies accuracy of around 88.33

Summary

Vectorizer	Hyperparameter(K)	Bruteforce_F1_score	KD-tree_F1_score	CV_score

Vectorizer	Hyperparameter(K)	Bruteforce_F1_score	KD-tree_F1_score	CV_score
BOW	7	93.47	93.45	93.83
TF-IDF	5	93.40	93.40	93.75
Avg-W2V	15	93.69	93.69	94.60
TF-IDF-avg-W2V	19	93.6	93.6	94.06