Pradip Dharam

3. k-NN on Amazon

0
voters

last run 3 days ago · IPython Notebook HTML
using data from multiple data sources · Private Make Public

3] Apply k-NN on Amazon reviews data-set [M]

1. Apply k-NN brute and kd-tree
2. Featurize / Vectorize: using BoW, tf-idf, avg w2v, if-idf weighted w2v
3. Splitting of data set into train and test: Apply time based slicing. 70% fata is train dataset and rest 30% becomes test dataset
4. **Then report the accuracy:** 10 fold cross validation to find optimal k in k-NN. Report the test accuracy for all dataset vectors BoW, tf-idf, avg w2v, if-idf weighted w2v

SOLUTION:

I am using kaggle.com to complete exercise. I have pre-processed the data and stored the result is csv file and then commented the pre-processing code below.

I am using pre-processing code given by you in the course

TEXT PREPROCESSING

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.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
```

```
# using the SQLite Table to read data.
#con = sqlite3.connect('./amazon-fine-food-reviews/database.sqlite')
#con = sqlite3.connect('../input/database.sqlite')
con = sqlite3.connect('../input/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):
    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

display= pd.read_sql_query("""
 ###SELECT *
 ###FROM Reviews
 ###WHERE Score != 3 AND UserId="AR5J8UI46CURR"
 ###ORDER BY ProductID
""", con)
display

#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind='q
uicksort', na_position='last')

#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', in
place=False)
final.shape

#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100

display= pd.read_sql_query("""
```

```
 ###SELECT *
 ###FROM Reviews
 ###WHERE Score != 3 AND Id=44737 OR Id=64422
 ###ORDER BY ProductID
""", con)
display

final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]

"""
```

Out[1]:

```
', con)\ndisplay\n\nfinal=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]\n\n'
```

Text Preprocessing: Stemming, stop-word removal and Lemmatization

I am using the code given by you for pre-processing

In [2]:

```
"""
# find sentences containing HTML tags
import re
i=0;
for sent in final['Text'].values:
    if (len(re.findall('<.*?>', sent))):
        print(i)
        print(sent)
        break;
    i += 1;

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

#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 final['Text'].values:
    filtered_sentence=[]
    #print(sent);
    sent=cleanhtml(sent) # remove HTML1 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')
                    #Comment above line to remove stemming an replce this line with below line
                    #s=cleaned_words.lower()
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to describe positive
 reviews
                    if(final['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describe negative
 reviews reviews
                else:
                    continue
            else:
                continue
    #print(filtered_sentence)
    str1 = b" ".join(filtered_sentence) #final string of cleaned words
    #print("**************************************************************")

    final_string.append(str1)  ## Replace str1 by str1.lstrip('b\'').rstrip('\'')
    i+=1

final['CleanedText']=final_string #adding a column of CleanedText which displays the data after p
re-processing of the review
cleanedText=final['CleanedText']
```

```
finalScore = final['Score']
print(finalScore.head(2))
print(cleanedText.head(2))
print("Shape cleanedText: ",cleanedText.shape)
print("Shape: finalScore",finalScore.shape)

final1=pd.DataFrame()
final1['Text'] = final['Text']
final1['CleanedText']  = final['CleanedText']
final1['Time']  = final['Time']
final1['Score'] = finalScore

final1.to_csv('final1.csv')

"""
```

Out[2]:

```
'\n# find sentences containing HTML tags\nimport re\ni=0;\nfor sent in final[\'Text\'].value
s:\n    if (len(re.findall(\'<.*?>\', sent))):\n        print(i)\n        print(sent)\n
break;\n    i += 1;    \n\nimport re\n# Tutorial about Python regular expressions: https://pym
otw.com/2/re/\nimport string\nfrom nltk.corpus import stopwords\nfrom nltk.stem import PorterS
temmer\nfrom nltk.stem.wordnet import WordNetLemmatizer\n\nstop = set(stopwords.words(\'englis
h\')) #set of stopwords\nsno = nltk.stem.SnowballStemmer(\'english\') #initialising the snowba
ll stemmer\n\ndef cleanhtml(sentence): #function to clean the word of any html-tags\n    clean
r = re.compile(\'<.*?>\')\n    cleantext = re.sub(cleanr, \' \', sentence)\n    return cleante
xt\ndef cleanpunc(sentence): #function to clean the word of any punctuation or special charact
ers\n    cleaned = re.sub(r\'[?|!|\'|"|#]\',r\'\',sentence)\n    cleaned = re.sub(r\'[.|,|)|(|
\\|/]\',r\' \',cleaned)\n    return  cleaned\n\n#Code for implementing step-by-step the checks
mentioned in the pre-processing phase\n# this code takes a while to run as it needs to run on
500k sentences.\ni=0\nstr1=\' \'\nfinal_string=[]\nall_positive_words=[] # store words from +v
e reviews here\nall_negative_words=[] # store words from -ve reviews here.\ns=\'\'\nfor sent i
n final[\'Text\'].values:\n    filtered_sentence=[]\n    #print(sent);\n    sent=cleanhtml(sen
t) # remove HTml tags\n    for w in sent.split():\n        for cleaned_words in cleanpunc(w).s
plit():\n            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):    \n
    if(cleaned_words.lower() not in stop):\n                    s=(sno.stem(cleaned_words.lowe
r())).encode(\'utf8\')\n                    #Comment above line to remove stemming an replce t
his line with below line\n                    #s=cleaned_words.lower()\n                    fi
ltered_sentence.append(s)\n                    if (final[\'Score\'].values)[i] == \'positive
\': \n                        all_positive_words.append(s) #list of all words used to describe
positive reviews\n                    if(final[\'Score\'].values)[i] == \'negative\':\n
                all_negative_words.append(s) #list of all words used to describe negative rev
iews reviews\n                else:\n                    continue\n            else:\n
        continue \n    #print(filtered_sentence)\n    str1 = b" ".join(filtered_sentence) #fin
al string of cleaned words\n    #print("**********************************************************
**********")\n    \n    final_string.append(str1)  ## Replace str1 by str1.lstrip(\'b
\'\').rstrip(\'\'\')\n    i+=1\n    \nfinal[\'CleanedText\']=final_string #adding a column of
```

