Predicting reviews in Amazon Fine Food Reviews data set using KNN.

Here we are using 10 fold cross-validation to predict our optimal k . And then we are using that k and predicting our accuracy for the test Dataset

Introduction to the Dataset

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (Rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use the Score/Rating. A rating of 4 or 5 could be cosnidered a positive review. A review of 1 or 2 could be considered negative. A review of 3 is nuetral and ignored. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

Importing all the necessary packages.

In [2]:

```
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
import sqlite3
import pandas as pd
import nltk
import string
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
```

```
C:\Users\Yaakuza\Anaconda3\lib\site-packages\sklearn\cross_validation.py:4
1: DeprecationWarning: This module was deprecated in version 0.18 in favor
of the model_selection module into which all the refactored classes and fu
nctions are moved. Also note that the interface of the new CV iterators ar
e different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

2. Connecting to Amazon food review dataset

In [3]:

```
con=sqlite3.connect('./database.sqlite')
filtered_data=pd.read_sql_query("""select * from reviews where score!=3""",con)
def partition(x):
    if x<3:
        return 'negative'
    else:
        return 'positive'
actual_score=filtered_data['Score']
PositiveNegative=actual_score.map(partition)
filtered_data['Score']=PositiveNegative
print(filtered_data.shape)
filtered_data.head()
```

```
(525814, 10)
```

Out[3]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0

3. Sorting our data on the basis of date and removing the Duplicate reviews

In [4]:

```
sorted_data=filtered_data.sort_values('ProductId',axis=0,ascending=True,inplace=False,k
ind='quicksort',na_position='last')
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep='f
irst',inplace=False)
print(final.shape)
```

```
(364173, 10)
```

4. we are also removing the rows which has HelpfulnessDenominator greater then HelpfulnessNumerator because its not practically possile

In [5]:

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

In [6]:

```
print(final.shape)
```

```
(364171, 10)
```

In [7]:

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

we are also cleaning our text of html tags , stop words, and puncuations

In [8]:

```
# 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;
```

```
6
I set aside at least an hour each day to read to my son (3 y/o). At this p
oint, I consider myself a connoisseur of children's books and this is one
of the best. Santa Clause put this under the tree. Since then, we've read
it perpetually and he loves it.<br /><br />First, this book taught him the
months of the year.<br /><br />Second, it's a pleasure to read. Well suite
d to 1.5 y/o old to 4+.<br /><br />Very few children's books are worth own
ing. Most should be borrowed from the library. This book, however, deserve
s a permanent spot on your shelf. Sendak's best.
```

In [9]:

```
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 char
acters
    cleaned = re.sub(r'[?|!|\'|"|#]',r'',sentence)
    cleaned = re.sub(r'[.|,|)|(|\|/]',r' ',cleaned)
    return  cleaned
print(stop)
print('************************************')
print(sno.stem('tasty'))
```

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

In [10]:

```
#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 HTML tags
    for w in sent.split():
        for cleaned_words in cleanpunc(w).split():
            if((cleaned_words.isalpha()) & (len(cleaned_words)>2)):
                if(cleaned_words.lower() not in stop):
                    s=(sno.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(s)
                    if (final['Score'].values)[i] == 'positive':
                        all_positive_words.append(s) #list of all words used to describ
e positive reviews
                    if(final['Score'].values)[i] == 'negative':
                        all_negative_words.append(s) #list of all words used to describ
e 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)
    i+=1
```

In [11]:

```
final['CleanedText']=final_string #adding a column of CleanedText which displays the da
ta after pre-processing of the review
```

In [12]:

```
final.head(3) #below the processed review can be seen in the CleanedText Column


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

6. Here we are Seperating all the review information of user on the basis of their Score i.e positive or negative.

Then we are taking 306913 positive and 57087 negative reviews respectively from positive and negative data frame and we are concating them together in one data frame bigdata. We are also taking the scores of these 364000 reviews seperately in s1. We then divide 364000 reviews to train and test data, and we convert the text column of the test and train into BOW.

