Amazon food classification using KNN

June 2, 2018

1 Amazon food classification using KNN

```
In [1]: import sqlite3
        import pandas as pd
        import numpy as np
        import nltk
        import string
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [2]: con = sqlite3.connect('./final.sqlite')

        review= pd.read_sql_query('''select * from Reviews''',con)
```

```
In [3]: review.Score.value_counts()
```

```
Out[3]: positive    8379
        negative    1603
        Name: Score, dtype: int64
```

```
In [4]: review=review.sort_values('Time')
```

1.1 BOW

```
In [5]: from sklearn.feature_extraction.text import TfidfTransformer
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn import datasets, neighbors
        from matplotlib.colors import ListedColormap

        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
```

```
In [6]: #BoW
        count_vect = CountVectorizer() #in scikit-learn
```

```
In [7]: final_counts=count_vect.fit_transform(review['Text'].values)

In [8]: final_counts.get_shape()

Out[8]: (9982, 22825)

In [9]: final_counts.max()

Out[9]: 108

In [10]: per=int(0.7*final_counts.shape[0])
         a=final_counts[0:per]
         b=final_counts[per:]
         label=review.Score[0:per]
         label1=review.Score[per:]

In [11]: a.get_shape()

Out[11]: (6987, 22825)

In [12]: b.get_shape()

Out[12]: (2995, 22825)

In [13]: label

Out[13]: 30      positive
         7       positive
         421     positive
         329     positive
         420     positive
         390     positive
         315     negative
         229     positive
         414     positive
         422     positive
         368     positive
         240     positive
         239     positive
         467     positive
         845     positive
         829     positive
         389     positive
         376     positive
         828     positive
         352     positive
         328     positive
         846     positive
         426     negative
         311     negative
```

```
413     positive
848     positive
307     positive
6       positive
995     positive
306     positive
          ...
9827    negative
6972    negative
6848    positive
7541    positive
8657    positive
1247    positive
5689    positive
6541    positive
8338    positive
3540    positive
1246    positive
5532    positive
8366    negative
1206    negative
8559    positive
2640    positive
3785    positive
1053    positive
3781    positive
2139    positive
7076    positive
2041    negative
498     negative
9901    positive
1603    positive
5530    positive
6776    negative
4982    positive
496     positive
2319    negative
Name: Score, Length: 6987, dtype: object
```

In [14]:
```python
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler(with_mean=False).fit_transform(a)
print(standardized_data.shape)
```

```
(6987, 22825)


c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\utils\vali
  warnings.warn(msg, DataConversionWarning)
```

In [15]:
```
# TSNE

from sklearn.manifold import TSNE

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = standardized_data[0:2000].todense()


model = TSNE(n_components=2, random_state=0,perplexity=30)
tsne_data = model.fit_transform(data_1000)
```

In [16]:
```
tsne_data.shape
```

Out[16]:
```
(2000, 2)
```

In [17]:
```
label.shape
```

Out[17]:
```
(6987,)
```

In [19]:
```
def partition(x):
    if x =='negative':
        return 0
    return 1
```

In [20]:
```
#changing reviews with score less than 3 to be positive and vice-versa
actualScore = label
positiveNegative = actualScore.map(partition)
label= positiveNegative
```

In [21]:
```
label
```

Out[21]:
```
30     1
7      1
421    1
329    1
420    1
390    1
315    0
229    1
414    1
422    1
368    1
240    1
239    1
467    1
845    1
829    1
389    1
376    1
```

```
828     1
352     1
328     1
846     1
426     0
311     0
413     1
848     1
307     1
6       1
995     1
306     1
       ..
9827    0
6972    0
6848    1
7541    1
8657    1
1247    1
5689    1
6541    1
8338    1
3540    1
1246    1
5532    1
8366    0
1206    0
8559    1
2640    1
3785    1
1053    1
3781    1
2139    1
7076    1
2041    0
498     0
9901    1
1603    1
5530    1
6776    0
4982    1
496     1
2319    0
Name: Score, Length: 6987, dtype: int64
```

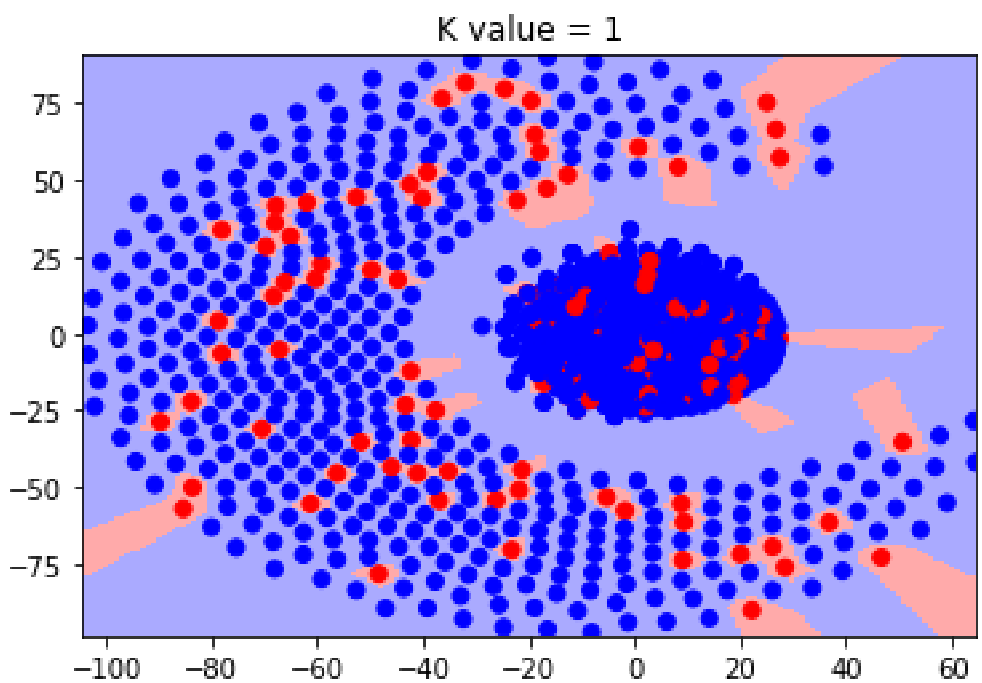
In [136]:
```
from sklearn.neighbors import KNeighborsClassifier
def knn_comparison(data, n_neighbors = 15):
    '''
```

```
This function finds k-NN and plots the data.
'''
X = data[:]
y = label[:100]

# grid cell size
h = 1
cmap_light = ListedColormap(['#FFAAAA',  '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#0000FF'])


# the core classifier: k-NN
clf = KNeighborsClassifier(n_neighbors)
clf.fit(X, y)

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

# we create a mesh grid (x_min,y_min) to (x_max y_max) with 0.02 grid spaces
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

print(xx.ravel())
print(yy.ravel())
print(np.c_[xx.ravel(), yy.ravel()].shape)

# we predict the value (either 0 or 1) of each element in the grid
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# xx.ravel() will give a flatten array

# np.c_ : Translates slice objects to concatenation along the second axis.
# > np.c_[np.array([1,2,3]), np.array([4,5,6])]
# > array([[1, 4],
#          [2, 5],
#          [3, 6]])   (source: np.c_ documentation)




# convert the out back to the xx shape (we need it to plot the decission boundry
Z = Z.reshape(xx.shape)


# pcolormesh will plot the (xx,yy) grid with colors according to the values of Z
# it looks like decision boundry
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
```

```
    # scatter plot of with given points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)




    #defining scale on both axises
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())


    # set the title
    plt.title('K value = '+str(n_neighbors))


    plt.show()
```

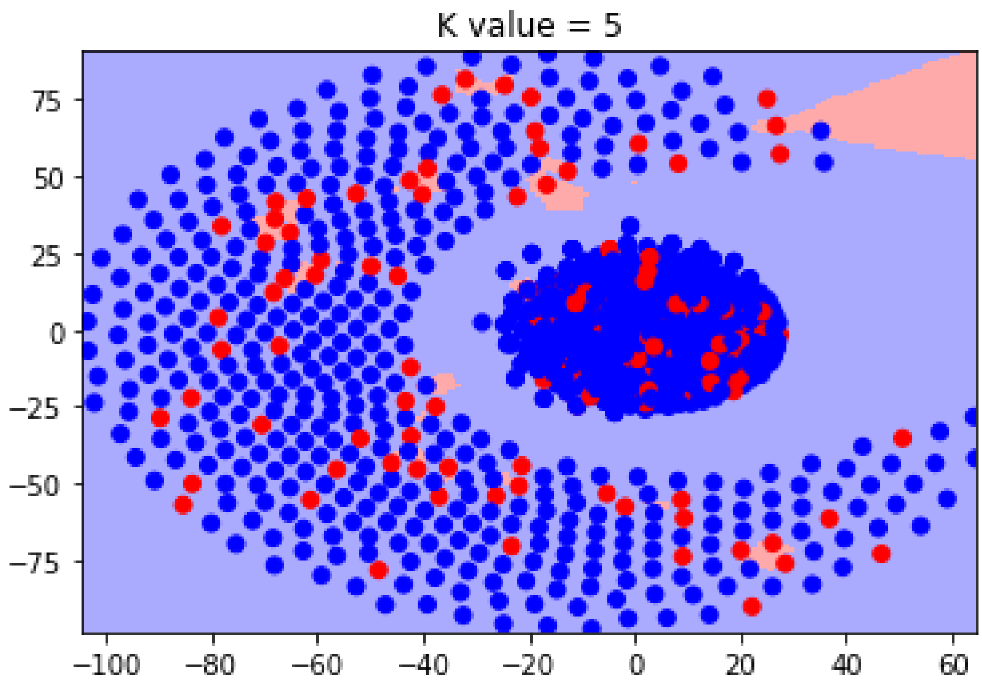
```
In [43]: knn_comparison(tsne_data, 1)
```

```
[-104.44168091 -103.44168091 -102.44168091 ...   62.55831909   63.55831909
   64.55831909]
[-98.30327606 -98.30327606 -98.30327606 ...  90.69672394  90.69672394
  90.69672394]
(32300, 2)
```



In [44]: knn_comparison(tsne_data, 5)

```
[-104.44168091 -103.44168091 -102.44168091 ...   62.55831909   63.55831909
   64.55831909]
[-98.30327606 -98.30327606 -98.30327606 ...  90.69672394  90.69672394
  90.69672394]
(32300, 2)
```



In [31]:
```
# ============================== loading libraries ==================================
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from collections import Counter
from sklearn.metrics import accuracy_score
from sklearn import cross_validation
# =====================================================================================
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [32]: # Part I
         # ============================== data preprocessing ==================================

         # define column names
         names = ['x', 'y', 'class']

         # loading training data


         # create design matrix X and target vector y
         X = np.array(tsne_data) # end index is exclusive
         y = np.array(label[0:2000]) # showing you two ways of indexing a pandas df

In [33]: from sklearn.preprocessing import StandardScaler
         standardized_data = StandardScaler(with_mean=False).fit_transform(b)
         print(standardized_data.shape)
```

```
(2995, 22825)


c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\utils\valid
  warnings.warn(msg, DataConversionWarning)
```

```
In [34]: # TSNE

         from sklearn.manifold import TSNE

         # Picking the top 1000 points as TSNE takes a lot of time for 15K points
         data_1000 = standardized_data[0:2000].todense()


         model = TSNE(n_components=2, random_state=0,perplexity=30)
         tsne_data1 = model.fit_transform(data_1000)

In [37]: actualScore = label1
         positiveNegative = actualScore.map(partition)
         label1= positiveNegative

In [38]: print(tsne_data1.shape)
         print(label1.shape)
```

```
(2000, 2)
(2995,)
```

```
In [51]: # split the data set into train and test
         X_test, y_test = np.array(tsne_data1), np.array(label1[0:2000])
```

```python
# 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, y, test_size=0.3)

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

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

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

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

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