```
CleanedText which displays the data after pre-processing of the review \ncleanedText=final[\'C
leanedText\']\nfinalScore = final[\'Score\']\nprint(finalScore.head(2))\nprint(cleanedText.hea
d(2))\nprint("Shape cleanedText: ",cleanedText.shape)\nprint("Shape: finalScore",finalScore.sh
ape)\n\nfinal1=pd.DataFrame()\nfinal1[\'Text\'] = final[\'Text\']\nfinal1[\'CleanedText\']  =
final[\'CleanedText\']\nfinal1[\'Time\']  = final[\'Time\']\nfinal1[\'Score\'] = finalScore\n
\nfinal1.to_csv(\'final1.csv\')   \n\n'
```

Importing required libraries

In [3]:

```
%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.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

from sklearn.cross_validation import train_test_split
from sklearn.model_selection import TimeSeriesSplit
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
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: Thi
s 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 ite
rators are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Preprocessed data was stored in csv file. Reading the same into panda data frame variable data

In [4]:
```
#data.CleanedText[1]=data.CleanedText[1].lstrip('b\'').rstrip('\'')
#data.CleanedText[1]
#data.CleanedText[]=data.CleanedText[].lstrip('b\'').rstrip('\'')
#(txt. for txt in data.CleanedText)
#data.CleanedText[1]
data=pd.read_csv('../input/final1-with-time/final1 (1).csv')
"""
i=0
for sent in data.CleanedText:
    data.CleanedText[i]=sent.lstrip('b\'').rstrip('\'')
    i+=1
#CleanedText_trimmed=data.CleanedText
print(CleanedText.head)
data.sort_values(['Time'], ascending=1)
"""
```

Out[4]:
```
"\ni=0\nfor sent in data.CleanedText:\n    data.CleanedText[i]=sent.lstrip('b'').rstrip(''')\n
    i+=1\n#CleanedText_trimmed=data.CleanedText\nprint(CleanedText.head)\ndata.sort_values(['T
ime'], ascending=1)\n"
```

#

Defining below functions

1. Function knn10fold_and_MSE

- This function takes training data both X_train and Y_train to train the model
- This function also takes argument algo where in we can provide whether to use brute or kd_tree
- We need to do time based splitting but not random while training 10 fold cross validation kNN model; I am using TimeSeriesSplit and passing 10 as argument and my traing data is already sorted in ascending order.
- I did added print statements to undestand the execution after writing code. Now those print statements are commented.
- This function plots the graph "Number of Neighbors K')" against "Misclassification Error"
- It takes average accuracy of 10 folds for each k and then calculates the mis-classification error as (1-average_accuracy_for_each_k)
- It then select the optimal k such that misclassification error is minimum and returns the same

In [5]:

```
def knn10fold_and_MSE(x_train, y_train, algo):
    neighbors = list(range(1,20,2))
    cv_scores = []
    # perform 10-fold cross validation
    time_series_10foldcv = TimeSeriesSplit(n_splits=10)
    for k in neighbors:
        #print("kNN for k=",k)
        #knn = KNeighborsClassifier(n_neighbors=k, weights='distance', algorithm=algo)
        knn = KNeighborsClassifier(n_neighbors=k, weights='uniform', algorithm=algo)
        #Below function does random k' folds and not time based, that why this is not used
        #scores = cross_val_score(knn, x_train, y_train, cv=10, scoring='accuracy')
        this_cv_scores=[]
        for train_index, cv_index in time_series_10foldcv.split(x_train):
            # tscv_data.split(x_train) : This will give 10 iterations
            # Each iteration has train and cross validation indexes based on time but not the ran
dom splitting
            # But not the random creation of train and cross validation data sets

            ######### We get the indexes in sequence since TimeSeriesSplit i.t time based splitti
ng #########
            ##print("Train set index min :", np.min(train_index))
            ##print("Train set index max :", np.max(train_index))
            ##print("CV set index min    :", np.min(cv_index))
            ##print("CV set index max    :", np.max(cv_index))

            X_train, X_cv = x_train[train_index], x_train[cv_index]
            Y_train, Y_cv = y_train[train_index], y_train[cv_index]

            #Randomly choosing data points to train the model
            #sample_indices = np.random.choice(train_index[-1],20000)
            #x_train_sample = X_train[sample_indices]
            #y_train_sample = Y_train.iloc[sample_indices]
            #knn.fit(x_train_sample, y_train_sample)

            knn.fit(X_train, Y_train)
            pred = knn.predict(X_cv)
            accuracy = accuracy_score(Y_cv, pred, normalize=True)
            #print("Accuracy:",accuracy)
            this_cv_scores.append(accuracy)

        ##print("k=",k, " 10 folds cross validation scores ", this_cv_scores)
        ##print("k=",k, " Agerage is: ", np.average(this_cv_scores))
        cv_scores.append(np.average(this_cv_scores))
    # changing to misclassification error
    MSE = [1 - x for x in cv_scores]
```

```
    # determining best k
    best_k_index = MSE.index(min(MSE))
    optimal_k = neighbors[best_k_index]
    CV_Accuracy = cv_scores[best_k_index] * 100
    # 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.xlabel('Number of Neighbors K')
    plt.ylabel('Misclassification Error')
    plt.show()
    print("The misclassification error for each k value is : ", np.round(MSE,3))
    print('\nThe optimal number of neighbors is %d and respective accuracy over training data o
r cross validation accuracy is %f%%' % (optimal_k,CV_Accuracy))

    return optimal_k
##############################################################################################
```

