In [2]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
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
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import mean_squared_error
```

```
C:\Users\Soni\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: Depre
cationWarning: This module was deprecated in version 0.18 in favor of the model
_selection module into which all the refactored classes and functions are move
d. Also note that the interface of the new CV iterators are different from that
of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [3]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re

import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, neighbors
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 itertools
from sklearn import svm, datasets
from sklearn.metrics import f1_score
```

```
C:\Users\Soni\Anaconda3\lib\site-packages\gensim\utils.py:1212: UserWarning: de
tected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [4]:
```
conn = sqlite3.connect('final.sqlite')
final = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 """, conn)
```

In [4]:
```
final.head(5)
```

Out[4]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfulnes
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
1	138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
2	138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
3	138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
4	138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	

In [5]:
```
final = final.sort_values('Time', axis=0, ascending=True, inplace=False, kind='qu
```

In [6]:
```
final.head()
```

Out[6]:

	index	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuln
0	138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
30	138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	
424	417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	
330	346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1	
423	417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0	

In [7]:
```
final.shape
```

Out[7]:
```
(364171, 12)
```

In [12]:
```
x = final['CleanedText'].values[0:60000]
y = final['Score'].values[0:60000]
```

In [13]:
```
from sklearn.model_selection import train_test_split
from sklearn import cross_validation
x_1, x_test, y_1, y_test = cross_validation.train_test_split(x, y, test_size=0.2,

# split the train data set into cross validation train and cross validation test
x_train, x_cv, y_train, y_cv = cross_validation.train_test_split(x_1, y_1, train_
```

In [7]:
```
def knn_brute(x_train,x_cv,y_train,y_cv):

    myList = list(range(3,91,2))
    neighbors = list(filter(lambda x: x % 2 != 0, myList))

    cv_scores = []

    for i in neighbors:

        knn = KNeighborsClassifier(n_neighbors=i, algorithm = 'brute')
        scores = cross_val_score(knn, x_train, y_train, scoring='accuracy')
        cv_scores.append(scores.mean())

     # fitting the model
    knn.fit(x_train, y_train)

    # predict the response
    pred = knn.predict(x_cv)
    MSE = [1 - x for x in cv_scores]

# determining best k
    optimal_k = neighbors[MSE.index(min(MSE))]
    print('\nThe optimal no. of k is %d.' % optimal_k)

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

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



    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()

    return optimal_k
```

In [11]:
```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer() #Vectorization for BOW
X_train = count_vect.fit_transform(x_train)# Fitting and training our dataset on
X_cv = count_vect.transform(x_cv)
X_test = count_vect.transform(x_test)
print("Train Data Size: ",X_train.shape)
print("CV Data Size: ",X_cv.shape)
print("Test Data Size: ",X_test.shape)
```

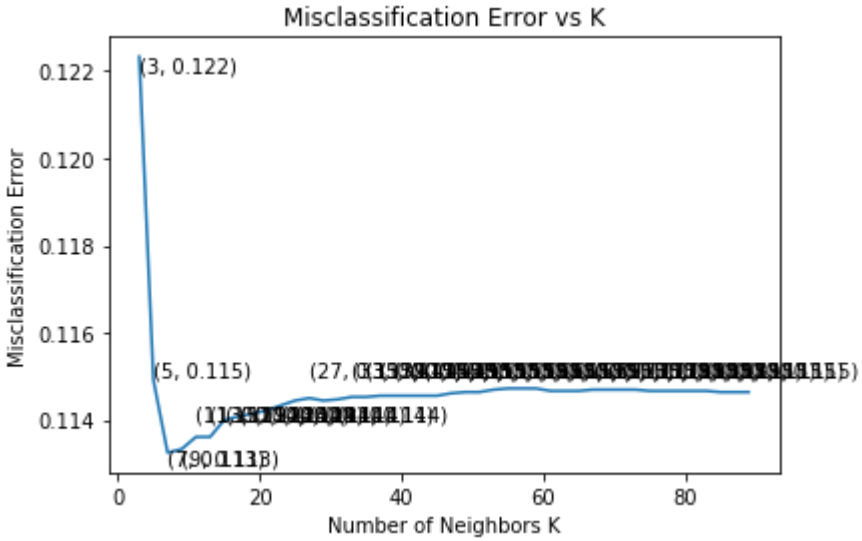
```
Train Data Size:  (36000, 23194)
CV Data Size:  (12000, 23194)
Test Data Size:  (12000, 23194)
```

In [12]:
```
print("the type of count vectorizer ",type(X_train))
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

In [13]:
```
import warnings
warnings.filterwarnings("ignore")
optimal_k = knn_brute(X_train,X_cv,y_train,y_cv)
optimal_k
```

```
The optimal no. of k is 7.

CV accuracy for k = 89 is 88%
the misclassification error for each k value is :  [0.122 0.115 0.113 0.113 0.1
14 0.114 0.114 0.114 0.114 0.114 0.114 0.114
 0.115 0.114 0.114 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115
 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115
 0.115 0.115 0.115 0.115 0.115 0.115 0.115 0.115]
```



Out[13]: 7

In [14]:
```
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(X_train, y_train)

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

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

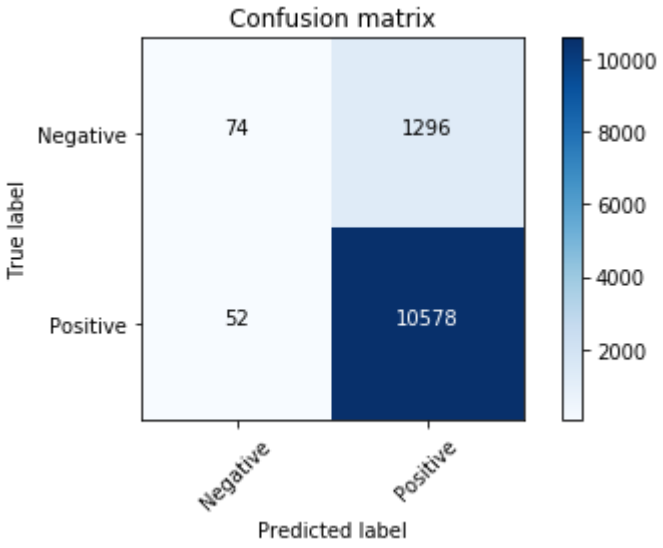
```
The accuracy of the knn classifier for k = 7 is 88.766667%
```