```
CV accuracy for k = 1 is 81%

CV accuracy for k = 3 is 85%

CV accuracy for k = 5 is 86%

CV accuracy for k = 7 is 87%

CV accuracy for k = 9 is 88%

CV accuracy for k = 11 is 88%

CV accuracy for k = 13 is 88%

CV accuracy for k = 15 is 88%

CV accuracy for k = 17 is 88%

CV accuracy for k = 19 is 88%

CV accuracy for k = 21 is 88%

CV accuracy for k = 23 is 88%
```

```
CV accuracy for k = 25 is 88%

CV accuracy for k = 27 is 88%

CV accuracy for k = 29 is 88%

****Test accuracy for k = 1 is 72%
```

```
In [41]: knn = KNeighborsClassifier(5)
         knn.fit(X_tr,y_tr)
         pred = knn.predict(X_test)
         acc = accuracy_score(y_test, pred, normalize=True) * float(100)

In [53]: print('\n****Test accuracy for k = 5 is %d%%' % (acc))
```

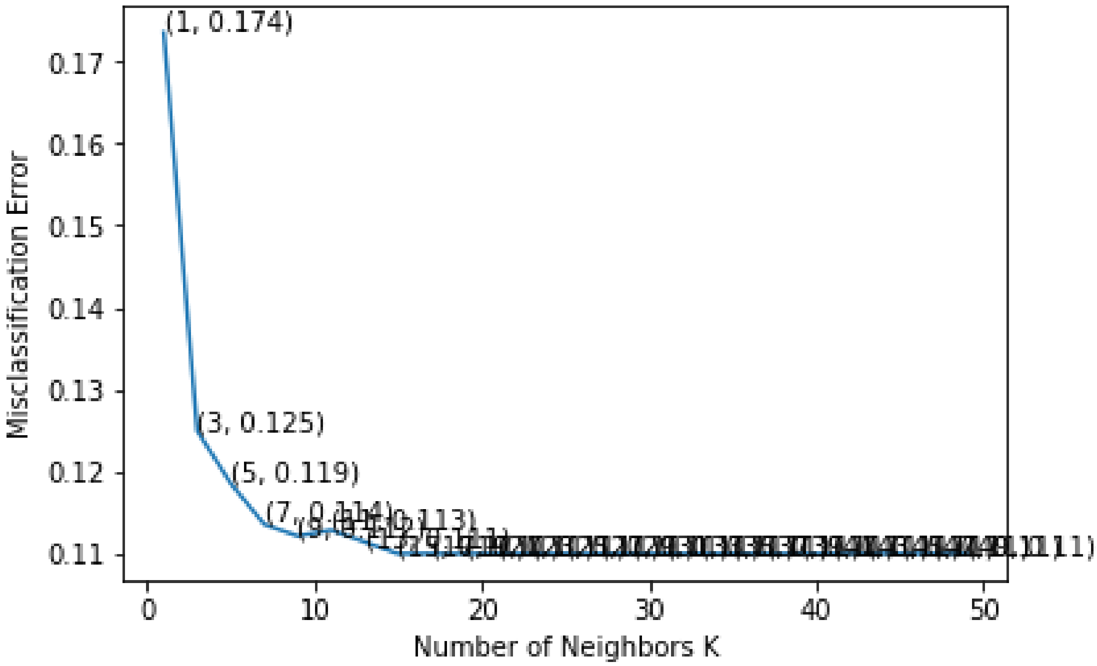
```
****Test accuracy for k = 5 is 81%
```

1.1.1 Using 10-fold CV

```
In [54]: # 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, X_tr, y_tr, 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 15.
```




```
the misclassification error for each k value is :  [0.174 0.125 0.119 0.114 0.112 0.113 0.111 0
 0.11  0.11  0.11  0.11  0.11  0.11  0.11  0.11  0.11  0.11  0.11  0.11
 0.11 ]
```

```
In [55]: # ============================== 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_tr, y_tr)

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

         # 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 = 15 is 82.450000%
```

1.1.2 Using kd-tree

In [39]:
```
# split the data set into train and test
X_test, y_test = np.array(tsne_data1), np.array(label1[0:2000])

# 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, y, test_size=0.3)

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

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

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

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

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

```
CV accuracy for k = 1 is 81%

CV accuracy for k = 3 is 86%

CV accuracy for k = 5 is 87%

CV accuracy for k = 7 is 88%

CV accuracy for k = 9 is 89%

CV accuracy for k = 11 is 89%
```

```
CV accuracy for k = 13 is 89%

CV accuracy for k = 15 is 89%

CV accuracy for k = 17 is 89%

CV accuracy for k = 19 is 89%

CV accuracy for k = 21 is 89%

CV accuracy for k = 23 is 89%

CV accuracy for k = 25 is 89%

CV accuracy for k = 27 is 89%

CV accuracy for k = 29 is 89%

****Test accuracy for k = 1 is 85%
```

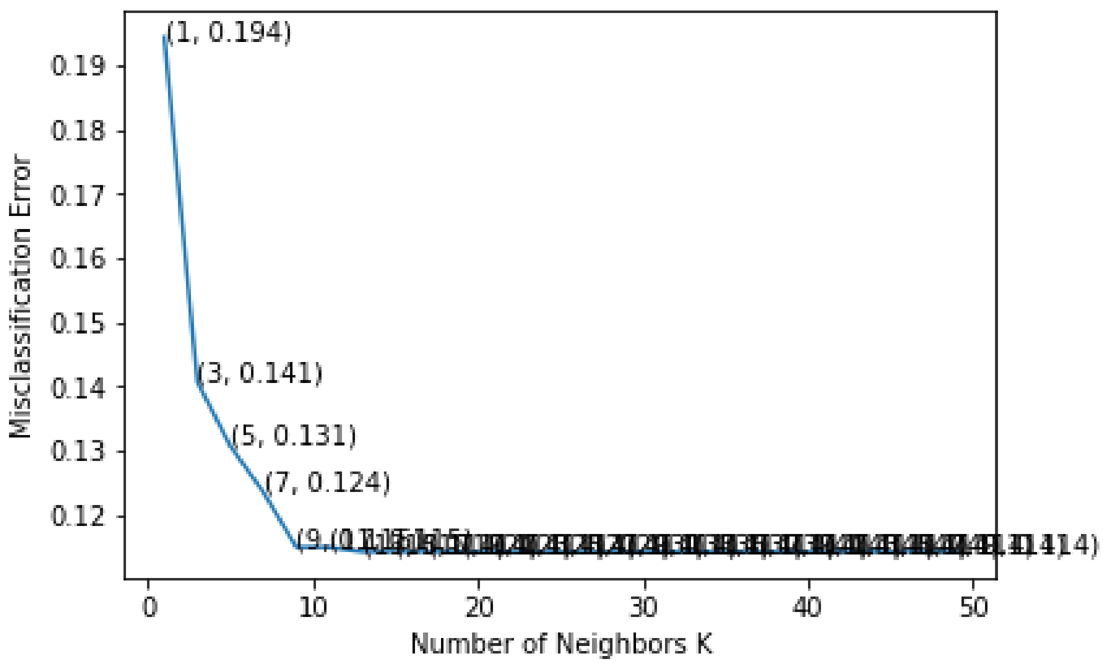
```
In [44]: knn = KNeighborsClassifier(5)
         knn.fit(X_tr,y_tr)
         pred = knn.predict(X_test)
         acc = accuracy_score(y_test, pred, normalize=True) * float(100)
         print('\n****Test accuracy for k = 5 is %d%%' % (acc))
```

```
****Test accuracy for k = 5 is 99%
```

```
In [40]: # 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,algorithm='kd_tree')
             scores = cross_val_score(knn, X_tr, y_tr, 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 13.
```




```
the misclassification error for each k value is :  [0.194 0.141 0.131 0.124 0.115 0.115 0.114 0
 0.114 0.114 0.114 0.114 0.114 0.114 0.114 0.114 0.114 0.114 0.114 0.114
 0.114]
```

```
In [43]: # ============================== KNN with k = optimal_k ===============================
         # instantiate learning model k = optimal_k
```

```
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_tree')

# fitting the model
knn_optimal.fit(X_tr, y_tr)

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

# 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 = 13 is 100.000000%
```

1.2 TF-idf

In [67]:
```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2))
final_tf_idf = tf_idf_vect.fit_transform(review['Text'].values)
```

In [68]:
```
final_tf_idf.get_shape()
```

Out[68]:
```
(9982, 284906)
```

In [69]:
```
features = tf_idf_vect.get_feature_names()
len(features)
```

Out[69]:
```
284906
```

In [70]:
```
per=int(0.7*final_tf_idf.shape[0])
a=final_tf_idf[0:per]
b=final_tf_idf[per:]
label=review.Score[0:per]
label1=review.Score[per:]
```

In [71]:
```
def partition(x):
    if x =='negative':
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = label
positiveNegative = actualScore.map(partition)
label= positiveNegative
```

In [72]:
```
label
```

```
Out[72]: 30      1
         7       1
         421     1
         329     1
         420     1
         390     1
         315     0
         229     1
         414     1
         422     1
         368     1
         240     1
         239     1
         467     1
         845     1
         829     1
         389     1
         376     1
         828     1
         352     1
         328     1
         846     1
         426     0
         311     0
         413     1
         848     1
         307     1
         6       1
         995     1
         306     1
                ..
         9827    0
         6972    0
         6848    1
         7541    1
         8657    1
         1247    1
         5689    1
         6541    1
         8338    1
         3540    1
         1246    1
         5532    1
         8366    0
         1206    0
         8559    1
         2640    1
         3785    1
```

```
1053    1
3781    1
2139    1
7076    1
2041    0
498     0
9901    1
1603    1
5530    1
6776    0
4982    1
496     1
2319    0
Name: Score, Length: 6987, dtype: int64
```

In [73]:
```python
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler(with_mean=False).fit_transform(a)
print(standardized_data.shape)
```

```
(6987, 284906)
```