2. Function knn_with_optimal_k

- Here, lets test the model over unknown data which is x_test and y_test here
- Train the model using x_train and y_train; then predict using x_test
- Then get the accuracy using prected class labels and y_train and report the accuracy for optimal k

In [6]:

```
def knn_with_optimal_k(x_train, y_train, x_test, y_test, optimal_k, algo):
    # ============================== KNN with k = optimal_k ====================================
==========
    # instantiate learning model k = optimal_k
    #knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k, weights='distance', algorithm=alg
o)
    knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k, weights='uniform', algorithm=algo
)
    # fitting the model
    knn_optimal.fit(x_train, y_train)
    # predict the response
    pred = knn_optimal.predict(x_test)
    # evaluate accuracy
    acc = accuracy_score(y_test, pred) * 100
    print('\nThe test accuracy or accuracy over unknown data of the ', algo ,' knn classifier f
or optimal k = %d is %f%%' % (optimal_k, acc))
    print('\n')
    print('\n##############################################################################################
############')
```

```
########## )
    print('\nOBSERVATION: Able to achieve ', acc, '% of test accuracy using ', algo ,' knn clas
sifier' )
    print('\n##################################################################################
##########')
    print('\n')
```

#

Sorting data in ascending order since time based splitting

In [7]:
```
#Sorting the data on ascending order of Time since time based splitting needs to be done further
data=data.sort_values(['Time'], ascending=1)
print("Time min:", data['Time'].min())
print("Time max:", data['Time'].max())
```

```
Time min: 939340800
Time max: 1351209600
```

In [8]:
```
#import numpy as np
#from sklearn.cross_validation import train_test_split
#a = np.arange(10).reshape((5, 2))
#b = np.arange(5)
#print("Array a: \n",a); print("Array b: \n",b);

#a_train, a_test, b_train, b_test = train_test_split(a, b,  test_size=0.3,  random_state=0)
#print("a_train :\n", a_train)
#print("b_train :\n", b_train)
```

#

BAG OF WORDS

Getting the BoW vector representation for all reviews

In [9]:

In [9]:
```
#BoW
count_vect = CountVectorizer() #in scikit-learn
final_counts_bow = count_vect.fit_transform(data.CleanedText.values)
final_counts_bow.shape
```

Out[9]:
```
(364171, 71632)
```

Processing only 5000 records since entire dataset processing takes too much RAM

In [10]:
```
print(type(final_counts_bow))
#X = final_counts_bow[0:50000,:].toarray()
#Y = np.array(data['Score'][0:50000])
### Mention the total number of data points to be worked on since I have machine with low configu
ration
num_of_points=5000
X = final_counts_bow[0:num_of_points,:].toarray()
Y = np.array(data['Score'][0:num_of_points])
print("X Shape : ",X.shape, " X Ndim: ",X.ndim)
print("Y Shape : ",Y.shape, " Y Ndim: ",Y.ndim)
#print(X); print(Y)
data['Score'][0:num_of_points].describe()
```

```
<class 'scipy.sparse.csr.csr_matrix'>
X Shape :  (5000, 71632)  X Ndim:  2
Y Shape :  (5000,)  Y Ndim:  1
```

Out[10]:
```
count         5000
unique           2
top       positive
freq          4423
Name: Score, dtype: object
```

Splitting dataset into train and test with 70:30 ratio

In [11]:
```
# Train to test ratio is 70:30
boundry=int(num_of_points*0.7)
print("Boundry: ", boundry)
# split the data set into train and test based in time and not random splitting
```

```
x_train = X[:boundry]; x_test = X[boundry:]
y_train = Y[:boundry]; y_test = Y[boundry:]

#x_train, x_test, y_train, y_test = cross_validation.train_test_split(X, Y, test_size=0.3, random
_state=0)
print("X train Shape : ",x_train.shape, " X Ndim: ",x_train.ndim); print("Y train Shape : ",y_t
rain.shape, " Y Ndim: ",y_train.ndim)
print("X test Shape : ",x_test.shape, " X Ndim: ",x_test.ndim); print("Y test Shape : ",y_test.
shape, " Y Ndim: ",y_test.ndim)
#print("Time min:", data['Time'].min())
#print("Time max:", data['Time'].max())
```