In [15]:
```
err_bow = 100-acc_bow
err_bow
```

Out[15]:
```
11.233333333333334
```

In [20]:
```
#Plot Confusion Matrix
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
     #else:
      #  print('Confusion matrix, without normalization')

    #print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

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

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [17]:
```
plot_confusion_matrix(confusion_matrix(y_test, pred), classes=["Negative","Positi
```



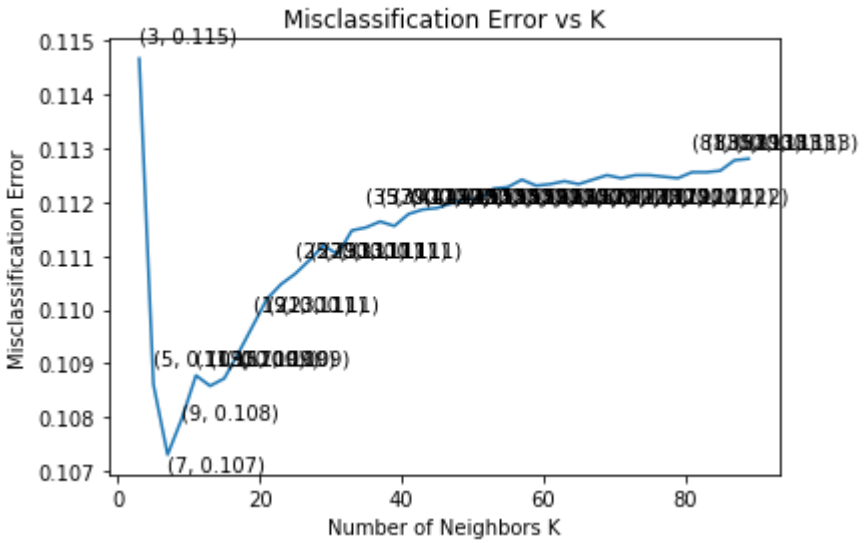
TFIDF

In [35]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))#Vectorizering the data
X_train_tfidf = tf_idf_vect.fit_transform(x_train)
X_cv_tfidf = tf_idf_vect.transform(x_cv)
X_test_tfidf = tf_idf_vect.transform(x_test)
print("the type of count vectorizer ",type(X_train_tfidf))
print("Train Data Size: ",X_train_tfidf.shape)
print("CV Data Size: ",X_cv_tfidf.shape)
print("Test Data Size: ",X_test_tfidf.shape)
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
Train Data Size:  (36000, 609794)
CV Data Size:  (12000, 609794)
Test Data Size:  (12000, 609794)
```

In [19]:
```
import warnings
warnings.filterwarnings("ignore")
optimal_k = knn_brute(X_train_tfidf,X_cv_tfidf,y_train,y_cv)
optimal_k
```

```
The optimal no. of k is 7.

CV accuracy for k = 89 is 88%
the misclassification error for each k value is :  [0.115 0.109 0.107 0.108 0.1
09 0.109 0.109 0.109 0.11  0.11  0.11  0.111
 0.111 0.111 0.111 0.111 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112
 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112
 0.112 0.112 0.112 0.113 0.113 0.113 0.113 0.113]
```



Out[19]: 7

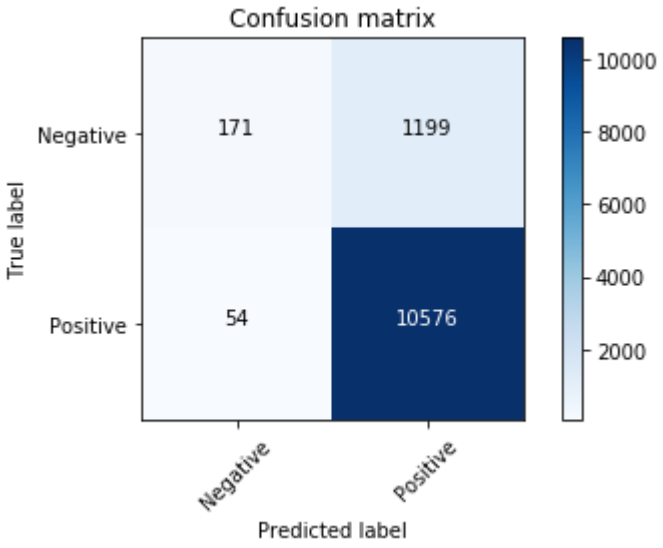
In [20]:
```
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(X_train_tfidf, y_train)

# predict the response
pred_tfidf = knn_optimal.predict(X_test_tfidf)

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

```
The accuracy of the knn classifier for k = 7 is 89.558333%
```

In [21]:
```
err_bow = 100-acc_bow
err_bow
```

Out[21]: 10.441666666666677

In [22]:
```
plot_confusion_matrix(confusion_matrix(y_test, pred_tfidf), classes=["Negative","
```



AVG W2Vec

In [14]:
```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import os
```

In [15]:
```
#Word 2 Vector for train corpus

list_of_sent_train=[]
for sent in x_train:
    list_of_sent_train.append(sent.split())

w2v_model_train=Word2Vec(list_of_sent_train,min_count=5,size=50, workers=4)
print(w2v_model_train)

w2v_words_train = list(w2v_model_train.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_train))
print("sample words ", w2v_words_train[0:50])
```

```
Word2Vec(vocab=8245, size=50, alpha=0.025)
number of words that occured minimum 5 times  8245
sample words  ['way', 'beyond', 'toler', 'level', 'peopl', 'room', 'temperatu
r', 'palat', 'like', 'hot', 'flavor', 'one', 'buy', 'order', 'near', 'futur',
'purchas', 'bone', 'month', 'old', 'black', 'lab', 'love', 'gave', 'kept', 'bus
i', 'long', 'pay', 'next', 'day', 'diarhea', 'pure', 'water', 'type', 'read',
'review', 'would', 'known', 'better', 'drink', 'choic', 'addict', 'cup', 'simpl
i', 'noth', 'els', 'tast', 'good', 'ive', 'tri']
```

In [16]:
```
list_of_sent_cv=[]
for sent in x_cv:
    list_of_sent_cv.append(sent.split())

w2v_model_cv=Word2Vec(list_of_sent_cv,min_count=5,size=50, workers=4)
print(w2v_model_cv)

w2v_words_cv = list(w2v_model_cv.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_cv))
print("sample words ", w2v_words_cv[0:50])
```

```
Word2Vec(vocab=5032, size=50, alpha=0.025)
number of words that occured minimum 5 times  5032
sample words  ['vibrant', 'awesom', 'color', 'never', 'buy', 'food', 'throw',
'away', 'other', 'add', 'littl', 'sour', 'cream', 'done', 'someth', 'swear', 'h
ome', 'made', 'great', 'product', 'compact', 'salt', 'kit', 'real', 'winner',
'metal', 'box', 'neat', 'construct', 'make', 'easi', 'see', 'insid', 'weve', 'g
iven', 'sever', 'collect', 'gift', 'alway', 'gotten', 'rave', 'review', 'fact',
'bought', 'one', 'use', 'travel', 'love', 'quick', 'classic']
```

In [17]:
```
#Word 2 Vector for test corpus
list_of_sent_test=[]
for sent in x_test:
    list_of_sent_test.append(sent.split())

w2v_model_test=Word2Vec(list_of_sent_test,min_count=5,size=50, workers=4)
print(w2v_model_test)

w2v_words_test = list(w2v_model_test.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_test))
print("sample words ", w2v_words_test[0:50])
```