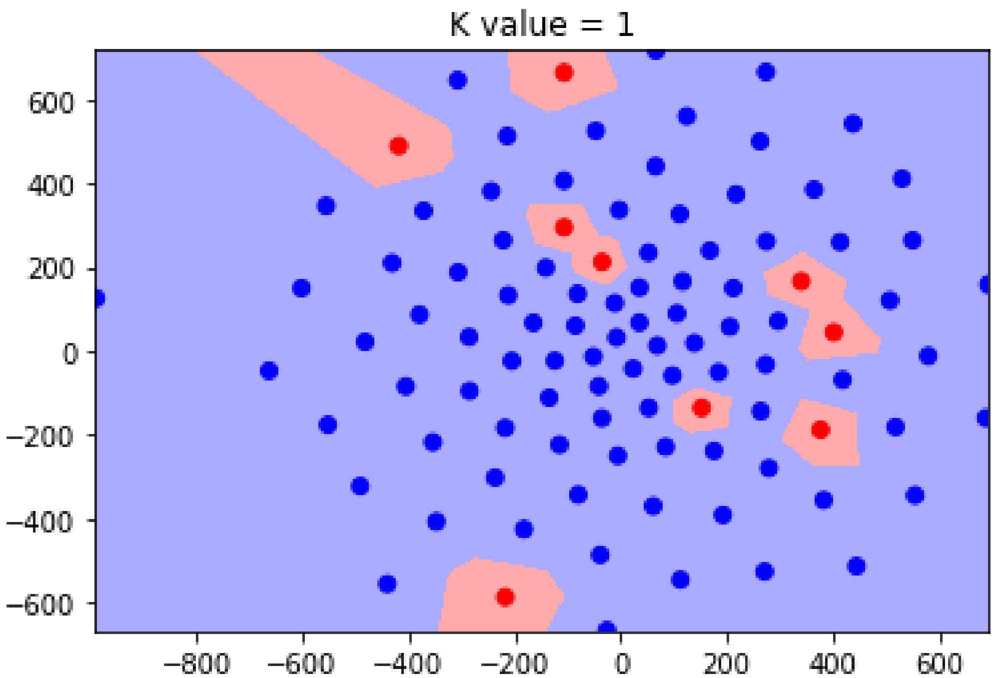
In [74]:
```python
# TSNE

from sklearn.manifold import TSNE

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = standardized_data[0:100].todense()


model = TSNE(n_components=2, random_state=0,perplexity=30)
tsne_data = model.fit_transform(data_1000)
```

In [75]:
```python
tsne_data.shape
```

Out[75]:
```
(100, 2)
```

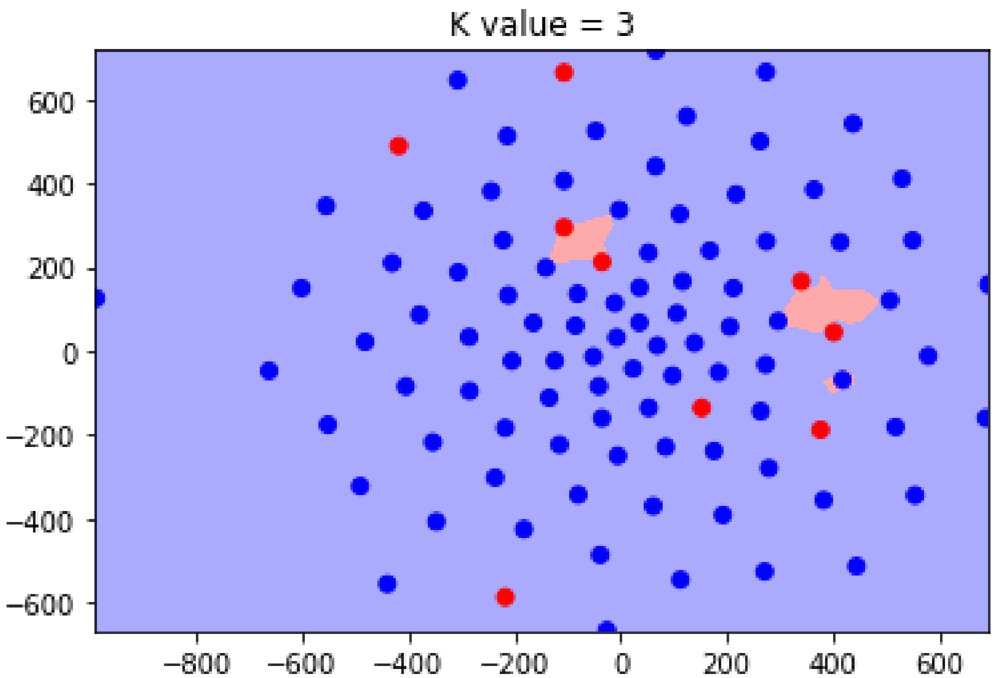
In [73]:
```python
knn_comparison(tsne_data, 1)
```

```
[-989.53839111 -988.53839111 -987.53839111 ...  691.46160889  692.46160889
  693.46160889]
[-666.60211182 -666.60211182 -666.60211182 ...  716.39788818  716.39788818
  716.39788818]
(2330656, 2)
```



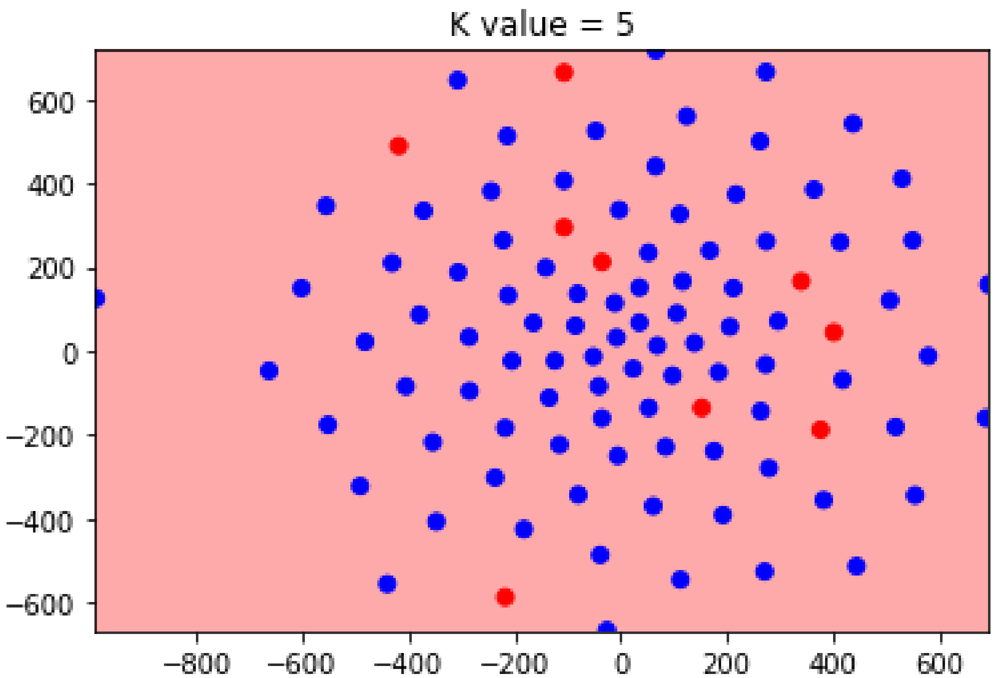

```
In [74]: knn_comparison(tsne_data, 3)
```

```
[-989.53839111 -988.53839111 -987.53839111 ...  691.46160889  692.46160889
  693.46160889]
[-666.60211182 -666.60211182 -666.60211182 ...  716.39788818  716.39788818
  716.39788818]
(2330656, 2)
```




```
In [75]: knn_comparison(tsne_data, 5)
```

```
[-989.53839111 -988.53839111 -987.53839111 ...  691.46160889  692.46160889
  693.46160889]
[-666.60211182 -666.60211182 -666.60211182 ...  716.39788818  716.39788818
  716.39788818]
(2330656, 2)
```




```
In [76]: from sklearn.preprocessing import StandardScaler
         standardized_data = StandardScaler(with_mean=False).fit_transform(b)
         print(standardized_data.shape)
```

```
(2995, 284906)
```

```
In [77]: # TSNE

         from sklearn.manifold import TSNE

         # Picking the top 1000 points as TSNE takes a lot of time for 15K points
         data_1000 = standardized_data[0:100].todense()


         model = TSNE(n_components=2, random_state=0,perplexity=30)
         tsne_data1 = model.fit_transform(data_1000)
```

```
In [78]: actualScore = label1
         positiveNegative = actualScore.map(partition)
         label1= positiveNegative
```

```
In [79]: X = np.array(tsne_data) # end index is exclusive
         y = np.array(label[0:100])
```

```
In [82]: # split the data set into train and test
         X_test, y_test = np.array(tsne_data1), np.array(label1[0:100])

         # 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, y, test_size=0.3)

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

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

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

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

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

```
CV accuracy for k = 1 is 86%

CV accuracy for k = 3 is 86%

CV accuracy for k = 5 is 86%

CV accuracy for k = 7 is 86%

CV accuracy for k = 9 is 86%

CV accuracy for k = 11 is 86%

CV accuracy for k = 13 is 86%

CV accuracy for k = 15 is 86%

CV accuracy for k = 17 is 86%

CV accuracy for k = 19 is 86%

CV accuracy for k = 21 is 86%
```

```
CV accuracy for k = 23 is 86%

CV accuracy for k = 25 is 86%

CV accuracy for k = 27 is 86%

CV accuracy for k = 29 is 86%

****Test accuracy for k = 1 is 84%
```

```
In [83]: knn = KNeighborsClassifier(5)
         knn.fit(X_tr,y_tr)
         pred = knn.predict(X_test)
         acc = accuracy_score(y_test, pred, normalize=True) * float(100)
         print('\n****Test accuracy for k = 5 is %d%%' % (acc))
```

```
****Test accuracy for k = 5 is 86%
```

1.2.1 Using 10 fold

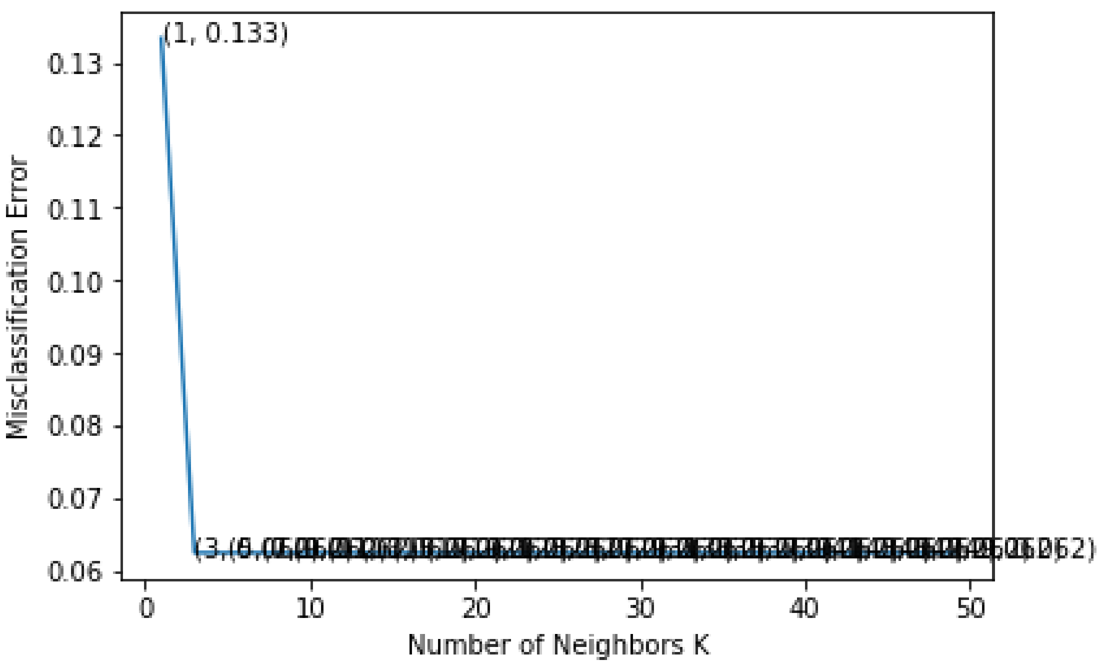
```
In [84]: # creating odd list of K for KNN
         myList = list(range(0,50))
         neighbors = list(filter(lambda x: x % 2 != 0, myList))