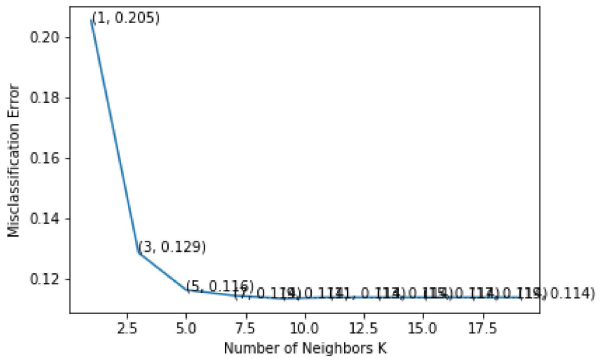
```
Boundry:  3500
X train Shape :  (3500, 71632)  X Ndim:  2
Y train Shape :  (3500,)  Y Ndim:  1
X test Shape :  (1500, 71632)  X Ndim:  2
Y test Shape :  (1500,)  Y Ndim:  1
```

Optimal k accuracy: Brute 10 fold k-NN for BoW

- Performing kNN 10 fold cross validation over training data to get the accuracy and mis classification error using kNN Brute algorithm
- Getting optimal k and with low misclassification error
- Then accuracy for test data is reported using optimal k

In [12]:

```
optimal_k = knn10fold_and_MSE(x_train, y_train, 'brute')
knn_with_optimal_k(x_train, y_train, x_test, y_test, optimal_k, 'brute')
```



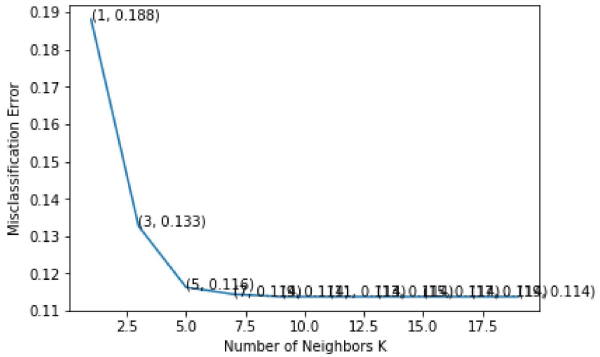

```
The misclassification error for each k value is :  [0.205 0.129 0.116 0.114 0.114 0.114 0.114
```

```
0.114 0.114 0.114]

The optimal number of neighbors is 9 and respective accuracy over training data or cross valid
ation accuracy is 88.647799%

The test accuracy or accuracy over unknown data of the  brute  knn classifier for optimal k =
9 is 87.066667%



#############################################################################################

OBSERVATION: Able to achieve  87.06666666666666 % of test accuracy using  brute  knn classifie
r

#############################################################################################
```

Optimal k accuracy: kd-tree 10 fold k-NN for BoW

- Performing kNN 10 fold cross validation over training data to get the accuracy and mis classification error using kNN kd_tree algorithm
- Getting optimal k and with low misclassification error
- Then accuracy for test data is reported using optimal k

In [13]:
```
optimal_k = knn10fold_and_MSE(x_train, y_train, 'kd_tree')
knn_with_optimal_k(x_train, y_train, x_test, y_test, optimal_k, 'kd_tree')
```



```
The misclassification error for each k value is :  [0.188 0.133 0.116 0.114 0.114 0.114 0.114
0.114 0.114 0.114]

The optimal number of neighbors is 9 and respective accuracy over training data or cross valid
ation accuracy is 88.616352%

The test accuracy or accuracy over unknown data of the  kd_tree  knn classifier for optimal k
= 9 is 86.933333%



#############################################################################################

OBSERVATION: Able to achieve  86.93333333333332 % of test accuracy using  kd_tree  knn classif
ier

#############################################################################################
```

#

TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY

Getting the tf-idf vector representation for all reviews

In [14]:
```
#TF-IDF
tf_idf_vect = TfidfVectorizer()
final_tf_idf = tf_idf_vect.fit_transform(data.CleanedText.values)
print("final_tf_idf shape: ",final_tf_idf.shape)
```

```
final_tf_idf shape:  (364171, 71632)
```

Processing only 5000 records since entire dataset processing takes too much RAM

In [15]:
```
# Define data points and labels
### Mention the total number of data points to be worked on since I have machine with low configu
```

```
### mention the total number of data points to be worked on since I have machine with low configu
ration
num_of_points=5000
X = final_tf_idf[0:num_of_points,:].toarray()
Y = np.array(data['Score'][0:num_of_points])
print("X Shape : ",X.shape, " X Ndim: ",X.ndim)
print("Y Shape : ",Y.shape, " Y Ndim: ",Y.ndim)
data['Score'][0:num_of_points].describe()
```

```
X Shape :  (5000, 71632)  X Ndim:  2
Y Shape :  (5000,)  Y Ndim:  1
```