```
Word2Vec(vocab=4974, size=50, alpha=0.025)
number of words that occured minimum 5 times  4974
sample words  ['seem', 'littl', 'suspici', 'ingredi', 'list', 'deceiv', 'aspart
am', 'ginger', 'bodi', 'extrem', 'toxic', 'sensit', 'consum', 'poison', 'woul
d', 'steer', 'clear', 'got', 'box', 'correct', 'old', 'fashion', 'grit', 'tak
e', 'long', 'time', 'cook', 'color', 'blue', 'instead', 'red', 'snackmast', 'sa
lmon', 'jerki', 'nice', 'purchas', 'nation', 'health', 'food', 'store', 'someti
m', 'hard', 'packag', 'deliv', 'amazon', 'perfect', 'soft', 'right', 'tast', 'f
ishi']
```

In [18]:
```
#train corpus
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this
for sent in list_of_sent_train: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train:
            #print(word)
            vec = w2v_model_train.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train.append(sent_vec)
print(len(sent_vectors_train))
print(len(sent_vectors_train[0]))
```

```
36000
50
```

In [19]:
```
sent_vectors_cv = []; # the avg-w2v for each sentence/review is stored in this li
for sent in list_of_sent_cv: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_cv:
            #print(word)
            vec = w2v_model_cv.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv.append(sent_vec)
print(len(sent_vectors_cv))
print(len(sent_vectors_cv[0]))
```

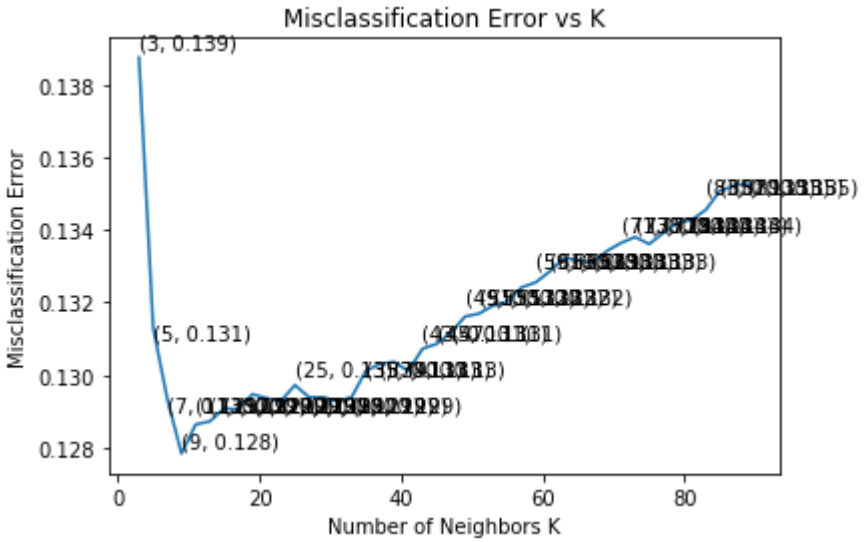
```
12000
50
```

In [20]:
```
#test Corpus
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this
for sent in list_of_sent_test: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_test:

            vec = w2v_model_test.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[0]))
```

```
12000
50
```

In [21]:
```
optimal_k = knn_brute(sent_vectors_train,sent_vectors_cv,y_train,y_cv)
```

```
The optimal no. of k is 9.

CV accuracy for k = 89 is 85%
the misclassification error for each k value is :  [0.139 0.131 0.129 0.128 0.1
29 0.129 0.129 0.129 0.129 0.129 0.129 0.13
 0.129 0.129 0.129 0.129 0.13  0.13  0.13  0.13  0.131 0.131 0.131 0.132
 0.132 0.132 0.132 0.132 0.133 0.133 0.133 0.133 0.133 0.133 0.134 0.134
 0.134 0.134 0.134 0.134 0.135 0.135 0.135 0.135]
```



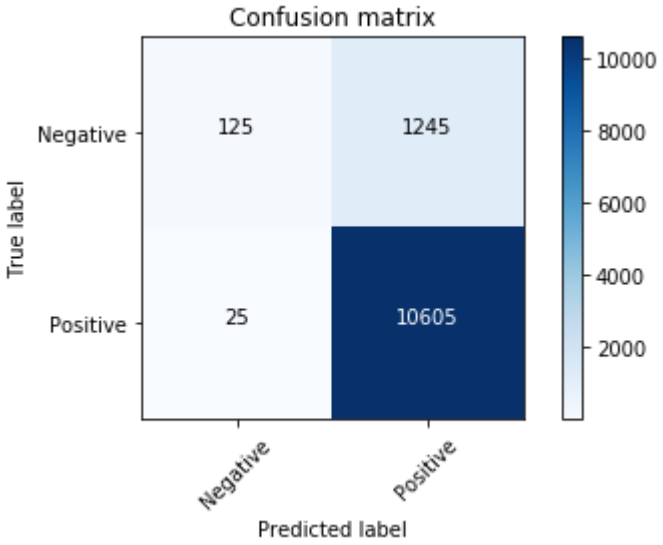
In [36]:
```
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(X_train_tfidf, y_train)

# predict the response
pred_avgw2v = knn_optimal.predict(X_test_tfidf)

# evaluate accuracy
acc_bow = accuracy_score(y_test, pred_avgw2v) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc
```

```
The accuracy of the knn classifier for k = 7 is 86.600000%
```

In [32]:
```
err_bow = 100-acc_bow
err_bow
```

Out[32]:
```
10.583333333333329
```

In [33]:
```
plot_confusion_matrix(confusion_matrix(y_test, pred_avgw2v), classes=["Negative",
```



AVG TFIDF W2V

In [34]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [35]:
```
tfidf_feat = tf_idf_vect.get_feature_names()#getting feature list
```

In [36]:
```
from tqdm import tqdm
tfidf_sent_vectors_train = []; # the tfidf-w2v for each sentence/review is stored
row=0;
for sent in tqdm(list_of_sent_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train:
            vec = w2v_model_train.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))#taking the dic
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_train.append(sent_vec)
    row += 1

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

```
100%|██████████████████████████████████████████████████████████████████████████
| 36000/36000 [00:45<00:00, 793.73it/s]

36000
50
```

In [37]:
```
from tqdm import tqdm
tfidf_sent_vectors_cv = []; # the tfidf-w2v for each sentence/review is stored in
row=0;
for sent in tqdm(list_of_sent_cv): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_cv:
            vec = w2v_model_cv.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))#taking the dic
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_cv.append(sent_vec)
    row += 1

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