         # 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, X_tr, y_tr, 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))
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
```

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




```
the misclassification error for each k value is :  [0.133 0.062 0.062 0.062 0.062 0.062 0.062 0
 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062
 0.062]
```

In [85]:
```
# ============================== 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_tr, y_tr)

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

    # 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 = 3 is 86.000000%
```

1.2.2 Using kd-tree

```
In [88]: # split the data set into train and test
         X_test, y_test = np.array(tsne_data1), np.array(label1[0:100])

         # 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, y, test_size=0.3)

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

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

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

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

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

```
CV accuracy for k = 1 is 83%

CV accuracy for k = 3 is 96%

CV accuracy for k = 5 is 96%
```

```
CV accuracy for k = 7 is 96%

CV accuracy for k = 9 is 96%

CV accuracy for k = 11 is 96%

CV accuracy for k = 13 is 96%

CV accuracy for k = 15 is 96%

CV accuracy for k = 17 is 96%

CV accuracy for k = 19 is 96%

CV accuracy for k = 21 is 96%

CV accuracy for k = 23 is 96%

CV accuracy for k = 25 is 96%

CV accuracy for k = 27 is 96%

CV accuracy for k = 29 is 96%

****Test accuracy for k = 1 is 83%
```

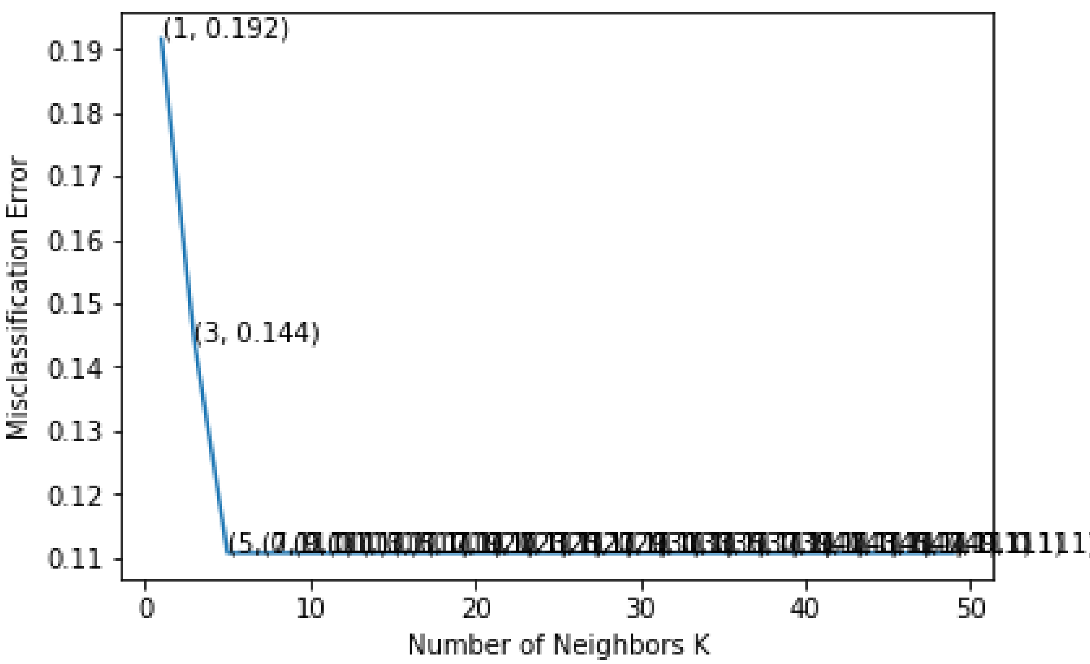
In [89]:
```
# creating odd list of K for KNN
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

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

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
    scores = cross_val_score(knn, X_tr, y_tr, 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)
```

```python
# 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))
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
```

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




```
the misclassification error for each k value is :  [0.192 0.144 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 0.111 0.111 0.111 0.111
 0.111]
```

```
In [90]: # ============================== KNN with k = optimal_k ==============================
         # instantiate learning model k = optimal_k
         knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_tree')

         # fitting the model
         knn_optimal.fit(X_tr, y_tr)

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

         # 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 = 5 is 86.000000%
```

1.3 W2V

```
In [91]: import codecs
         import glob
         import logging
         import multiprocessing
         import os
         import pprint

         import gensim.models.word2vec as w2v
         import sklearn.manifold

         import re
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\gensim\utils.py:11
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

```
In [92]: corpus_raw = u""
         for i in review['Text']:
             corpus_raw += i

         print("Corpus is now {0} characters long".format(len(corpus_raw)))
```

```
Corpus is now 4395080 characters long
```

```
In [93]: tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')

In [94]: raw_sentences = tokenizer.tokenize(corpus_raw)
```

In [95]:
```
#convert into a list of words
#rtemove unnnecessary,, split into words, no hyphens
#list of words
def sentence_to_wordlist(raw):
    clean = re.sub("[^a-zA-Z]"," ", raw)
    words = clean.split()
    return words
```

In [96]:
```
#sentence where each word is tokenized
sentences = []
for raw_sentence in raw_sentences:
    if len(raw_sentence) > 0:
        sentences.append(sentence_to_wordlist(raw_sentence))
```

In [97]:
```
token_count = sum([len(sentence) for sentence in sentences])
print("The corpus contains {0:,} tokens".format(token_count))
```

```
The corpus contains 829,639 tokens
```

In [98]:
```
#ONCE we have vectors
#step 3 - build model
#3 main tasks that vectors help with
#DISTANCE, SIMILARITY, RANKING

# Dimensionality of the resulting word vectors.
#more dimensions, more computationally expensive to train
#but also more accurate
#more dimensions = more generalized
num_features = 300
# Minimum word count threshold.
min_word_count = 3

# Number of threads to run in parallel.
#more workers, faster we train
num_workers = multiprocessing.cpu_count()

# Context window length.
context_size = 7

# Downsample setting for frequent words.
#0 - 1e-5 is good for this
downsampling = 1e-3

# Seed for the RNG, to make the results reproducible.
#random number generator
#deterministic, good for debugging
seed = 1
```

```
In [99]: food2vec = w2v.Word2Vec(
             sg=1,
             seed=seed,
             workers=num_workers,
             size=num_features,
             min_count=min_word_count,
             window=context_size,
             sample=downsampling
         )
```

```
In [100]: food2vec.build_vocab(sentences)
```

```
In [101]: print("Word2Vec vocabulary length:", len(food2vec.wv.vocab))
```

```
Word2Vec vocabulary length: 11450
```

```
In [102]: food2vec.train(sentences ,total_examples=food2vec.corpus_count, epochs=15)
```

```
Out[102]: (9165221, 12444585)
```

```
In [103]: if not os.path.exists("trained"):
              os.makedirs("trained")
```

```
In [104]: food2vec.save(os.path.join("trained", "food2vec.w2v"))
```

```
In [105]: model = w2v.Word2Vec.load(os.path.join("trained", "food2vec.w2v"))
```

```
In [106]: # Train your own Word2Vec model using your own text corpus
          import gensim
          import re
          i=0
          list_of_sent=[]

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


          for sent in review['Text'].values:
              filtered_sentence=[]
              sent=cleanhtml(sent)
              for w in sent.split():
```

```
                for cleaned_words in cleanpunc(w).split():
                    if(cleaned_words.isalpha()):
                        filtered_sentence.append(cleaned_words.lower())
                    else:
                        continue
            list_of_sent.append(filtered_sentence)
```

```
In [107]: w2v_model=gensim.models.Word2Vec(list_of_sent,min_count=5,size=50, workers=4)

In [108]: words = list(w2v_model.wv.vocab)
          print(len(words))
```

```
6928
```

```
In [109]: w2v_model.wv.most_similar('tasty')

Out[109]: [('flavorful', 0.8728250861167908),
           ('delicious', 0.8680999279022217),
           ('light', 0.8501684069633484),
           ('yummy', 0.8380963206291199),
           ('sweet', 0.8199455738067627),
           ('nice', 0.8169261813163757),
           ('satisfying', 0.8122435808181763),
           ('strong', 0.8122204542160034),
           ('smooth', 0.8070423007011414),
           ('creamy', 0.8062540888786316)]

In [110]: model.wv.similarity('tasty','tast')

Out[110]: 0.23638015285000968

In [111]: all_word_vectors_matrix = w2v_model.wv.syn0
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\ipykernel_launcher
  """Entry point for launching an IPython kernel.
```

```
In [112]: per=int(0.7*all_word_vectors_matrix.shape[0])
          a=all_word_vectors_matrix[0:per]
          b=all_word_vectors_matrix[per:]
          label=review.Score[0:per]
          label1=review.Score[per:]

In [113]: def partition(x):
              if x =='negative':
                  return 0
              return 1
```

```
        #changing reviews with score less than 3 to be positive and vice-versa
        actualScore = label
        positiveNegative = actualScore.map(partition)
        label= positiveNegative
```

```
In [114]: # perplexity=30
          from sklearn.preprocessing import StandardScaler
          standardized_data = StandardScaler(with_mean=False).fit_transform(a)
          print(standardized_data.shape)
```

```
(4849, 50)
```

```
In [115]: # TSNE

          from sklearn.manifold import TSNE

          # Picking the top 1000 points as TSNE takes a lot of time for 15K points
          data_1000 = standardized_data[0:100]


          model = TSNE(n_components=2, random_state=0,perplexity=30)
          tsne_data = model.fit_transform(data_1000)
```

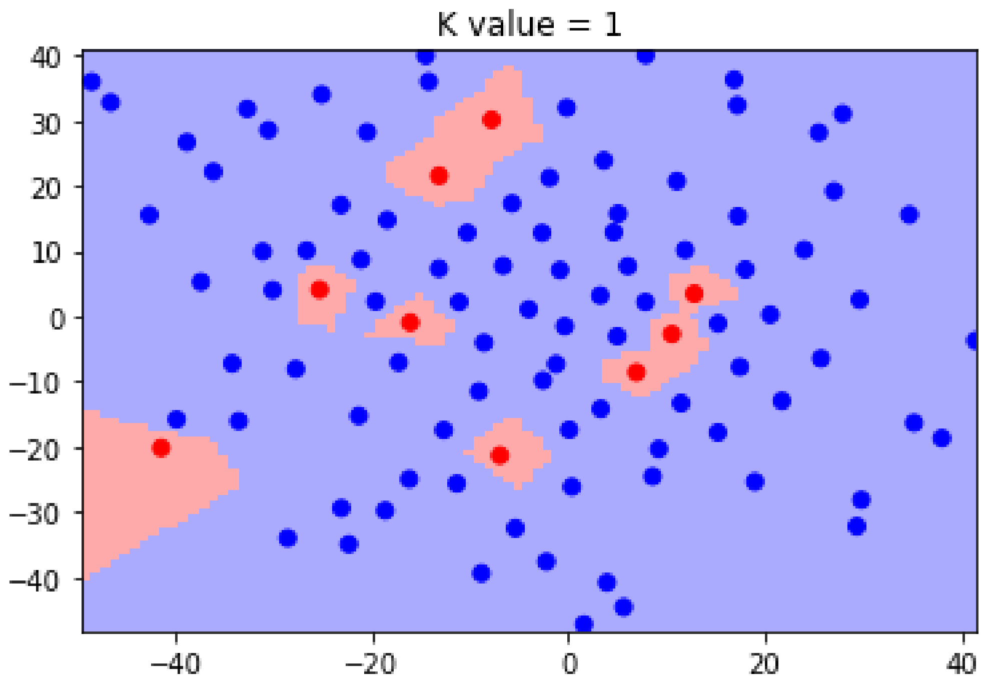
```
In [116]: tsne_data.shape
```