Out[15]:

```
count         5000
unique           2
top       positive
freq          4423
Name: Score, dtype: object
```

Splitting dataset into train and test with 70:30 ratio

In [16]:

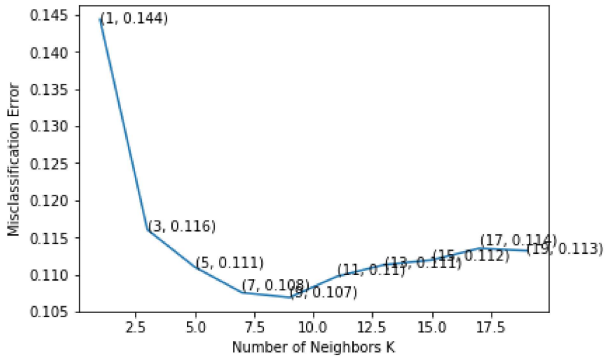
```
# Train to test ratio is 70:30
boundry=int(num_of_points*0.7)
print("Boundry: ", boundry)
# split the data set into train and test based in time and not random splitting
x_train = X[:boundry]; x_test = X[boundry:]
y_train = Y[:boundry]; y_test = Y[boundry:]

#x_train, x_test, y_train, y_test = cross_validation.train_test_split(X, Y, test_size=0.3, random
_state=0)
print("X train Shape : ",x_train.shape, " X Ndim: ",x_train.ndim); print("Y train Shape : ",y_t
rain.shape, " Y Ndim: ",y_train.ndim)
print("X test Shape : ",x_test.shape, " X Ndim: ",x_test.ndim);    print("Y test Shape : ",y_te
st.shape, " Y Ndim: ",y_test.ndim)
```

```
Boundry:  3500
X train Shape :  (3500, 71632)  X Ndim:  2
Y train Shape :  (3500,)  Y Ndim:  1
X test Shape :  (1500, 71632)  X Ndim:  2
Y test Shape :  (1500,)  Y Ndim:  1
```

Optimal k accuracy: Brute 10 fold k-NN for TF-IDF

- Performing kNN 10 fold cross validation over training data to get the accuracy and mis classification error using kNN Brute algorithm
- Getting optimal k and with low misclassification error
- Then accuracy for test data is reported using optimal k

In [17]:
```
optimal_k = knn10fold_and_MSE(x_train, y_train, 'brute')
knn_with_optimal_k(x_train, y_train, x_test, y_test, optimal_k, 'brute')
```

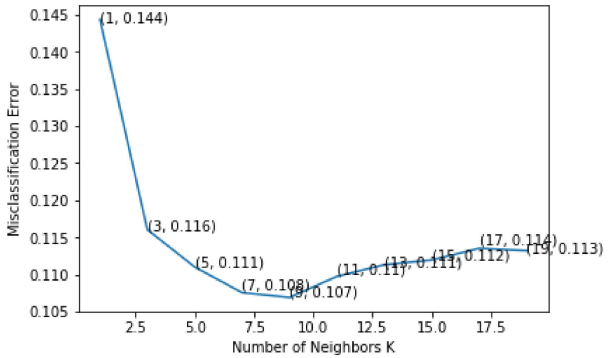



```
The misclassification error for each k value is :  [0.144 0.116 0.111 0.108 0.107 0.11  0.111
0.112 0.114 0.113]

The optimal number of neighbors is 9 and respective accuracy over training data or cross valid
ation accuracy is 89.308176%

The test accuracy or accuracy over unknown data of the  brute  knn classifier for optimal k =
9 is 87.200000%



#################################################################################################

OBSERVATION: Able to achieve  87.2 % of test accuracy using  brute  knn classifier

#################################################################################################
```

Optimal k accuracy: kd-tree 10 fold k-NN for TF-IDF

- Performing kNN 10 fold cross validation over training data to get the accuracy and mis classification error using kNN kd_tree algorithm
- Getting optimal k and with low misclassification error
- Then accuracy for test data is reported using optimal k

In [18]:
```
optimal_k = knn10fold_and_MSE(x_train, y_train, 'kd_tree')
knn_with_optimal_k(x_train, y_train, x_test, y_test, optimal_k, 'kd_tree')
```




```
The misclassification error for each k value is :  [0.144 0.116 0.111 0.108 0.107 0.11  0.111
0.112 0.114 0.113]

The optimal number of neighbors is 9 and respective accuracy over training data or cross valid
ation accuracy is 89.308176%

The test accuracy or accuracy over unknown data of the  kd_tree  knn classifier for optimal k
= 9 is 87.200000%



#################################################################################################

OBSERVATION: Able to achieve  87.2 % of test accuracy using  kd_tree  knn classifier

#################################################################################################
```

#

I'm using google word to vector to measure semantic similarities

Word2Vec

In [19]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
model = KeyedVectors.load_word2vec_format('../input/word2vec-model/GoogleNews-vectors-negative3
00.bin', binary=True)
```

#

Average Word2Vec

Processing only 5000 records since entire dataset processing takes too much RAM

In [20]:
```
### Mention the total number of data points to be worked on since I have machine with low configu
ration
num_of_points=5000
# Define data points and labels
X = data.CleanedText[0:num_of_points]
#Y = np.array(data['Score'][0:500])
```

Generating the Average word3vec vector representation for all reviews. Getting vector for each word in review and creating new scentense vector such that its an average of word2vector of all the words in review text

In [21]:
```
# average Word2Vec
# compute average word2vec for each review.
i=0
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in X: # for each review/sentence
    #if (i == 1626 or i == 1698 or i == 1950 or i == 4532 or i == 4752 ):
```