```
100%|██████████████████████████████████████████████████████████████████████████████
| 12000/12000 [00:11<00:00, 1003.89it/s]

12000
50
```

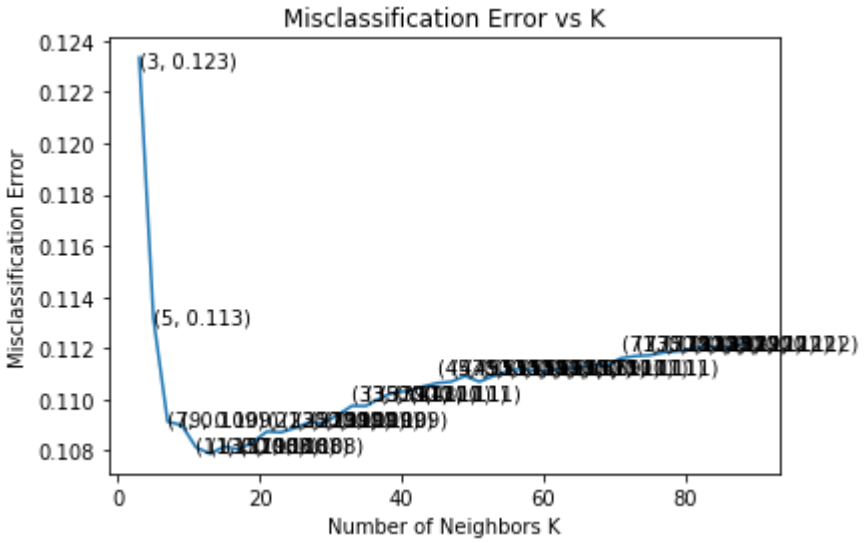
In [38]:
```
model = TfidfVectorizer(ngram_range=(1,2))
tf_idf_matrix = model.fit_transform(final['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_test = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [39]:
```
#test corpus
tfidf_sent_vectors_test = []; # the tfidf-w2v for each sentence/review is stored
row=0;
for sent in tqdm(list_of_sent_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_test:
            vec = w2v_model_test.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_test[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test.append(sent_vec)
    row += 1

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

```
100%|██████████████████████████████████████████████████████████████████████████████
| 12000/12000 [00:11<00:00, 1051.34it/s]

12000
50
```

In [40]:
```
optimal_k = knn_brute(tfidf_sent_vectors_train,tfidf_sent_vectors_cv,y_train,y_cv
```

```
The optimal no. of k is 13.

CV accuracy for k = 89 is 88%
the misclassification error for each k value is :  [0.123 0.113 0.109 0.109 0.1
08 0.108 0.108 0.108 0.108 0.109 0.109 0.109
 0.109 0.109 0.109 0.11  0.11  0.11  0.11  0.11  0.11  0.111 0.111 0.111
 0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.112 0.112
 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112]
```



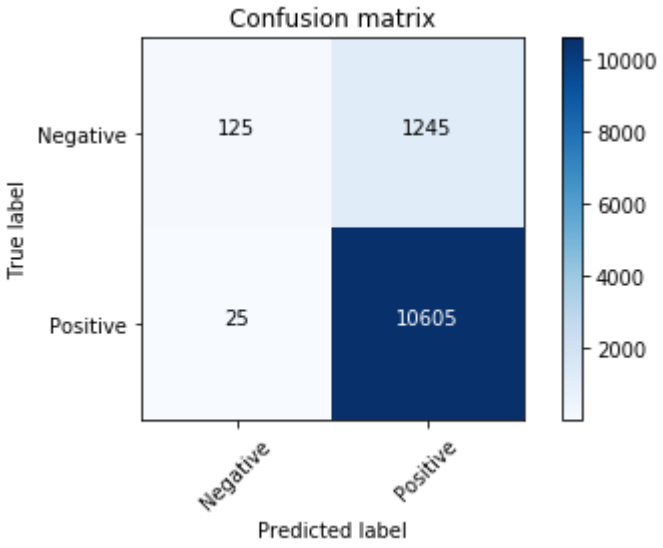
In [41]:
```
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(X_train_tfidf, y_train)

# predict the response
pred_avgw2v_tfidf = knn_optimal.predict(X_test_tfidf)

# evaluate accuracy
acc_bow = accuracy_score(y_test, pred_avgw2v_tfidf) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc
```

```
The accuracy of the knn classifier for k = 13 is 89.416667%
```

In [42]:
```
err_bow = 100-acc_bow
err_bow
```

Out[42]:
```
10.583333333333329
```

In [43]:
```
plot_confusion_matrix(confusion_matrix(y_test, pred_avgw2v_tfidf), classes=["Nega
```



KD-Tree Algorithm

Taking 50K points

In [6]:
```
x1 = final['CleanedText'].values[0:60000]
y1 = final['Score'].values[0:60000]
```

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn import cross_validation
x_1, x_test_kd, y_1, y_test_kd = cross_validation.train_test_split(x1, y1, test_s

# split the train data set into cross validation train and cross validation test
x_train_kd, x_cv_kd, y_train_kd, y_cv_kd = cross_validation.train_test_split(x_1,
```

In [8]:
```
def knn_kd(x_train_kd,x_cv_kd,y_train_kd,y_cv_kd):

    myList = list(range(3,91,2))
    neighbors = list(filter(lambda x: x % 2 != 0, myList))

    cv_scores = []

    for i in neighbors:

        knn = KNeighborsClassifier(n_neighbors=i, algorithm = 'kd_tree')
        scores = cross_val_score(knn, x_train_kd, y_train_kd, scoring='accuracy')
        cv_scores.append(scores.mean())

     # fitting the model
    knn.fit(x_train_kd, y_train_kd)

    # predict the response
    pred = knn.predict(x_cv_kd)
    MSE = [1 - x for x in cv_scores]

# determining best k
    optimal_k = neighbors[MSE.index(min(MSE))]
    print('\nThe optimal no. of k is %d.' % optimal_k)

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

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



    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()

    return optimal_k
```

In [8]:
```
count_vect = CountVectorizer()
X_train_kd = count_vect.fit_transform(x_train_kd)
X_cv_kd = count_vect.transform(x_cv_kd)
X_test_kd = count_vect.transform(x_test_kd)
print("Train Data Size: ",X_train_kd.shape)
print("Test Data Size: ",X_test_kd.shape)
```

```
Train Data Size:  (36000, 23005)
Test Data Size:  (12000, 23005)
```

In [9]:
```
print("the type of count vectorizer ",type(X_train_kd))
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
```

In [14]:
```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components = 150)
X_train_svd = svd.fit_transform(X_train_kd)
X_cv_svd = svd.transform(X_cv_kd)
X_test_svd = svd.transform(X_test_kd)
```