```
Out[116]: (100, 2)
```

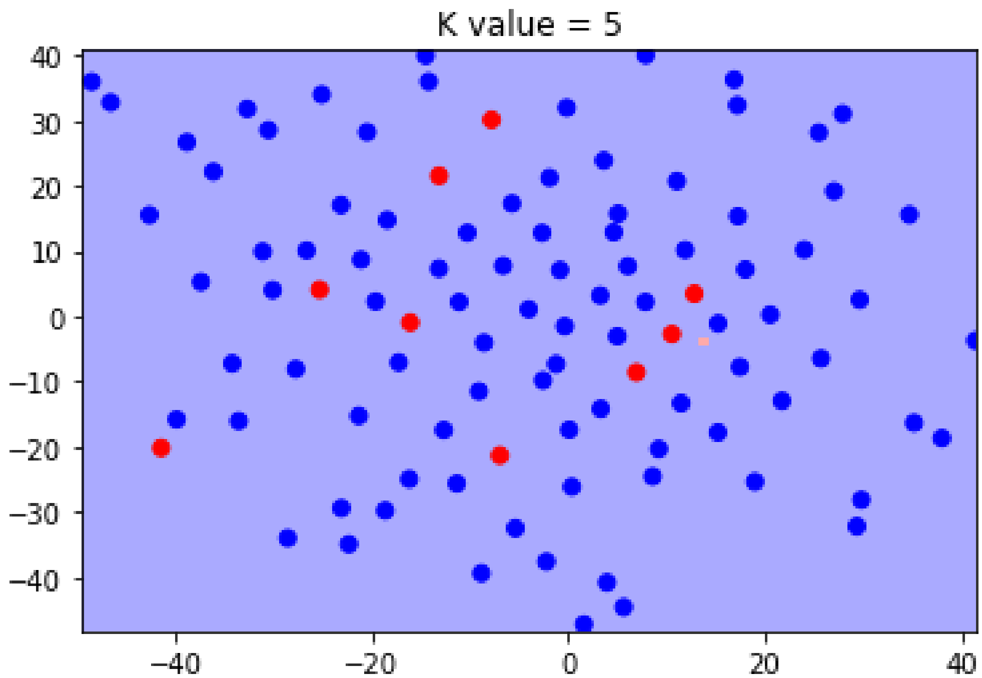
```
In [117]: from sklearn import datasets, neighbors
          from matplotlib.colors import ListedColormap
```

```
In [111]: knn_comparison(tsne_data, 1)
```

```
[-49.52266312 -48.52266312 -47.52266312 ...  39.47733688  40.47733688
  41.47733688]
[-48.23508453 -48.23508453 -48.23508453 ...  40.76491547  40.76491547
  40.76491547]
(8280, 2)
```




```
In [112]: knn_comparison(tsne_data, 5)

[-49.52266312 -48.52266312 -47.52266312 ...  39.47733688  40.47733688
  41.47733688]
[-48.23508453 -48.23508453 -48.23508453 ...  40.76491547  40.76491547
  40.76491547]
(8280, 2)
```




```
In [118]: from sklearn.preprocessing import StandardScaler
          standardized_data = StandardScaler(with_mean=False).fit_transform(b)
          print(standardized_data.shape)
```

(2079, 50)

```
In [119]: # TSNE

          from sklearn.manifold import TSNE

          # Picking the top 1000 points as TSNE takes a lot of time for 15K points
          data_1000 = standardized_data[0:100]


          model = TSNE(n_components=2, random_state=0,perplexity=30)
          tsne_data1 = model.fit_transform(data_1000)
```

```
In [120]: actualScore = label1
          positiveNegative = actualScore.map(partition)
          label1= positiveNegative
```

```
In [121]: X = np.array(tsne_data) # end index is exclusive
          y = np.array(label[0:100])
```

```
In [122]: from sklearn.feature_extraction.text import CountVectorizer


          # split the data set into train and test
          X_test, y_test = np.array(tsne_data1), np.array(label1[0:100])

          # 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, y, test_size=0.3)

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

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

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

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

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

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

CV accuracy for k = 3 is 86%

CV accuracy for k = 5 is 86%

CV accuracy for k = 7 is 86%

CV accuracy for k = 9 is 86%

CV accuracy for k = 11 is 86%

CV accuracy for k = 13 is 86%

CV accuracy for k = 15 is 86%

CV accuracy for k = 17 is 86%
```

```
CV accuracy for k = 19 is 86%

CV accuracy for k = 21 is 86%

CV accuracy for k = 23 is 86%

CV accuracy for k = 25 is 86%

CV accuracy for k = 27 is 86%

CV accuracy for k = 29 is 86%

****Test accuracy for k = 1 is 84%
```

```
In [123]: knn = KNeighborsClassifier(5)
          knn.fit(X_tr,y_tr)
          pred = knn.predict(X_test)
          acc = accuracy_score(y_test, pred, normalize=True) * float(100)
          print('\n****Test accuracy for k = 5 is %d%%' % (acc))
```

```
****Test accuracy for k = 5 is 84%
```

1.3.1 using 10 fold

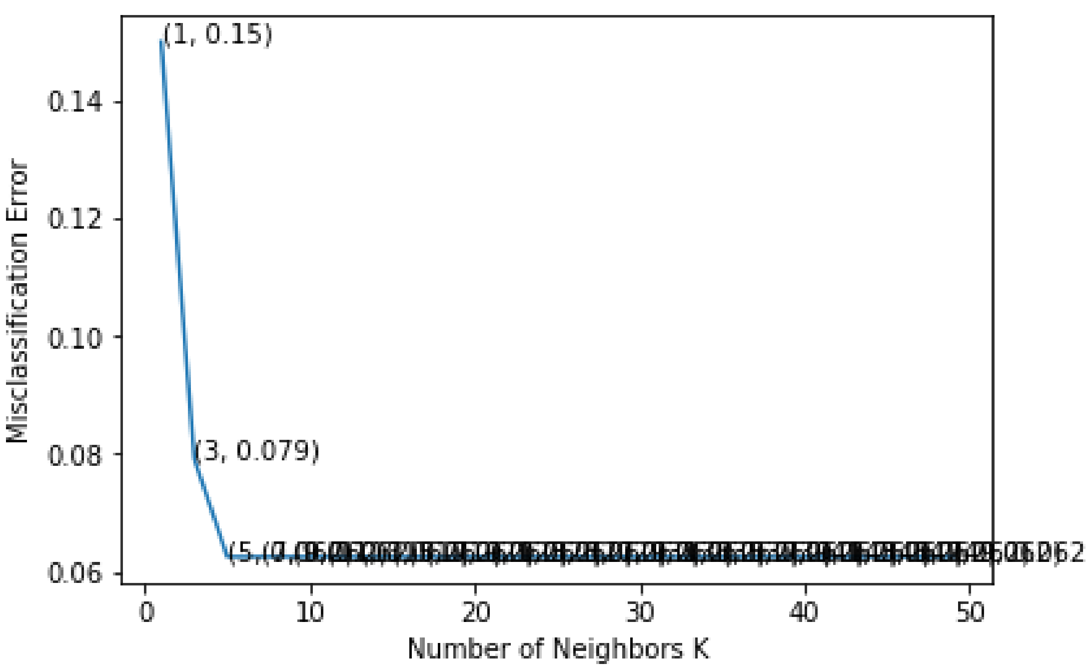
```
In [124]: # creating odd list of K for KNN
          myList = list(range(0,50))
          neighbors = list(filter(lambda x: x % 2 != 0, myList))

          # 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, X_tr, y_tr, 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
```

```python
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))
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
```

```
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
```

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




```
the misclassification error for each k value is :  [0.15  0.079 0.062 0.062 0.062 0.062 0.062 (
 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062
 0.062]
```

```
In [125]: # ============================== 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_tr, y_tr)

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

# 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 = 5 is 84.000000%
```

1.3.2 Using kd-tree

In [126]:
```
# split the data set into train and test
X_test, y_test = np.array(tsne_data1), np.array(label1[0:100])

# 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, y, test_size=0.3)

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

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

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

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

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

```
CV accuracy for k = 1 is 76%

CV accuracy for k = 3 is 93%
```

```
CV accuracy for k = 5 is 93%

CV accuracy for k = 7 is 93%

CV accuracy for k = 9 is 93%

CV accuracy for k = 11 is 93%

CV accuracy for k = 13 is 93%

CV accuracy for k = 15 is 93%

CV accuracy for k = 17 is 93%

CV accuracy for k = 19 is 93%

CV accuracy for k = 21 is 93%

CV accuracy for k = 23 is 93%

CV accuracy for k = 25 is 93%

CV accuracy for k = 27 is 93%

CV accuracy for k = 29 is 93%

****Test accuracy for k = 1 is 84%
```

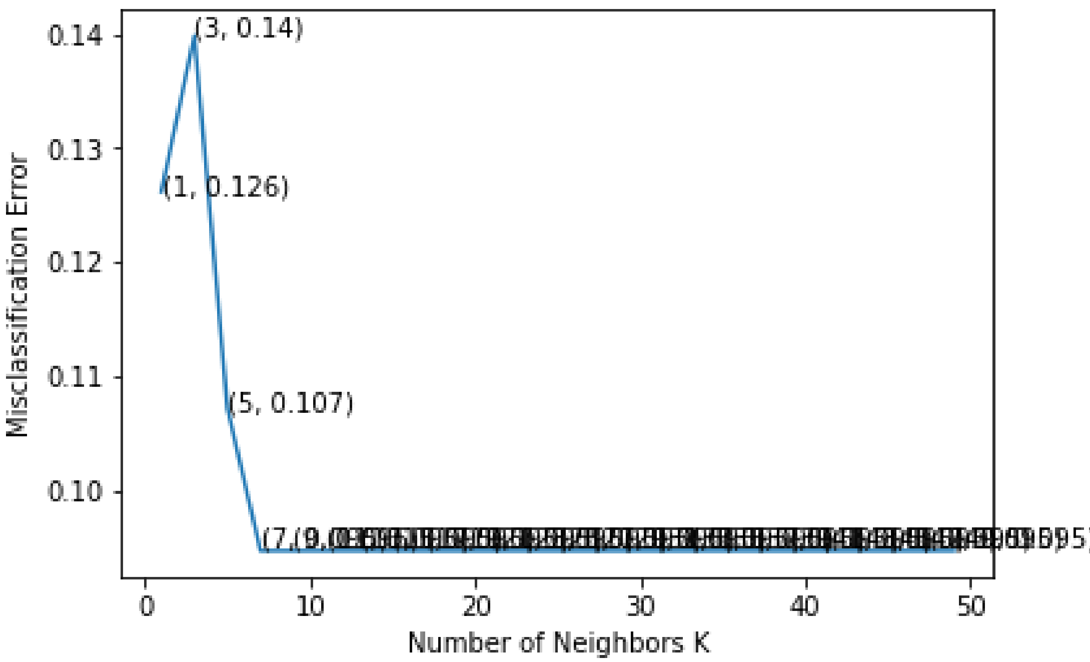
In [127]:
```python
# creating odd list of K for KNN
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

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

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
    scores = cross_val_score(knn, X_tr, y_tr, 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))]
```

```python
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))
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
```