```
        #print("i=",i," sent: ", sent)
    #print("Sentense : ", sent, "Type: ",type(sent))
    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 str(sent).split():
        #print("word: ", str(word))
    for word in str(sent).split(): # for each word in a review/sentence
        try:
            #print("word : ", word)
            vec = model.wv[word]
            #print("vec : ", vec.shape)
            sent_vec += vec
            cnt_words += 1
        except:
            #print("Exception")
            pass
    """
    if (i == 1626 or i == 1698 or i == 1950 or i == 4532 or i == 4752 ):
        print("sent: ", sent)
        print("cnt_words=",cnt_words)
        print("i=",i," sent_vec: ", sent_vec)
    """
    if (cnt_words !=0):
        sent_vec /= cnt_words
        ## If we divide zero by zero then it generated NaN will cause issue further
    """
    if (i == 1626 or i == 1698 or i == 1950 or i == 4532 or i == 4752 ):
        print("i=",i," sent_vec: ", sent_vec)
    """
    sent_vectors.append(sent_vec)
    i+=1
print(len(sent_vectors))
print(len(sent_vectors[0]))
#print("Shape: ",sent_vectors.shape)
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:16: DeprecationWarning: Call to d
eprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  app.launch_new_instance()


5000
300
```

#

I was facing below error and fixed the same

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

- ValueError Traceback (most recent call last)
- <ipython-input-57-2727e6eef3b2> in ()
- ----> 1 optimal_k = knn10fold_and_MSE(x_train, y_train, 'brute')
- ---> 34 pred = knn.predict(X_cv)
- --> 143 X = check_array(X, accept_sparse='csr')
- --> 453 _assert_all_finite(array)
- ---> 44 " or a value too large for %r." % X.dtype)
- ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

Root cause of the issue: Below words in below pre-processed review texts (which are as sent in below cell for 5 of the review texts) does not have vector in googles word2vec. We can train own word to vector as well

In [22]:

```
#for i in range(0,5000):
#    for j in range(0,300):
#        if(np.isnan(X[i][j])):
#            print("i=",i," j=",j)
#            break

    # i= 1626  j= 0
    # i= 1698  j= 0
    # i= 1950  j= 0
    # i= 4532  j= 0
    # i= 4752  j= 0


#[4752]
#ent_vectors[4752]
#data.CleanedText[1626]
"""
i= 1626  sent:  b'worth'
i= 1698  sent:  b'excel'
i= 1950  sent:  b'calori noodl delici fill'
i= 4532  sent:  b'good chewi salti sweet'
i= 4752  sent:  b'tea delici hestit buy'
```

```
"""
#model.wv['hestit']
```

Out[22]:

```
"\ni= 1626  sent:  b'worth'\ni= 1698  sent:  b'excel'\ni= 1950  sent:  b'calori noodl delici f
ill'\ni= 4532  sent:  b'good chewi salti sweet'\ni= 4752  sent:  b'tea delici hestit buy'\n"
```

In [23]:

```
##########################################################
```

In [24]:

```
X = np.array(sent_vectors)
Y = np.array(data['Score'][0:num_of_points])
print("X Shape : ",X.shape, " X Ndim: ",X.ndim)
print("Y Shape : ",Y.shape, " Y Ndim: ",Y.ndim)
data['Score'][0:num_of_points].describe()
```

```
X Shape :  (5000, 300)  X Ndim:  2
Y Shape :  (5000,)  Y Ndim:  1
```

Out[24]:

```
count         5000
unique           2
top       positive
freq          4423
Name: Score, dtype: object
```

Splitting dataset into train and test with 70:30 ratio

In [25]:

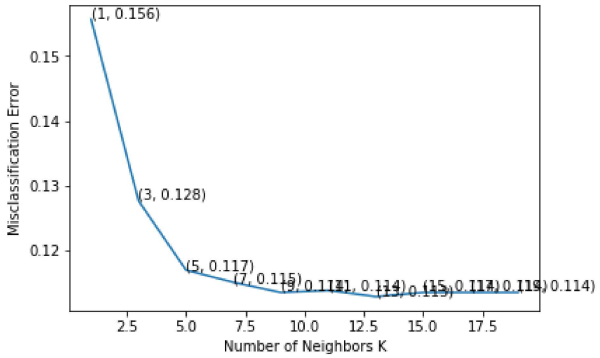
```
# Train to test ratio is 70:30
boundry=int(num_of_points*0.7)
print("Boundry: ", boundry)
# split the data set into train and test based in time and not random splitting
x_train = X[:boundry]; x_test = X[boundry:]
y_train = Y[:boundry]; y_test = Y[boundry:]