In [11]:
```
print("the type of count vectorizer ",type(X_train_svd))
```

```
the type of count vectorizer  <class 'numpy.ndarray'>
```

In [12]:
```
print (X_test_svd.size)
```

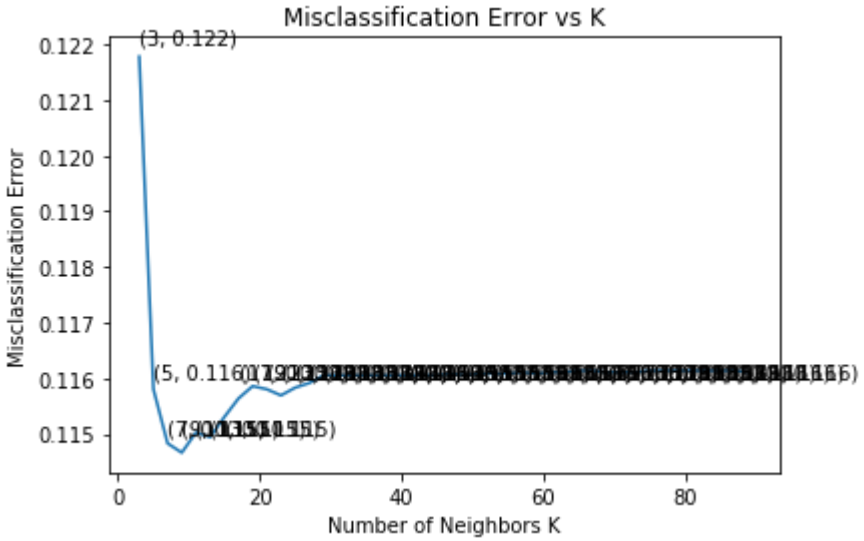
```
1800000
```

In [13]:
```
print (X_train_svd.size)
```

```
5400000
```

In [15]:
```
import warnings
warnings.filterwarnings("ignore")
optimal_k = knn_kd(X_train_kd,X_cv_kd, y_train_kd,y_cv_kd)
```

```
The optimal no. of k is 9.

CV accuracy for k = 89 is 89%
the misclassification error for each k value is :  [0.122 0.116 0.115 0.115 0.1
15 0.115 0.115 0.116 0.116 0.116 0.116 0.116
 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116
 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116
 0.116 0.116 0.116 0.116 0.116 0.116 0.116 0.116]
```



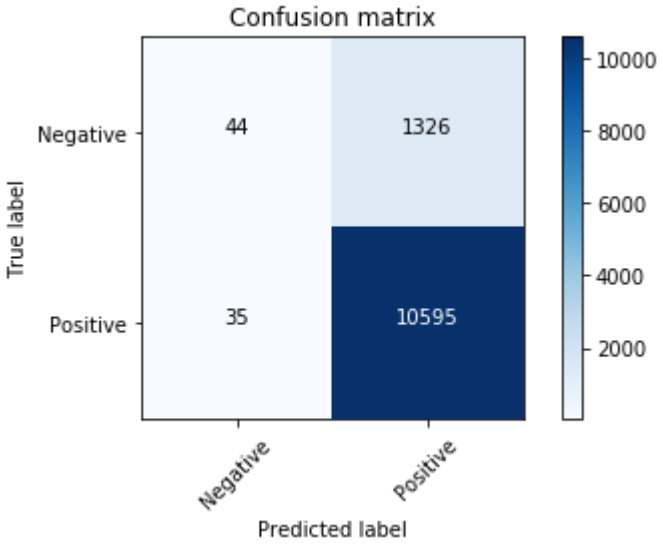
In [17]:
```
# ============================== KNN with k = optimal_k ========================
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(X_train_kd, y_train_kd)

# predict the response
pred_kd = knn_optimal.predict(X_test_kd)

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

```
The accuracy of the knn classifier for k = 9 is 88.658333%
```

In [18]:
```
err_bow = 100-acc_bow
err_bow
```

Out[18]:
```
11.341666666666669
```

In [22]:
```
plot_confusion_matrix(confusion_matrix(y_test_kd, pred_kd), classes=["Negative","
```



TF-IDF

In [9]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
#knn_tfidf_matrix = tf_idf_vect.fit_transform(review_data_sort['CleanedText'].val
X_train_tfidf_kd = tf_idf_vect.fit_transform(x_train_kd)
X_cv_tfidf_kd = tf_idf_vect.transform(x_cv_kd)
X_test_tfidf_kd = tf_idf_vect.transform(x_test_kd)
print("the type of count vectorizer ",type(X_train_tfidf_kd))
print("Train Data Size: ",X_train_tfidf_kd.shape)
print("Test Data Size: ",X_test_tfidf_kd.shape)
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
Train Data Size:  (36000, 569788)
Test Data Size:  (12000, 569788)
```

In [10]:
```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components = 210)
X_train_tfidf_svd = svd.fit_transform(X_train_tfidf_kd)
X_cv_tfidf_svd = svd.transform(X_cv_tfidf_kd)
X_test_tfidf_svd = svd.transform(X_test_tfidf_kd)
```

In [28]:
```
print("the type of count vectorizer ",type(X_train_tfidf_svd))
```

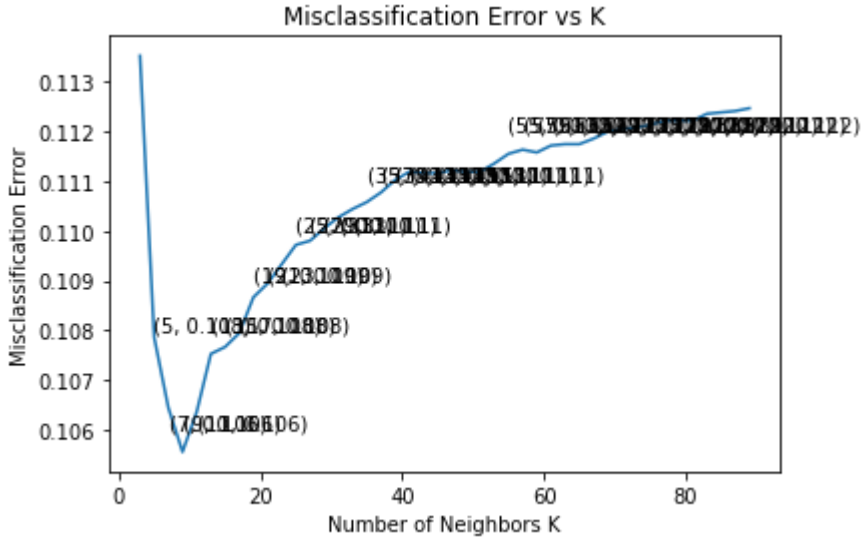
```
the type of count vectorizer  <class 'numpy.ndarray'>
```