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




```
the misclassification error for each k value is :  [0.126 0.14  0.107 0.095 0.095 0.095 0.095 0
 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095
 0.095]
```

In [128]:
```
# ============================== KNN with k = optimal_k ==============================
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_tree')

# fitting the model
knn_optimal.fit(X_tr, y_tr)

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

# 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 = 7 is 84.000000%
```

1.4 AVG W2V

In [129]:
```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = [];
# the avg-w2v for each sentence/review is stored in this list
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        try:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
        except:
            pass
    sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
9982
50
```

In [130]:
```
per=int(0.7*len(sent_vectors))
a=sent_vectors[0:per]
b=sent_vectors[per:]
label=review.Score[0:per]
label1=review.Score[per:]
```

In [131]:
```
# perplexity=30
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler(with_mean=False).fit_transform(a)
print(standardized_data.shape)
```

```
(6987, 50)
```

In [132]:
```
# TSNE

from sklearn.manifold import TSNE

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = standardized_data[0:100]


model = TSNE(n_components=2, random_state=0,perplexity=30)
tsne_data = model.fit_transform(data_1000)
```

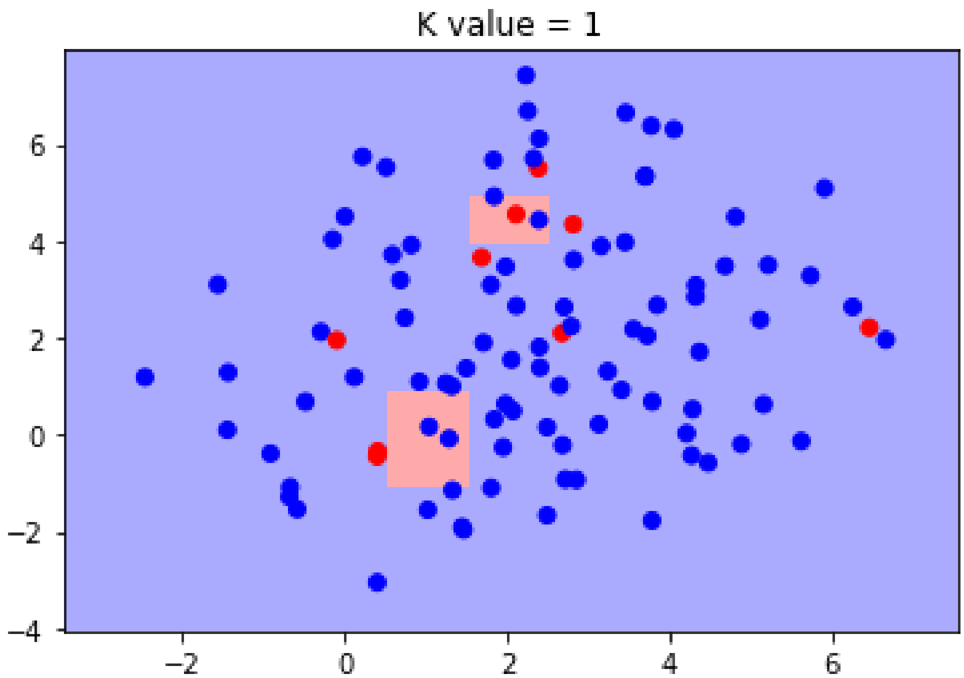
In [133]:
```
from sklearn import datasets, neighbors
from matplotlib.colors import ListedColormap
```

In [134]:
```
def partition(x):
    if x =='negative':
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = label
positiveNegative = actualScore.map(partition)
label= positiveNegative
```

In [137]:
```
knn_comparison(tsne_data, 1)
```

```
[-3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
```

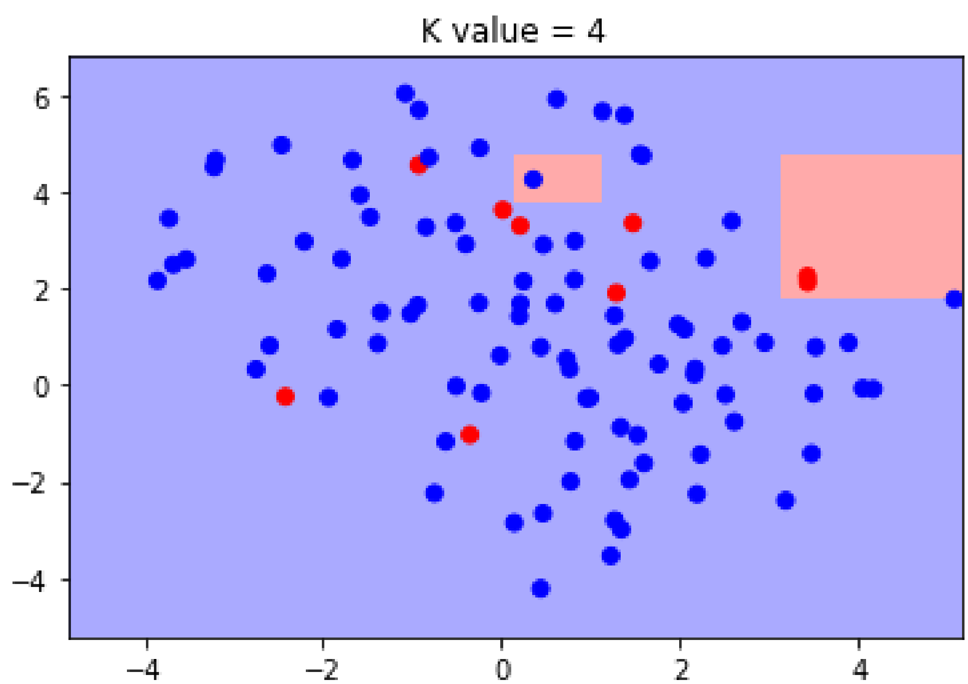
```
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433
 -3.45414567 -2.45414567 -1.45414567 -0.45414567  0.54585433  1.54585433
  2.54585433  3.54585433  4.54585433  5.54585433  6.54585433  7.54585433]
[-4.05260944 -4.05260944 -4.05260944 -4.05260944 -4.05260944 -4.05260944
 -4.05260944 -4.05260944 -4.05260944 -4.05260944 -4.05260944 -4.05260944
 -3.05260944 -3.05260944 -3.05260944 -3.05260944 -3.05260944 -3.05260944
 -3.05260944 -3.05260944 -3.05260944 -3.05260944 -3.05260944 -3.05260944
 -2.05260944 -2.05260944 -2.05260944 -2.05260944 -2.05260944 -2.05260944
 -2.05260944 -2.05260944 -2.05260944 -2.05260944 -2.05260944 -2.05260944
 -1.05260944 -1.05260944 -1.05260944 -1.05260944 -1.05260944 -1.05260944
 -1.05260944 -1.05260944 -1.05260944 -1.05260944 -1.05260944 -1.05260944
 -0.05260944 -0.05260944 -0.05260944 -0.05260944 -0.05260944 -0.05260944
 -0.05260944 -0.05260944 -0.05260944 -0.05260944 -0.05260944 -0.05260944
  0.94739056  0.94739056  0.94739056  0.94739056  0.94739056  0.94739056
  0.94739056  0.94739056  0.94739056  0.94739056  0.94739056  0.94739056
  1.94739056  1.94739056  1.94739056  1.94739056  1.94739056  1.94739056
  1.94739056  1.94739056  1.94739056  1.94739056  1.94739056  1.94739056
  2.94739056  2.94739056  2.94739056  2.94739056  2.94739056  2.94739056
  2.94739056  2.94739056  2.94739056  2.94739056  2.94739056  2.94739056
  3.94739056  3.94739056  3.94739056  3.94739056  3.94739056  3.94739056
  3.94739056  3.94739056  3.94739056  3.94739056  3.94739056  3.94739056
  4.94739056  4.94739056  4.94739056  4.94739056  4.94739056  4.94739056
  4.94739056  4.94739056  4.94739056  4.94739056  4.94739056  4.94739056
  5.94739056  5.94739056  5.94739056  5.94739056  5.94739056  5.94739056
  5.94739056  5.94739056  5.94739056  5.94739056  5.94739056  5.94739056
  6.94739056  6.94739056  6.94739056  6.94739056  6.94739056  6.94739056
  6.94739056  6.94739056  6.94739056  6.94739056  6.94739056  6.94739056
  7.94739056  7.94739056  7.94739056  7.94739056  7.94739056  7.94739056
  7.94739056  7.94739056  7.94739056  7.94739056  7.94739056  7.94739056]
(156, 2)
```



In [129]: knn_comparison(tsne_data, 4)

```
[-4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298
 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702
  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298
 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702
  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298
 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702
  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298
 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702
  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298
  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702
 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298
 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702
  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298
 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702
  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298
 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702
  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298
 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702
  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298
  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702
 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
```

```
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702]
[-5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911
 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -4.21001911
 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -4.21001911
 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -3.21001911 -3.21001911
 -3.21001911 -3.21001911 -3.21001911 -3.21001911 -3.21001911 -3.21001911
 -3.21001911 -3.21001911 -3.21001911 -2.21001911 -2.21001911 -2.21001911
 -2.21001911 -2.21001911 -2.21001911 -2.21001911 -2.21001911 -2.21001911
 -2.21001911 -2.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911
 -1.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911
 -1.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911
 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911
  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089
  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089  1.78998089
  1.78998089  1.78998089  1.78998089  1.78998089  1.78998089  1.78998089
  1.78998089  1.78998089  1.78998089  1.78998089  2.78998089  2.78998089
  2.78998089  2.78998089  2.78998089  2.78998089  2.78998089  2.78998089
  2.78998089  2.78998089  2.78998089  3.78998089  3.78998089  3.78998089
  3.78998089  3.78998089  3.78998089  3.78998089  3.78998089  3.78998089
  3.78998089  3.78998089  4.78998089  4.78998089  4.78998089  4.78998089
  4.78998089  4.78998089  4.78998089  4.78998089  4.78998089  4.78998089
  4.78998089  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089
  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089
  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089
  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089]
(143, 2)
```



```
In [138]: from sklearn.preprocessing import StandardScaler
          standardized_data = StandardScaler(with_mean=False).fit_transform(b)
          print(standardized_data.shape)
```

```
(2995, 50)
```

```
In [139]: # TSNE

          from sklearn.manifold import TSNE

          # Picking the top 1000 points as TSNE takes a lot of time for 15K points
          data_1000 = standardized_data[0:100]


          model = TSNE(n_components=2, random_state=0,perplexity=30)
          tsne_data1 = model.fit_transform(data_1000)
```

```
In [140]: actualScore = label1
          positiveNegative = actualScore.map(partition)
          label1= positiveNegative
```

```
In [141]: X = np.array(tsne_data) # end index is exclusive
          y = np.array(label[0:100])
```

```
In [160]: from sklearn.feature_extraction.text import CountVectorizer


          # split the data set into train and test
          X_test, y_test = np.array(tsne_data1), np.array(label1[0:100])

          # 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, y, test_size=0.3)

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

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

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

              # evaluate CV accuracy
              acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
```

```
            print('\nCV accuracy for k = %d is %d%%' % (i, acc))

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