#x_train, x_test, y_train, y_test = cross_validation.train_test_split(X, Y, test_size=0.3, random
_state=0)
print("X train Shape : ",x_train.shape, " X Ndim: ",x_train.ndim); print("Y train Shape : ",y_t
rain.shape, " Y Ndim: ",y_train.ndim)
print("X test Shape : ",x_test.shape, " X Ndim: ",x_test.ndim);    print("Y test Shape : ",y_te
st.shape, " Y Ndim: ",y_test.ndim)
```

```
Boundry:  3500
X train Shape :  (3500, 300)  X Ndim:  2
Y train Shape :  (3500,)  Y Ndim:  1
X test Shape :  (1500, 300)  X Ndim:  2
Y test Shape :  (1500,)  Y Ndim:  1
```

Optimal k accuracy: Brute 10 fold k-NN for Avg word2vec

- Performing kNN 10 fold cross validation over training data to get the accuracy and mis classification error using kNN Brute algorithm
- Getting optimal k and with low misclassification error
- Then accuracy for test data is reported using optimal k

In [26]:
```
optimal_k = knn10fold_and_MSE(x_train, y_train, 'brute')
knn_with_optimal_k(x_train, y_train, x_test, y_test, optimal_k, 'brute')
```




```
The misclassification error for each k value is :  [0.156 0.128 0.117 0.115 0.114 0.114 0.113
0.114 0.114 0.114]

The optimal number of neighbors is 13 and respective accuracy over training data or cross vali
dation accuracy is 88.710692%

The test accuracy or accuracy over unknown data of the  brute  knn classifier for optimal k =
13 is 87.066667%
```

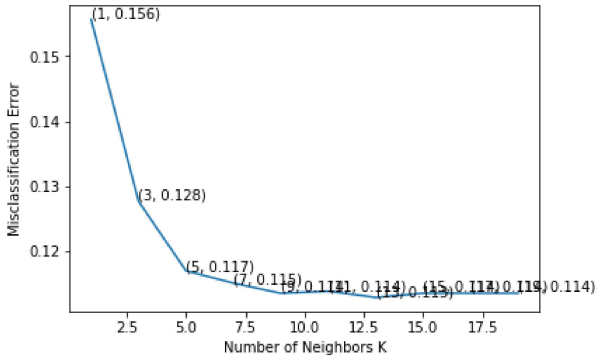
```
##########################################################################################
```

```
OBSERVATION: Able to achieve  87.06666666666666 % of test accuracy using  brute  knn classifie
r

##############################################################################################
```

Optimal k accuracy: kd_tree 10 fold k-NN for Avg word2vec

- Performing kNN 10 fold cross validation over training data to get the accuracy and mis classification error using kNN kd_tree algorithm
- Getting optimal k and with low misclassification error
- Then accuracy for test data is reported using optimal k

In [27]:
```
optimal_k = knn10fold_and_MSE(x_train, y_train, 'kd_tree')
knn_with_optimal_k(x_train, y_train, x_test, y_test, optimal_k, 'kd_tree')
```




```
The misclassification error for each k value is :  [0.156 0.128 0.117 0.115 0.114 0.114 0.113
0.114 0.114 0.114]

The optimal number of neighbors is 13 and respective accuracy over training data or cross vali
dation accuracy is 88.710692%

The test accuracy or accuracy over unknown data of the  kd_tree  knn classifier for optimal k
= 13 is 87.066667%
```

```
##############################################################################################

OBSERVATION: Able to achieve  87.06666666666666 % of test accuracy using  kd_tree  knn classif
ier

##############################################################################################
```

#

TF-IDF weighted Word2Vec

Processing only 5000 records since entire dataset processing takes too much RAM

In [28]:
```
### Mention the total number of data points to be worked on since I have machine with low configu
ration
num_of_points=5000
X = data.CleanedText[0:num_of_points]
# TF-IDF weighted Word2Vec
```

Getting vector for each word in review and creating new scentense vector such that word2vector is been weighted with respective TF-IDF of the review

In [29]:
```
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 X: # 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
    #sent='littl book make son laugh loud recit car drive along alway sing refrain hes learn whal
e india droop love new word book introduc silli classic book will bet son still abl recit memori
 colleg'
    #print("Sentense : ", sent)
    for word in sent.split(): # for each word in a review/sentence
        try:
```

```
            try:
                #print("Word: ", word)
                vec = model.wv[word]
                #print("Vector: ", vec, " Size: ", vec.shape)
                # obtain the tf_idfidf of a word in a sentence/review
                tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
                #print("tfidf: ", tf_idf)
                sent_vec += (vec * tf_idf)
                #print("sent_vec Calculated")
                weight_sum += tf_idf
            except:
               # print("Exception")
                #e = sys.exc_info()[0]
                #print("Exception: ",e)
                ##write_to_page( "<p>Error: %s</p>" % e )
                pass
    zero_weight_sum_count=0
    if(weight_sum != 0):
        sent_vec /= weight_sum
    else        :
        zero_weight_sum_count += 1
    tfidf_sent_vectors.append(sent_vec)
    row += 1

print(len(tfidf_sent_vectors))
print(len(tfidf_sent_vectors[0]))
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:15: DeprecationWarning: Call to d
eprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  from ipykernel import kernelapp as app


5000
300
```

In [30]:

```
X = np.array(tfidf_sent_vectors)
Y = np.array(data['Score'][0:num_of_points])
print("X Shape : ",X.shape, " X Ndim: ",X.ndim)
print("Y Shape : ",Y.shape, " Y Ndim: ",Y.ndim)
data['Score'][0:num_of_points].describe()
```

```
X Shape :  (5000, 300)  X Ndim:  2
Y Shape :  (5000,)  Y Ndim:  1
```

Out[30]:

```
count         5000
unique           2
top       positive
freq          4423
Name: Score, dtype: object
```

Splitting dataset into train and test with 70:30 ratio

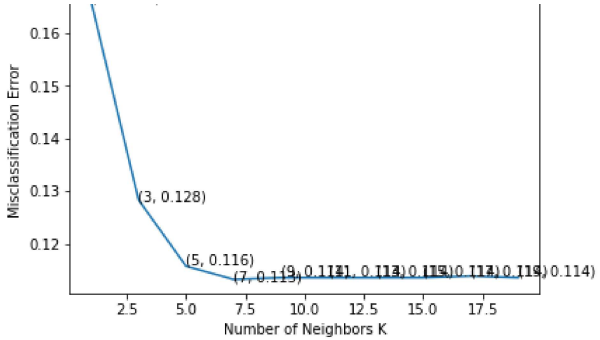
In [31]:
```
# Train to test ratio is 70:30
boundry=int(num_of_points*0.7)
print("Boundry: ", boundry)
# split the data set into train and test based in time and not random splitting
x_train = X[:boundry]; x_test = X[boundry:]
y_train = Y[:boundry]; y_test = Y[boundry:]

#x_train, x_test, y_train, y_test = cross_validation.train_test_split(X, Y, test_size=0.3, random
_state=0)
print("X train Shape : ",x_train.shape, " X Ndim: ",x_train.ndim); print("Y train Shape : ",y_t
rain.shape, " Y Ndim: ",y_train.ndim)
print("X test Shape : ",x_test.shape, " X Ndim: ",x_test.ndim);    print("Y test Shape : ",y_te
st.shape, " Y Ndim: ",y_test.ndim)
```

```
Boundry:  3500
X train Shape :  (3500, 300)  X Ndim:  2
Y train Shape :  (3500,)  Y Ndim:  1
X test Shape :  (1500, 300)  X Ndim:  2
Y test Shape :  (1500,)  Y Ndim:  1
```

Optimal k accuracy: Brute 10 fold k-NN for TF-IDF word2vec

- Performing kNN 10 fold cross validation over training data to get the accuracy and mis classification error using kNN Brute algorithm
- Getting optimal k and with low misclassification error
- Then accuracy for test data is reported using optimal k

In [32]:
```
optimal_k = knn10fold_and_MSE(x_train, y_train, 'brute')
knn_with_optimal_k(x_train, y_train, x_test, y_test, optimal_k, 'brute')
```

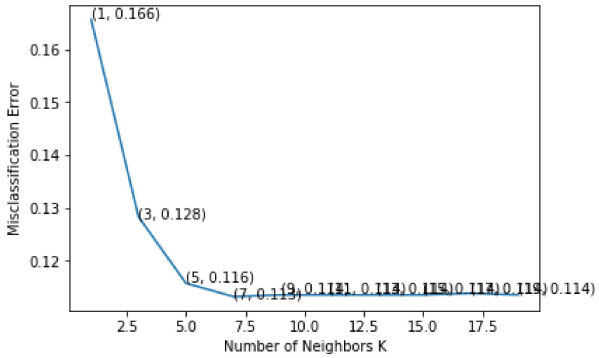



```
The misclassification error for each k value is :  [0.166 0.128 0.116 0.113 0.114 0.114 0.114
0.114 0.114 0.114]

The optimal number of neighbors is 7 and respective accuracy over training data or cross valid
ation accuracy is 88.679245%

The test accuracy or accuracy over unknown data of the  brute  knn classifier for optimal k =
7 is 87.466667%



#################################################################################################

OBSERVATION: Able to achieve  87.46666666666667 % of test accuracy using  brute  knn classifie
r

#################################################################################################
```

Optimal k accuracy: kd_tree 10 fold k-NN for TF-IDF word2vec

- Performing kNN 10 fold cross validation over training data to get the accuracy and mis classification error using kNN kd_tree algorithm
- Getting optimal k and with low misclassification error
- Then accuracy for test data is reported using optimal k

In [33]:
```
optimal_k = knn10fold_and_MSE(x_train, y_train, 'kd_tree')
knn_with_optimal_k(x_train, y_train, x_test, y_test, optimal_k, 'kd_tree')
```




```
The misclassification error for each k value is :  [0.166 0.128 0.116 0.113 0.114 0.114 0.114
0.114 0.114 0.114]

The optimal number of neighbors is 7 and respective accuracy over training data or cross valid
ation accuracy is 88.679245%

The test accuracy or accuracy over unknown data of the  kd_tree  knn classifier for optimal k
= 7 is 87.466667%



##############################################################################################

OBSERVATION: Able to achieve  87.46666666666667 % of test accuracy using  kd_tree  knn classif
ier

##############################################################################################
```

#

Did you find this Kernel useful?
Show your appreciation with an upvote

0

Comments (0)

Sort by Select...

Click here to enter a comment...