In [13]:

```
total_data=final.sample(364000)
```

In [14]:

```
conn = sqlite3.connect('total_data.sqlite')
c=conn.cursor()
conn.text_factory = str
total_data.to_sql('total', conn, flavor=None, schema=None, if_exists='replace', index=T
rue, index_label=None, chunksize=None, dtype=None)
```

In [15]:

```
positive_data=pd.read_sql_query("""select * from total where score='positive'""",conn)
negative_data=pd.read_sql_query("""select * from total where score='negative'""",conn)
```

In [16]:

```
print(positive_data.shape)
print(negative_data.shape)
```

```
(306916, 12)
(57084, 12)
```

In [90]:

```
positive_data2000=positive_data.head(20000)
negative_data2000=negative_data.head(20000)
bigdata = positive_data2000.append(negative_data2000, ignore_index=True)
print(bigdata.shape)
```

```
(40000, 12)
```

In [91]:

```
sorted_data=bigdata
```

In [92]:

```
du=sorted_data.sample(40000)
```

In [167]:

```
du.head(15)
```

Out[167]:

	index	Id	ProductId	UserId	ProfileName	Helpfuln
24972	346041	374343	B00004CI84	A1B2IZU1JLZA6	Wes	19
8389	346094	374400	B00004CI84	A2DEE7F9XKP3ZR	jerome	0
14571	346030	374332	B00004CI84	AEPJYN0NAX9N4	Jody L. Schoth	0
14123	346113	374419	B00004CI84	ADIDQRLLR4KBQ	"paradise_found"	2
3744	226060	245108	B001O8NLV2	A356HBGSVZ5NRH	B.P. "tilley_traveler"	14
33850	346040	374342	B00004CI84	A10L8O1ZMUIMR2	G. Kleinschmidt	61
30321	346095	374401	B00004CI84	A3M5O6UHXO9IBU	Gary	2
28857	388413	419994	B0000A0BS5	A238V1XTSK9NFE	Andrew Lynn	46
36494	121056	131233	B00004RAMX	A1PYZPS1QYR036	Kazantzakis "hinterlands"	5

	index	Id	ProductId	UserId	ProfileName	Helpfuln
13005	179643	194858	B0000E65WB	A2VZ11U5DXM8J5	C. Ebeling "ctlpareader"	1
11496	401776	434425	B0000CA4TK	A5VIGE8EO86RI	captmorgan1670 "captmorgan1670"	4
21944	428912	463849	B0000SXEKA	A2801SG8XA9LNX	PACW	7
14552	292866	317257	B0000DIYUK	A2ZM9BGE3K3SY2	Sara Swihart	0
8937	24061	26313	B000121BY6	A281NPSIMI1C2R	Rebecca of Amazon "The Rebecca Review"	9
18597	477821	516699	B0000DG87B	AF5EKQ4I9NHJ4	Smitty Peete	1

In [93]:

```
#Again sorting our data in Ascending order
du=du.sort_values('Time',axis=0,ascending=True,inplace=False,kind='quicksort',na_positi
on='last')
```

In [94]:

```
s1=du['Score']
```

In [100]:

```
X_1, X_test, y_1, y_test = cross_validation.train_test_split(du, s1, test_size=0.3, ran
dom_state=0)

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)
```

In [102]:

```
#BOW for train points
count_vect = CountVectorizer(max_features=23588) #in scikit-learn
X_1 = count_vect.fit_transform(X_1['Text'].values)
print(X_1.shape)
```

```
(28000, 23588)
```

In [101]:

```
#BOW for CV points
count_vect = CountVectorizer()
X_test = count_vect.fit_transform(X_test['Text'].values)
print(X_test.shape)
```

```
(12000, 23588)
```