```
CV accuracy for k = 1 is 86%

CV accuracy for k = 3 is 90%

CV accuracy for k = 5 is 90%

CV accuracy for k = 7 is 90%

CV accuracy for k = 9 is 90%

CV accuracy for k = 11 is 90%

CV accuracy for k = 13 is 90%

CV accuracy for k = 15 is 90%

CV accuracy for k = 17 is 90%

CV accuracy for k = 19 is 90%

CV accuracy for k = 21 is 90%

CV accuracy for k = 23 is 90%

CV accuracy for k = 25 is 90%

CV accuracy for k = 27 is 90%

CV accuracy for k = 29 is 90%

****Test accuracy for k = 1 is 82%
```

```
In [161]: knn = KNeighborsClassifier(5)
          knn.fit(X_tr,y_tr)
          pred = knn.predict(X_test)
          acc = accuracy_score(y_test, pred, normalize=True) * float(100)
          print('\n****Test accuracy for k = 5 is %d%%' % (acc))
```

****Test accuracy for k = 5 is 86%

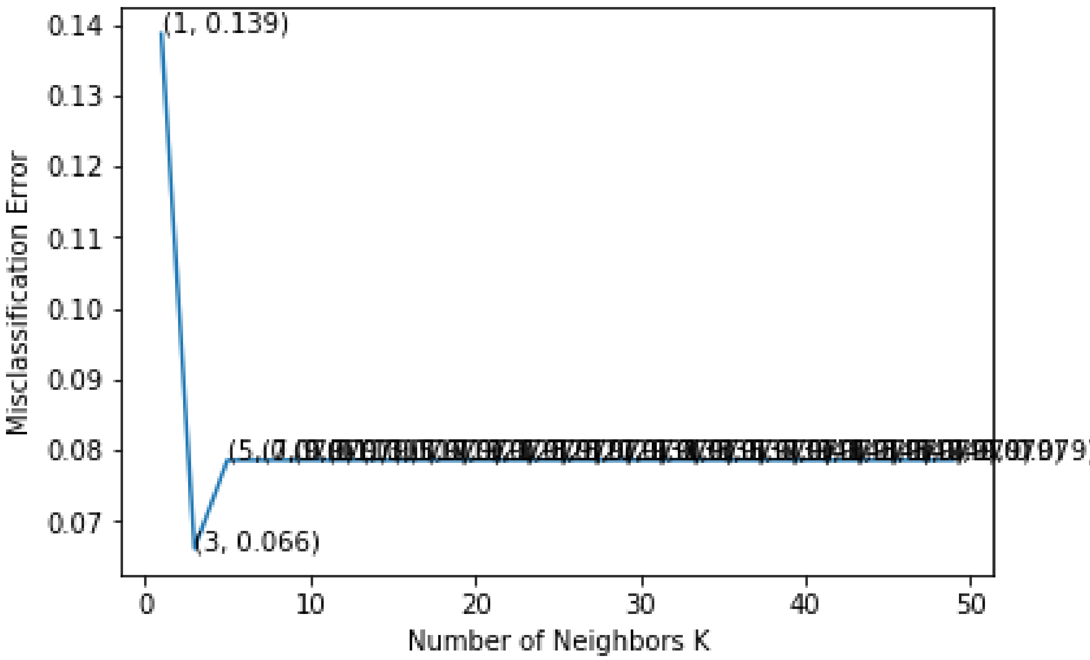
1.4.1 Using 10 fold

In [162]:
```
# 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, X_tr, y_tr, 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))
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valid
```

```
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
```

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




```
the misclassification error for each k value is :  [0.139 0.066 0.079 0.079 0.079 0.079 0.079 0
 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079 0.079
 0.079]
```

```
In [163]: # ============================== 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_tr, y_tr)

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

          # 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 = 3 is 86.000000%
```

1.4.2 Using kd_tree

```
In [164]: from sklearn.feature_extraction.text import CountVectorizer
```

```python
# split the data set into train and test
X_test, y_test = np.array(tsne_data1), np.array(label1[0:100])

# 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, y, test_size=0.3)

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

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

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

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

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

```
CV accuracy for k = 1 is 83%

CV accuracy for k = 3 is 80%

CV accuracy for k = 5 is 80%

CV accuracy for k = 7 is 80%

CV accuracy for k = 9 is 80%

CV accuracy for k = 11 is 80%

CV accuracy for k = 13 is 80%

CV accuracy for k = 15 is 80%

CV accuracy for k = 17 is 80%

CV accuracy for k = 19 is 80%
```

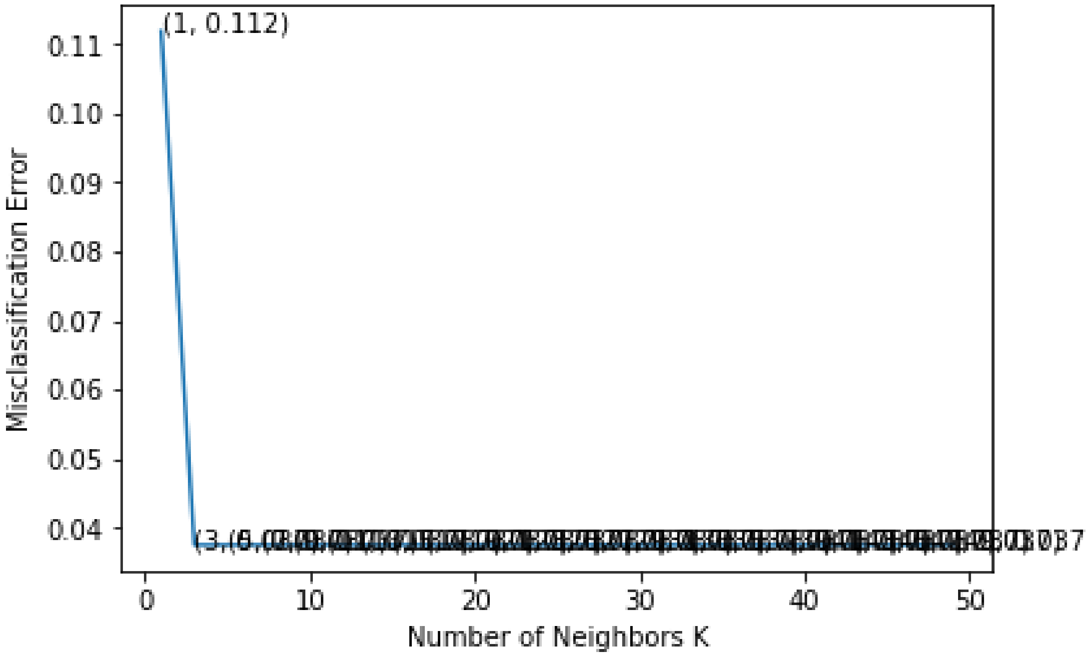
```
CV accuracy for k = 21 is 80%

CV accuracy for k = 23 is 80%

CV accuracy for k = 25 is 80%

CV accuracy for k = 27 is 80%

CV accuracy for k = 29 is 80%

****Test accuracy for k = 1 is 84%
```

```
In [165]: # 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,algorithm='kd_tree')
              scores = cross_val_score(knn, X_tr, y_tr, 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))
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
```

```
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
```

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




```
the misclassification error for each k value is :  [0.112 0.037 0.037 0.037 0.037 0.037 0.037 0
 0.037 0.037 0.037 0.037 0.037 0.037 0.037 0.037 0.037 0.037 0.037 0.037
 0.037]
```

In [166]:

```python
# ============================== KNN with k = optimal_k ==============================
# instantiate learning model k = optimal_k
knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_tree')

# fitting the model
knn_optimal.fit(X_tr, y_tr)

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

# 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 = 3 is 86.000000%
```

1.5 TF-IDF W2V

In [138]:
```
# TF-IDF weighted Word2Vec
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [139]:
```
tfidf_feat = tf_idf_vect.get_feature_names()
```

In [140]:
```
tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this
row=0;
for sent in list_of_sent: # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        try:
            vec = w2v_model.wv[word]
            #print(tfidf_feat.index(word))
            # obtain the tf_idfidf of a word in a sentence/review
            tf_idf = final_tf_idf[row, tfidf_feat.index(word)]
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
        except:
            pass
    sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

In [141]:
```
per=int(0.7*len(sent_vectors))
a=sent_vectors[0:per]
b=sent_vectors[per:]
label=review.Score[0:per]
label1=review.Score[per:]
```

In [142]:
```
# perplexity=30
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler(with_mean=False).fit_transform(a)
print(standardized_data.shape)
```

```
(6987, 50)
```

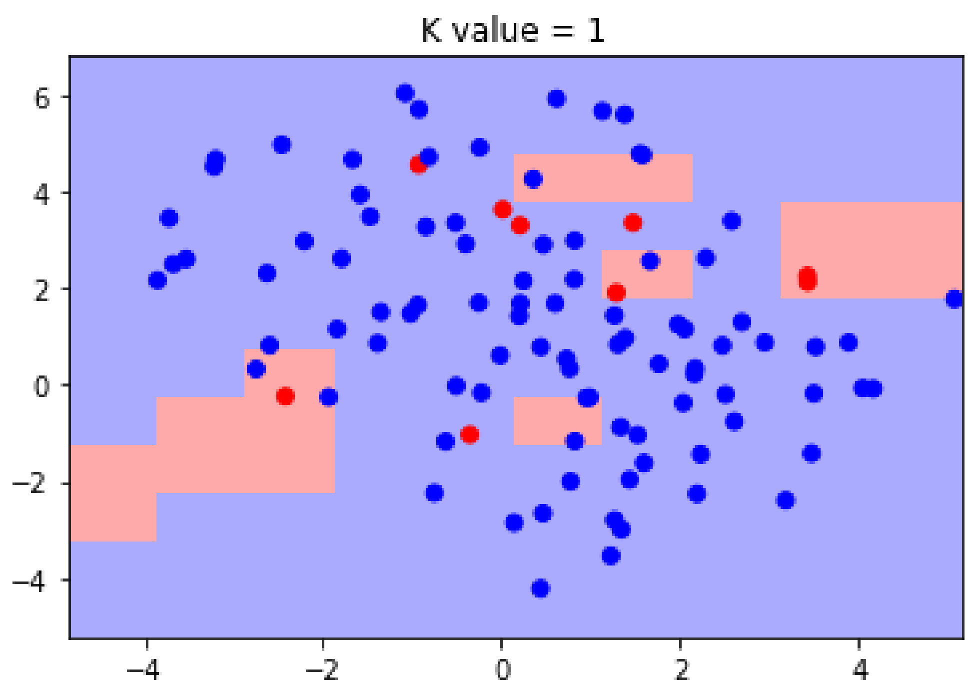
In [143]:
```
# TSNE

from sklearn.manifold import TSNE

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = standardized_data[0:100]


model = TSNE(n_components=2, random_state=0,perplexity=30)
tsne_data = model.fit_transform(data_1000)
```

```
In [144]: def partition(x):
              if x =='negative':
                  return 0
              return 1

          #changing reviews with score less than 3 to be positive and vice-versa
          actualScore = label
          positiveNegative = actualScore.map(partition)
          label= positiveNegative
```

```
In [145]: knn_comparison(tsne_data, 1)
```

```
[-4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298
 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702
  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298
 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702
  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298
 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702
  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298
 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702
  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298
  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702
 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298
 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702
  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298
 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702
  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298
 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702
  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298
 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702
  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298
  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702
 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702]
[-5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911
 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -4.21001911
 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -4.21001911
 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -3.21001911 -3.21001911
 -3.21001911 -3.21001911 -3.21001911 -3.21001911 -3.21001911 -3.21001911
 -3.21001911 -3.21001911 -3.21001911 -2.21001911 -2.21001911 -2.21001911
 -2.21001911 -2.21001911 -2.21001911 -2.21001911 -2.21001911 -2.21001911
 -2.21001911 -2.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911
 -1.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911
 -1.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911
 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911
  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089
```

```
  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089  1.78998089
  1.78998089  1.78998089  1.78998089  1.78998089  1.78998089  1.78998089
  1.78998089  1.78998089  1.78998089  1.78998089  2.78998089  2.78998089
  2.78998089  2.78998089  2.78998089  2.78998089  2.78998089  2.78998089
  2.78998089  2.78998089  2.78998089  3.78998089  3.78998089  3.78998089
  3.78998089  3.78998089  3.78998089  3.78998089  3.78998089  3.78998089
  3.78998089  3.78998089  4.78998089  4.78998089  4.78998089  4.78998089
  4.78998089  4.78998089  4.78998089  4.78998089  4.78998089  4.78998089
  4.78998089  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089
  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089
  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089
  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089]
(143, 2)
```