In [29]:
```
print (X_test_tfidf_svd.size)
```

```
2520000
```

In [10]:
```
import warnings
warnings.filterwarnings("ignore")
optimal_k = knn_kd(X_train_tfidf_kd,X_cv_tfidf_kd, y_train_kd,y_cv_kd)
```

```
The optimal no. of k is 9.

CV accuracy for k = 89 is 88%
the misclassification error for each k value is :  [0.114 0.108 0.106 0.106 0.1
06 0.108 0.108 0.108 0.109 0.109 0.109 0.11
 0.11  0.11  0.11  0.11  0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111
 0.111 0.111 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112
 0.112 0.112 0.112 0.112 0.112 0.112 0.112 0.112]
```



In [11]:
```
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(X_train_tfidf_kd, y_train_kd)

# predict the response
pred_kd_tfidf = knn_optimal.predict(X_test_tfidf_kd)

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-ec456940d73c> in <module>()
      8
      9 # evaluate accuracy
---> 10 acc_bow = accuracy_score(y_test, pred_kd_tfidf) * 100
     11 print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (opti
mal_k, acc_bow))

NameError: name 'y_test' is not defined
```

In [ ]:
```
err_bow = 100-acc_bow
err_bow
```

In [ ]:
```
plot_confusion_matrix(confusion_matrix(y_test_kd, pred_kd_tfidf), classes=["Negat
```

Word2Vec KD-Tree

In [16]:
```
#Word 2 Vector for train corpus

list_of_sent_train_kd=[]
for sent in x_train_kd:
    list_of_sent_train_kd.append(sent.split())

w2v_model_train_kd=Word2Vec(list_of_sent_train_kd,min_count=5,size=300, workers=4
print(w2v_model_train_kd)

w2v_words_train_kd = list(w2v_model_train_kd.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_train_kd))
print("sample words ", w2v_words_train_kd[0:300])
```

```
Word2Vec(vocab=8035, size=300, alpha=0.025)
number of words that occured minimum 5 times  8035
sample words  ['coffe', 'dark', 'rich', 'use', 'keurig', 'pod', 'system', 'stro
ng', 'brew', 'good', 'cafe', 'seattl', 'realli', 'blend', 'machin', 'bought',
'varieti', 'grandchildren', 'start', 'christma', 'heard', 'expens', 'gift', 'ki
d', 'still', 'stuck', 'mani', 'childhood', 'pattern', 'someth', 'silli', 'fun',
'tast', 'turn', 'hit', 'season', 'best', 'clam', 'chowder', 'mix', 'found', 'qu
ick', 'easi', 'great', 'even', 'order', 'came', 'expect', 'experi', 'coconut',
'sweet', 'moist', 'prefer', 'brand', 'grandmoth', 'delici', 'butteri', 'pecan',
'german', 'chocol', 'cake', 'frost', 'grandma', 'isnt', 'anymor', 'everi', 'tim
e', 'make', 'birthday', 'special', 'day', 'alway', 'think', 'stuff', 'amaz', 'j
ar', 'sauc', 'couldnt', 'recommend', 'high', 'love', 'differ', 'kind', 'nut',
'defint', 'old', 'bore', 'granola', 'bag', 'small', 'side', 'doesnt', 'last',
'long', 'wow', 'product', 'simpli', 'nice', 'textur', 'almost', 'like', 'shre
d', 'beef', 'better', 'jerki', 'opinion', 'spice', 'flavor', 'right', 'combin',
'smokey', 'salti', 'problem', 'tri', 'stop', 'eat', 'know', 'pack', 'protein',
'low', 'fat', 'calori', 'carb', 'content', 'gram', 'per', 'serv', 'seem', 'pret
ti', 'one', 'strip', 'watch', 'consumpt', 'although', 'replac', 'meal', 'two',
'primal', 'lost', 'pound', 'week', 'except', 'meat', 'altern', 'snack', 'quit',
'worri', 'jasmin', 'would', 'overpow', 'delic', 'white', 'tea', 'got', 'mello
w', 'light', 'certain', 'present', 'difficult', 'balanc', 'nail', 'huge', 'fa
n', 'impress', 'agre', 'smell', 'greatest', 'babi', 'liquid', 'form', 'give',
'powder', 'wont', 'stick', 'overal', 'easier', 'wish', 'larger', 'size', 'didn
t', 'refriger', 'freez', 'dri', 'mushroom', 'prior', 'could', 'kept', 'pantri',
'conveni', 'freezer', 'open', 'also', 'need', 'soak', 'foul', 'odor', 'husban
d', 'maker', 'shop', 'around', 'lowest', 'price', 'amazon', 'cheapest', 'gree
n', 'mountain', 'favorit', 'happi', 'purchas', 'gummi', 'cherri', 'ever', 'sant
a', 'took', 'find', 'thing', 'msg', 'surpris', 'pleas', 'son', 'year', 'bar',
'theyr', 'throw', 'diaper', 'go', 'gone', 'hour', 'hook', 'acid', 'burst', 'swi
tch', 'black', 'juic', 'beverag', 'extraordinarili', 'refresh', 'soda', 'ad',
'sugar', 'plus', 'rda', 'vitamin', 'ask', 'well', 'ill', 'tell', 'skimpi', 'oun
c', 'bare', 'rose', 'halfway', 'normal', 'drink', 'glass', 'sinc', 'corn', 'syr
up', 'sweeten', 'felt', 'heavi', 'yet', 'everyday', 'howev', 'econom', 'feasib
l', 'can', 'satisfi', 'quench', 'total', 'remind', 'brown', 'tart', 'pleasant',
'left', 'want', 'excel', 'tasti', 'cracker', 'cant', 'handl', 'gluten', 'dog',
'least', 'half', 'box', 'mayb', 'els', 'cooki', 'sure', 'tazo', 'china', 'tip',
'slight', 'stronger', 'zen', 'contain', 'natur', 'advantag', 'real']
```