In [103]:

```
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import StandardScaler
standardizedtest_data = StandardScaler(with_mean=False).fit_transform(X_1)
print(standardizedtest_data.shape)
X_1=standardizedtest_data
```

```
(28000, 23588)

C:\Users\Yaakuza\Anaconda3\lib\site-packages\sklearn\utils\validation.py:4
75: DataConversionWarning: Data with input dtype int64 was converted to fl
oat64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [104]:

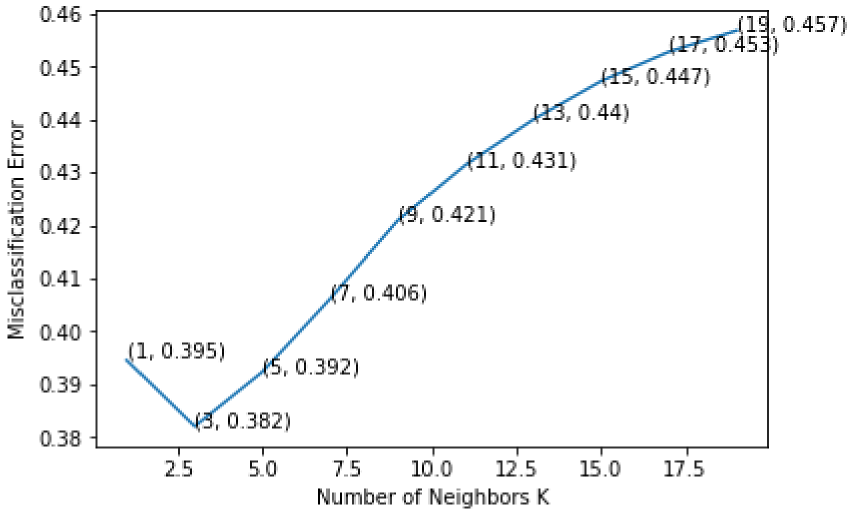
```
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import StandardScaler
standardizedtest_data = StandardScaler(with_mean=False).fit_transform(X_test)
print(standardizedtest_data.shape)
X_test=standardizedtest_data
```

```
(12000, 23588)

C:\Users\Yaakuza\Anaconda3\lib\site-packages\sklearn\utils\validation.py:4
75: DataConversionWarning: Data with input dtype int64 was converted to fl
oat64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [105]:

```
# creating odd list of K for KNN
myList = list(range(0,21))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

# empty list that will hold cv scores
cv_scores = []

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_1, y_1, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_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.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(MSE,3))
```

```
The optimal number of neighbors is 3.
```




```
the misclassification error for each k value is :  [0.395 0.382 0.392 0.40
6 0.421 0.431 0.44  0.447 0.453 0.457]
```

In [108]:

```
# ============================== KNN with k = optimal_k ===============================
================
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=3)

# fitting the model
knn_optimal.fit(X_1, y_1)

# predict the response
pred = knn_optimal.predict(X_test)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n Accuracy for Optimal k = %d is %d%%' % (optimal_k, acc))
```

```
 Accuracy for Optimal k = 3 is 51%
```

Confusion matrix , Precision, Recall, F-Score

In [109]:

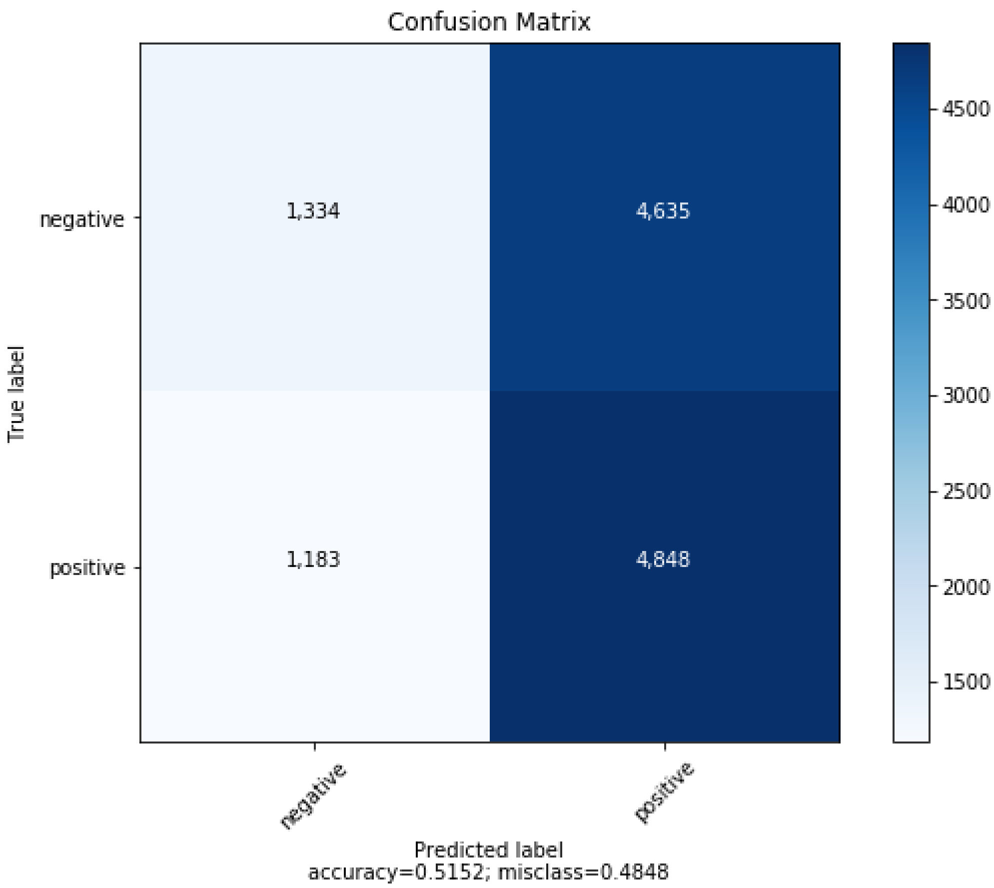
```
import numpy as np


def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    """
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    ---------
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from matplotlib.pyplot.cm
                  see http://matplotlib.org/examples/color/colormaps_reference.html
                  plt.get_cmap('jet') or plt.cm.Blues

    normalize:    If False, plot the raw numbers
                  If True, plot the proportions

    Usage
    -----
    plot_confusion_matrix(cm           = cm,                  # confusion matrix create
d by
                                                              # sklearn.metrics.confusi
on_matrix
                          normalize    = True,                # show proportions
                          target_names = y_labels_vals,       # list of names of the cl
asses
                          title        = best_estimator_name) # title of graph

    Citiation
    ---------
    http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.
html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
```

```
    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, m
isclass))
    plt.show()
```

In [111]:

```
# print the confusion matrix
from sklearn.metrics import confusion_matrix
from sklearn import metrics
gb=metrics.confusion_matrix(y_test,pred)
print(gb)
plot_confusion_matrix(cm           = np.array([[ 1334  ,4635],[1183  ,4848]]),
                      normalize    = False,
                      target_names = ['negative', 'positive'],
                      title        = "Confusion Matrix")
```

```
[[1334 4635]
 [1183 4848]]
```



In [112]:

```
#Recall From above Confusion Metric
recall=(gb[1,1]+0.0)/sum(gb[1,:])
recall
```

Out[112]:

```
0.8038467915768529
```

In [113]:

```
#precision From above Confusion Metric
pre=(gb[1,1]+0.0)/sum(gb[:,1])
print(pre)
```

```
0.5112306232204998
```

In [114]:

```
# caculating F1 Score By using HP i.e
#F1=2*TP/2*TP+FP+FN
F1=(2*pre*recall)/(pre+recall)
F1
```

Out[114]:

```
0.6249838855227536
```

Now Doing this same process with TF-idf vectors

In [143]:

```
X_1, X_test, y_1, y_test = cross_validation.train_test_split(du, s1, test_size=0.3, ran
dom_state=0)

# split the train data set into cross validation train and cross validation test
X_tr, X_cv, y_tr, y_cv = cross_validation.train_test_split(X_1, y_1, test_size=0.3)
```

In [144]:

```
#Now we Use TF-IDF vectors to predict our reviews using Naive Bayes
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),max_features=320494)
new1 = tf_idf_vect.fit_transform(X_1['Text'].values)
new1.get_shape()
```

Out[144]:

```
(28000, 320494)
```

In [145]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
new2 = tf_idf_vect.fit_transform(X_test['Text'].values)
new2.get_shape()
```

Out[145]:

```
(12000, 320494)
```

Standardizing our Train and Test TF-IDF vectors

In [146]:

```
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import StandardScaler
new11 = StandardScaler(with_mean=False).fit_transform(new1)
```

In [150]:

```
#from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import StandardScaler
new22 = StandardScaler(with_mean=False).fit_transform(new2)
```

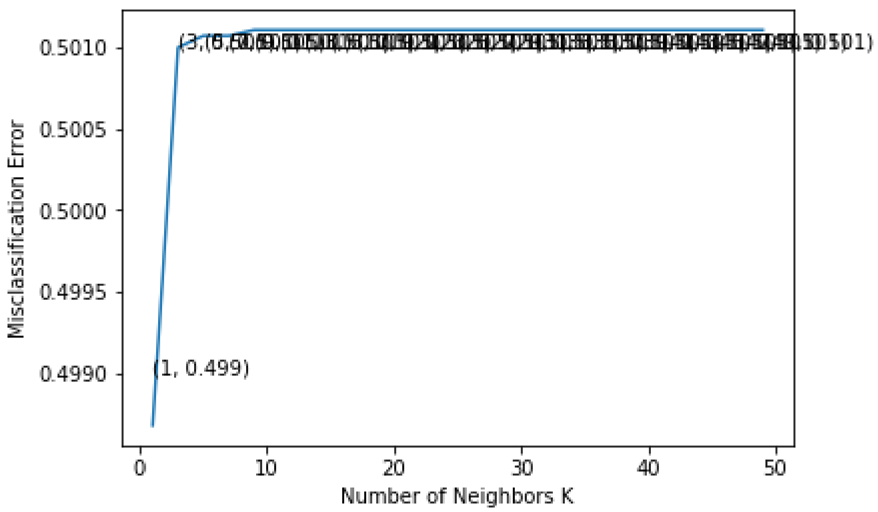
Using KNN to train for Different values of k using the 10 fold cross validation

Now we are applying 10 fold cross validation to our Train dataset and then choosing the best value of k to find out our final accuracy on our test data set

In [148]:

```
# creating odd list of K for KNN
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

# empty list that will hold cv scores
cv_scores = []

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, new11, y_1, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())

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

# determining best k
c = 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.xlabel('Number of Neighbors K')
plt.ylabel('Misclassification Error')
plt.show()

print("the misclassification error for each k value is : ", np.round(MSE,3))
```

```
The optimal number of neighbors is 1.
```




```
the misclassification error for each k value is :  [0.499 0.501 0.501 0.50
1 0.501 0.501 0.501 0.501 0.501 0.501 0.501 0.501
 0.501 0.501 0.501 0.501 0.501 0.501 0.501 0.501 0.501 0.501 0.501 0.501
 0.501]
```

In [159]:

```
# ============================== KNN with k = optimal_k ===============================
================
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(new11, y_1)

# predict the response
pred = knn_optimal.predict(new22)

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

```
The accuracy of the knn classifier for k = 1 is 50.225000%
```