In [17]:
```
list_of_sent_cv_kd=[]
for sent in x_cv_kd:
    list_of_sent_cv_kd.append(sent.split())

w2v_model_cv_kd=Word2Vec(list_of_sent_cv_kd,min_count=5,size=300, workers=4)
print(w2v_model_cv_kd)

w2v_words_cv_kd = list(w2v_model_cv_kd.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_cv_kd))
print("sample words ", w2v_words_cv_kd[0:300])
```

```
Word2Vec(vocab=4802, size=300, alpha=0.025)
number of words that occured minimum 5 times  4802
sample words  ['love', 'toffe', 'dark', 'chocol', 'cover', 'english', 'superb',
'purchas', 'kashi', 'golean', 'peanut', 'butter', 'bar', 'famili', 'tri', 'on
e', 'nobodi', 'want', 'finish', 'tast', 'way', 'much', 'like', 'power', 'didn
t', 'expect', 'candi', 'healthi', 'good', 'sat', 'kitchen', 'week', 'threw', 'r
est', 'away', 'nasti', 'wast', 'money', 'contact', 'return', 'sent', 'coupon',
'free', 'flavor', 'werent', 'better', 'would', 'rather', 'back', 'use', 'almos
t', 'year', 'subscript', 'order', 'realli', 'make', 'coffe', 'buy', 'besid', 'g
reat', 'expresso', 'glad', 'final', 'fresh', 'whole', 'rabbit', 'readi', 'righ
t', 'amazon', 'might', 'think', 'ship', 'charg', 'pricey', 'item', 'compar', 'i
ve', 'pay', 'petco', 'steal', 'especi', 'consid', 'prep', 'work', 'save', 'val
u', 'time', 'youll', 'bunni', 'onlin', 'delici', 'plump', 'none', 'hop', 'plai
n', 'parmesan', 'chees', 'sprinkl', 'boil', 'may', 'take', 'minut', 'two', 'lon
ger', 'suggest', 'box', 'sure', 'dont', 'overcook', 'rais', 'index', 'food', 'b
est', 'pancak', 'mix', 'wheat', 'eat', 'sometim', 'snack', 'syrup', 'never', 'k
now', 'absolut', 'waffl', 'celiac', 'yum', 'jack', 'mustard', 'pack', 'receiv',
'bottl', 'call', 'dave', 'told', 'care', 'came', 'forget', 'credit', 'statemen
t', 'bank', 'terribl', 'custom', 'servic', 'lost', 'client', 'italian', 'remem
b', 'childhood', 'difficult', 'find', 'obtain', 'gave', 'husband', 'licoric',
'say', 'hint', 'menthol', 'enough', 'refresh', 'disappoint', 'thrill', 'abl',
'small', 'pound', 'lot', 'enjoy', 'quit', 'awhil', 'favorit', 'chip', 'souther
n', 'california', 'live', 'texa', 'get', 'via', 'mail', 'got', 'offic', 'hook',
'ear', 'wash', 'high', 'sensit', 'allerg', 'everyth', 'coupl', 'even', 'avoid',
'prescript', 'medic', 'catch', 'soon', 'pleasant', 'smell', 'along', 'extract',
'calm', 'product', 'pet', 'add', 'tea', 'tree', 'oil', 'antibiot', 'effect', 'l
uck', 'old', 'seen', 'movi', 'bought', 'christma', 'needless', 'ever', 'sinc',
'person', 'michael', 'keaton', 'perform', 'harri', 'rock', 'also', 'savori', 'r
ice', 'mini', 'avail', 'least', 'potato', 'bigger', 'crunch', 'despis', 'everyo
n', 'textur', 'lowfat', 'regular', 'weird', 'puppi', 'chew', 'rubber', 'tug',
'pull', 'within', 'min', 'worth', 'read', 'bad', 'review', 'surpris', 'third',
'case', 'pumpkin', 'alway', 'price', 'fair', 'less', 'loos', 'form', 'brew', 'p
ot', 'leav', 'strain', 'doesnt', 'bitter', 'let', 'sit', 'long', 'smooth', 'str
ong', 'black', 'cant', 'wrong', 'decemb', 'still', 'bulk', 'bag', 'choic', 'vin
aigrett', 'eaten', 'thank', 'cooki', 'littl', 'learn', 'feed', 'hand', 'mouth',
'gum', 'around', 'dissolv', 'pretti', 'quick', 'huge', 'mess', 'biscuit', 'toas
t', 'stale', 'day', 'open', 'packag', 'could', 'give']
```

In [18]:
```
#Word 2 Vector for test corpus
list_of_sent_test_kd=[]
for sent in x_test_kd:
    list_of_sent_test_kd.append(sent.split())

w2v_model_test_kd=Word2Vec(list_of_sent_test_kd,min_count=5,size=300, workers=4)
print(w2v_model_test_kd)

w2v_words_test_kd = list(w2v_model_test_kd.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words_test_kd))
print("sample words ", w2v_words_test_kd[0:300])
```

```
Word2Vec(vocab=4788, size=300, alpha=0.025)
number of words that occured minimum 5 times  4788
sample words  ['must', 'good', 'fortun', 'tri', 'gummi', 'bear', 'kid', 'caus',
'knew', 'great', 'exist', 'pound', 'bag', 'chewi', 'ton', 'flavor', 'make', 'be
st', 'around', 'unlik', 'one', 'review', 'mine', 'ship', 'almost', 'immedi', 'm
ayb', 'amazon', 'prime', 'dave', 'postum', 'tast', 'caffein', 'none', 'store',
'carri', 'anymor', 'everywher', 'onlin', 'stock', 'unknown', 'avail', 'go', 'pr
oduct', 'expens', 'get', 'stuff', 'price', 'high', 'everyday', 'use', 'got', 'p
acket', 'health', 'food', 'that', 'funni', 'thing', 'box', 'think', 'assum', 'e
nclos', 'home', 'open', 'bit', 'bare', 'line', 'bottom', 'dont', 'know', 'put',
'littl', 'big', 'wait', 'want', 'mislead', 'public', 'buck', 'love', 'fact', 'o
rgan', 'come', 'sugar', 'cane', 'erythritol', 'corn', 'far', 'tell', 'made', 'i
ve', 'call', 'everi', 'compani', 'could', 'lower', 'happi', 'camper', 'addict',
'though', 'thank', 'god', 'doesnt', 'calori', 'dark', 'much', 'stronger', 'medi
um', 'awar', 'like', 'strong', 'compar', 'eas', 'origin', 'may', 'pricey', 'alw
ay', 'two', 'averag', 'cup', 'mind', 'well', 'worth', 'conveni', 'cake', 'eve
r', 'anyth', 'restaur', 'often', 'better', 'bottl', 'soda', 'version', 'marke
t', 'also', 'nice', 'pack', 'month', 'old', 'babi', 'girl', 'picki', 'eater',
'nestl', 'cerelac', 'popular', 'brand', 'india', 'abl', 'find', 'indian', 'near
bi', 'accident', 'found', 'sinc', 'meal', 'time', 'fun', 'enjoy', 'flavour', 'd
efinit', 'gerber', 'earth', 'understand', 'enter', 'usa', 'breast', 'milk', 'su
pplement', 'start', 'daughter', 'would', 'never', 'eat', 'cereal', 'fuzzi', 'th
roughout', 'life', 'saver', 'avid', 'user', 'yrs', 'prefer', 'sweetner', 'subst
itut', 'right', 'bitter', 'bad', 'other', 'easi', 'cook', 'bake', 'anyon', 'you
r', 'diabet', 'hubbi', 'becam', 'insulin', 'depend', 'pink', 'hmm', 'problem',
'ill', 'purchas', 'extra', 'even', 'unavail', 'area', 'search', 'internet', 'pl
easant', 'pleas', 'sale', 'way', 'formul', 'slight', 'differ', 'artifici', 'bel
iev', 'give', 'bar', 'delici', 'green', 'fill', 'center', 'chocol', 'coat', 'un
expect', 'soft', 'faint', 'honey', 'firmer', 'contrast', 'textur', 'total', 'co
unt', 'par', 'energi', 'yet', 'stay', 'longer', 'perhap', 'due', 'carbohydr',
'balanc', 'recommend', 'hummus', 'dip', 'seen', 'supermarket', 'decid', 'orde
r', 'excel', 'recip', 'black', 'cherri', 'natur', 'zero', 'son', 'altern', 'bu
y', 'groceri', 'three', 'breakfast', 'theyr', 'gluten', 'cooki', 'diagnos', 'pa
rt', 'free', 'flour', 'first', 'batch', 'amount', 'gum', 'math', 'gram', 'reali
z', 'duh', 'outstand', 'result', 'read', 'direct', 'wont', 'disappoint', 'yello
w', 'lab', 'rawhid', 'prompt', 'worri', 'local', 'hous', 'famili', 'oatmeal',
'sweet', 'actual', 'see', 'oat', 'introduc', 'illi', 'pari']
```

In [19]:
```
#train corpus
sent_vectors_train_kd = []; # the avg-w2v for each sentence/review is stored in t
for sent in list_of_sent_train_kd: # for each review/sentence
    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 sent: # for each word in a review/sentence
        if word in w2v_words_train_kd:
            #print(word)
            vec = w2v_model_train_kd.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_train_kd.append(sent_vec)
print(len(sent_vectors_train_kd))
print(len(sent_vectors_train_kd[0]))
```

```
36000
300
```

In [20]:
```
sent_vectors_cv_kd = []; # the avg-w2v for each sentence/review is stored in this
for sent in list_of_sent_cv_kd: # for each review/sentence
    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 sent: # for each word in a review/sentence
        if word in w2v_words_cv_kd:
            #print(word)
            vec = w2v_model_cv_kd.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_cv_kd.append(sent_vec)
print(len(sent_vectors_cv_kd))
print(len(sent_vectors_cv_kd[0]))
```

```
12000
300
```

In [21]:
```
#test Corpus
sent_vectors_test_kd = []; # the avg-w2v for each sentence/review is stored in th
for sent in list_of_sent_test_kd: # for each review/sentence
    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 sent: # for each word in a review/sentence
        if word in w2v_words_test_kd:

            vec = w2v_model_test_kd.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_test_kd.append(sent_vec)
print(len(sent_vectors_test_kd))
print(len(sent_vectors_test_kd[0]))
```

```
12000
300
```

In [ ]:
```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components = 200)
X1_train_tfidf_svd = svd.fit_transform(sent_vectors_train_kd)
X1_cv_tfidf_svd = svd.transform(sent_vectors_cv_kd)
X1_test_tfidf_svd = svd.transform(sent_vectors_test_kd)
```

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
optimal_k = knn_kd(X1_train_tfidf_svd,X1_cv_tfidf_svd,y_train_kd,y_cv_kd)
```

In [ ]:
```
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(X1_train_tfidf_svd, y_train_kd)

# predict the response
pred_avgw2v_kd = knn_optimal.predict(X1_test_tfidf_svd)

# evaluate accuracy
acc_bow = accuracy_score(y_test, pred_avgw2v_kd) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc
```

In [ ]:
```
err_bow = 100-acc_bow
err_bow
```

In [ ]:
```
plot_confusion_matrix(confusion_matrix(y_test, pred_avgw2v_kd), classes=["Negativ
```

AVG TFIDF W2V

In [22]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [23]:
```
tfidf_feat = tf_idf_vect.get_feature_names()#getting feature list
```

In [24]:
```
from tqdm import tqdm
tfidf_sent_vectors_train_kd = []; # the tfidf-w2v for each sentence/review is sto
row=0;
for sent in tqdm(list_of_sent_train_kd): # 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
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_train_kd:
            vec = w2v_model_train_kd.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))#taking the dic
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_train_kd.append(sent_vec)
    row += 1

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

```
100%|██████████████████████████████████████████████████████████████
| 36000/36000 [01:25<00:00, 422.25it/s]

36000
300
```

In [28]:
```
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(final['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_kd = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [29]:
```
from tqdm import tqdm
tfidf_sent_vectors_cv_kd = []; # the tfidf-w2v for each sentence/review is stored
row=0;
for sent in tqdm(list_of_sent_cv_kd): # 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
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_cv_kd:
            vec = w2v_model_cv_kd.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_kd[word]*(sent.count(word)/len(sent))#taking the
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_cv_kd.append(sent_vec)
    row += 1

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

```
100%|
| 12000/12000 [00:22<00:00, 544.12it/s]

12000
300
```

In [30]:
```
model = TfidfVectorizer(ngram_range=(1,2))
tf_idf_matrix = model.fit_transform(final['CleanedText'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_test = dict(zip(model.get_feature_names(), list(model.idf_)))
```

In [31]:
```
#test corpus
from tqdm import tqdm
tfidf_sent_vectors_test_kd = []; # the tfidf-w2v for each sentence/review is stor
row=0;
for sent in tqdm(list_of_sent_test_kd): # 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
    for word in sent: # for each word in a review/sentence
        if word in w2v_words_test_kd:
            vec = w2v_model_test_kd.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_test[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_test_kd.append(sent_vec)
    row += 1

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

```
100%|████████████████████████████████████████████████████████████████████████████
| 12000/12000 [00:23<00:00, 519.84it/s]

12000
300
```

In [50]:
```
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components = 200)
X2_train_tfidf_svd = svd.fit_transform(tfidf_sent_vectors_train_kd)
X2_cv_tfidf_svd = svd.transform(tfidf_sent_vectors_cv_kd)
X2_test_tfidf_svd = svd.transform(tfidf_sent_vectors_test_kd)
```

In [ ]:
```
import warnings
warnings.filterwarnings("ignore")
optimal_k = knn_brute(X2_train_tfidf_svd,X2_cv_tfidf_svd,y_train_kd,y_cv_kd)
```

In [ ]:
```
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)

# fitting the model
knn_optimal.fit(X2_train_tfidf_svd, y_train_kd)

# predict the response
pred_avgw2v_tfidf_kd = knn_optimal.predict(X2_test_tfidf_svd)

# evaluate accuracy
acc_bow = accuracy_score(y_test, pred_avgw2v_tfidf_kd) * 100
print('\nThe accuracy of the knn classifier for k = %d is %f%%' % (optimal_k, acc
```

In [ ]:
```
err_bow = 100-acc_bow
err_bow
```

In [ ]:
```
plot_confusion_matrix(confusion_matrix(y_test, pred_avgw2v_tfidf_kd), classes=["N
```