In [160]:

```
y_test.describe()
```

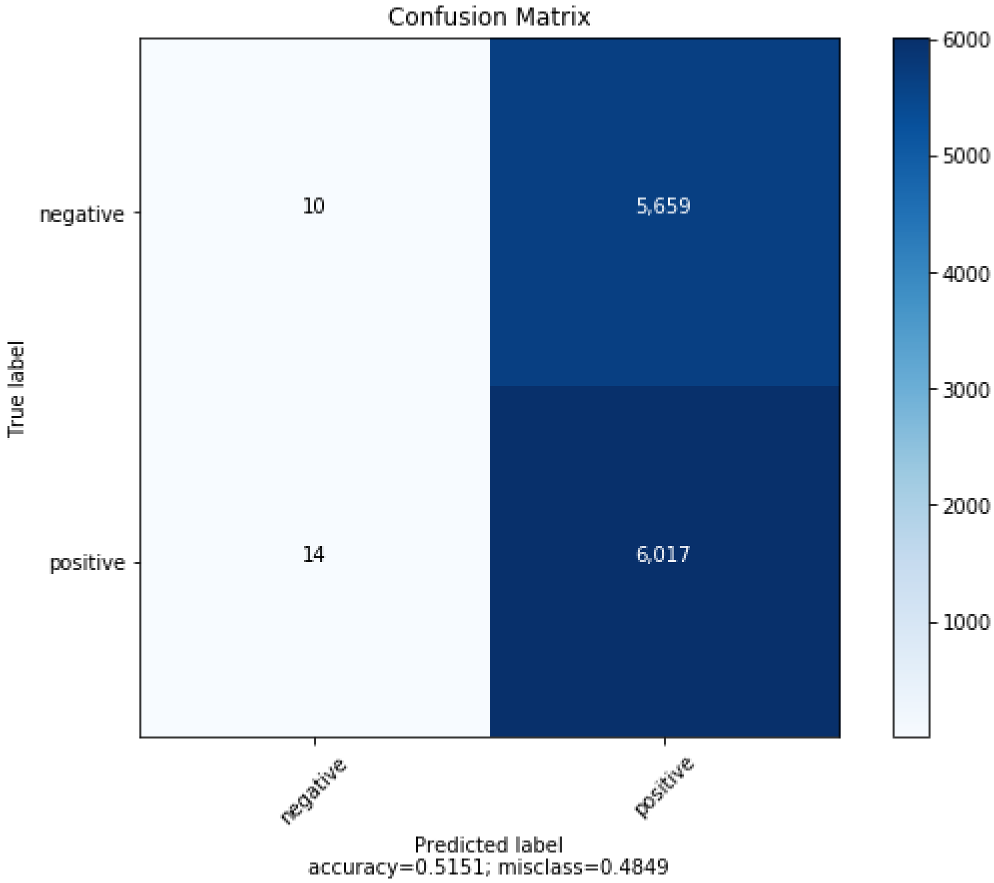
Out[160]:

```
count        12000
unique           2
top       positive
freq          6031
Name: Score, dtype: object
```

In [161]:

```
# print the confusion matrix
from sklearn.metrics import confusion_matrix
from sklearn import metrics
gb=metrics.confusion_matrix(y_test,pred)
print(gb)
#plotting the confusion matrix
#Plot of Confusion Metric
plot_confusion_matrix(cm           = np.array([[ 10   ,5659],[14   ,6017]]),
                      normalize    = False,
                      target_names = ['negative', 'positive'],
                      title        = "Confusion Matrix")
```

```
[[  10 5959]
 [  14 6017]]
```



In [162]:

```
#Recall From above Confusion Metric
recall=(gb[1,1]+0.0)/sum(gb[1,:])
recall
```

Out[162]:

```
0.9976786602553473
```

In [163]:

```
#precision From above Confusion Metric
pre=(gb[1,1]+0.0)/sum(gb[:,1])
print(pre)
```

```
0.5024215096860387
```

In [164]:

```
# caculating F1 Score By using HP i.e
#F1=2*TP/2*TP+FP+FN
F1=(2*pre*recall)/(pre+recall)
F1
```

Out[164]:

```
0.6682956627978008
```

Conclusion / Summary

(i) Sampled 40k reviews from our Dataset. (ii) Then dividing our reviews into train and test. (iii) Converting the text of reviews into vectors using both BOW and TD-idf Vectoriser. (iv) Applying 10 Fold Cross Validation to our Train dataset and finding the optimum value of k, using KNN. (v) Computing the Accuracy on our test dataset using the optimal value of K. (vi) Also finding Confusion Matrix , Precision, Recall, F-Score.

Model :- K nearest Neighbour HyperParameter:- K

1.FOR BOW

```
Optimal K:- 3
Train Error:- 39 %
Test Accuracy :-51%
F1 Score :- 0.62
```

2.FOR TF-IDF

```
Optimal K:- 1
Train Error:- 49.9 %
Test Accuracy :-50%
F1 Score :- 0.56
```

For TF-IDF we are slightly getting lower accuracy on our test dataset, but it's still very bad. KNN might not be a good model for this problem. We will use different models in future such as Naive Bayes and Logistic Regression that would improve the accuracy.