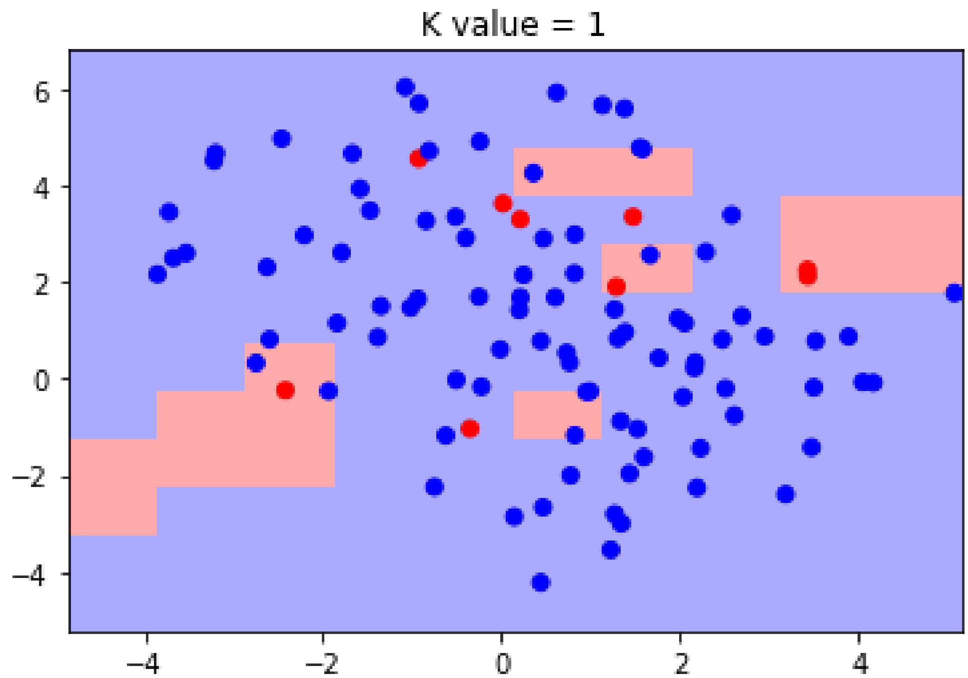


1.5.1 using kd tree

```
In [157]: knn_comparison(tsne_data, 1)
```

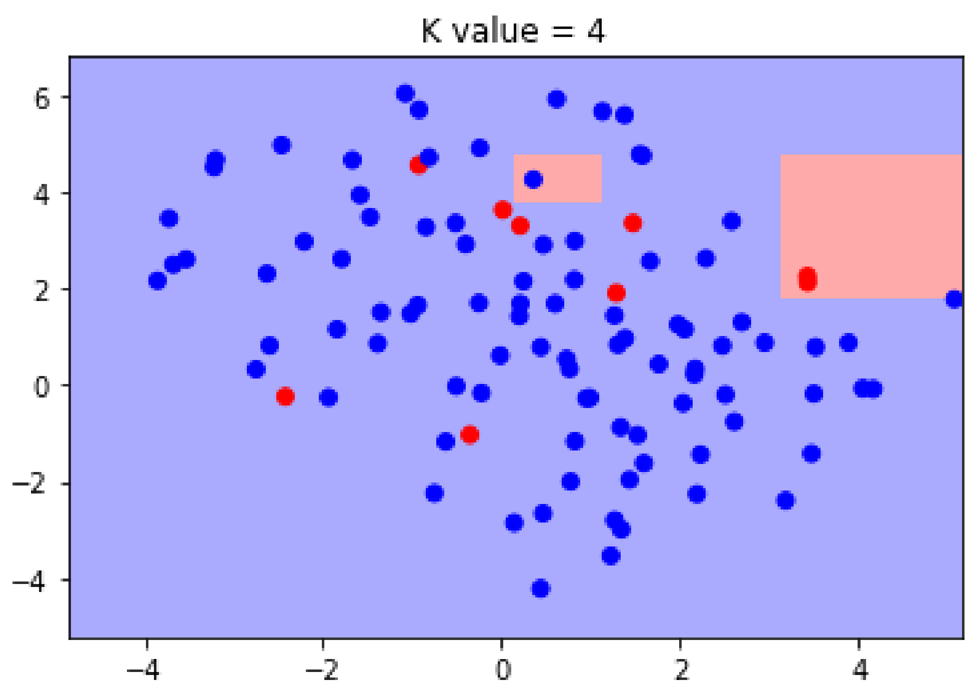
```
[-4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298
 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702
  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298
 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702
  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298
```

```
 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702
  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298
 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702
  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298
  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702
 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298
 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702
  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298
 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702
  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298
 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702
  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298
 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702
  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298
  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702
 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702]
[-5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911
 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -4.21001911
 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -4.21001911
 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -3.21001911 -3.21001911
 -3.21001911 -3.21001911 -3.21001911 -3.21001911 -3.21001911 -3.21001911
 -3.21001911 -3.21001911 -3.21001911 -2.21001911 -2.21001911 -2.21001911
 -2.21001911 -2.21001911 -2.21001911 -2.21001911 -2.21001911 -2.21001911
 -2.21001911 -2.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911
 -1.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911
 -1.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911
 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911
  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089
  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089  1.78998089
  1.78998089  1.78998089  1.78998089  1.78998089  1.78998089  1.78998089
  1.78998089  1.78998089  1.78998089  1.78998089  2.78998089  2.78998089
  2.78998089  2.78998089  2.78998089  2.78998089  2.78998089  2.78998089
  2.78998089  2.78998089  2.78998089  3.78998089  3.78998089  3.78998089
  3.78998089  3.78998089  3.78998089  3.78998089  3.78998089  3.78998089
  3.78998089  3.78998089  4.78998089  4.78998089  4.78998089  4.78998089
  4.78998089  4.78998089  4.78998089  4.78998089  4.78998089  4.78998089
  4.78998089  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089
  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089
  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089
  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089]
(143, 2)
```




```
In [146]: knn_comparison(tsne_data, 4)

[-4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298
 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702
  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298
 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702
  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298
 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702
  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298
 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702
  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298
  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702
 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298
 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702
  2.14667702  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298
 -2.85332298 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702
  3.14667702  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298
 -1.85332298 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702
  4.14667702  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298
 -0.85332298  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702
  5.14667702 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298
  0.14667702  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702
 -4.85332298 -3.85332298 -2.85332298 -1.85332298 -0.85332298  0.14667702
```

```
  1.14667702  2.14667702  3.14667702  4.14667702  5.14667702]
[-5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911
 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -5.21001911 -4.21001911
 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -4.21001911
 -4.21001911 -4.21001911 -4.21001911 -4.21001911 -3.21001911 -3.21001911
 -3.21001911 -3.21001911 -3.21001911 -3.21001911 -3.21001911 -3.21001911
 -3.21001911 -3.21001911 -3.21001911 -2.21001911 -2.21001911 -2.21001911
 -2.21001911 -2.21001911 -2.21001911 -2.21001911 -2.21001911 -2.21001911
 -2.21001911 -2.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911
 -1.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911 -1.21001911
 -1.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911
 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911 -0.21001911
  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089
  0.78998089  0.78998089  0.78998089  0.78998089  0.78998089  1.78998089
  1.78998089  1.78998089  1.78998089  1.78998089  1.78998089  1.78998089
  1.78998089  1.78998089  1.78998089  1.78998089  2.78998089  2.78998089
  2.78998089  2.78998089  2.78998089  2.78998089  2.78998089  2.78998089
  2.78998089  2.78998089  2.78998089  3.78998089  3.78998089  3.78998089
  3.78998089  3.78998089  3.78998089  3.78998089  3.78998089  3.78998089
  3.78998089  3.78998089  4.78998089  4.78998089  4.78998089  4.78998089
  4.78998089  4.78998089  4.78998089  4.78998089  4.78998089  4.78998089
  4.78998089  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089
  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089  5.78998089
  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089
  6.78998089  6.78998089  6.78998089  6.78998089  6.78998089]
(143, 2)
```



In [147]:
```
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler(with_mean=False).fit_transform(b)
print(standardized_data.shape)
```

```
(2995, 50)
```

In [148]:
```
# TSNE

from sklearn.manifold import TSNE

# Picking the top 1000 points as TSNE takes a lot of time for 15K points
data_1000 = standardized_data[0:100]


model = TSNE(n_components=2, random_state=0,perplexity=30)
tsne_data1 = model.fit_transform(data_1000)
```

In [149]:
```
actualScore = label1
positiveNegative = actualScore.map(partition)
label1= positiveNegative
```

In [150]:
```
X = np.array(tsne_data) # end index is exclusive
y = np.array(label[0:100])
```

In [182]:
```
from sklearn.feature_extraction.text import CountVectorizer


# split the data set into train and test
X_test, y_test = np.array(tsne_data1), np.array(label1[0:100])

# 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, y, test_size=0.3)

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

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

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

    # evaluate CV accuracy
    acc = accuracy_score(y_cv, pred, normalize=True) * float(100)
```

```
        print('\nCV accuracy for k = %d is %d%%' % (i, acc))

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

```

CV accuracy for k = 1 is 90%

CV accuracy for k = 3 is 86%

CV accuracy for k = 5 is 86%

CV accuracy for k = 7 is 86%

CV accuracy for k = 9 is 86%

CV accuracy for k = 11 is 86%

CV accuracy for k = 13 is 86%

CV accuracy for k = 15 is 86%

CV accuracy for k = 17 is 86%

CV accuracy for k = 19 is 86%

CV accuracy for k = 21 is 86%

CV accuracy for k = 23 is 86%

CV accuracy for k = 25 is 86%

CV accuracy for k = 27 is 86%

CV accuracy for k = 29 is 86%

****Test accuracy for k = 1 is 80%
```

In [183]:
```
knn = KNeighborsClassifier(5)
knn.fit(X_tr,y_tr)
pred = knn.predict(X_test)
acc = accuracy_score(y_test, pred, normalize=True) * float(100)
print('\n****Test accuracy for k = 5 is %d%%' % (acc))
```

****Test accuracy for k = 5 is 86%

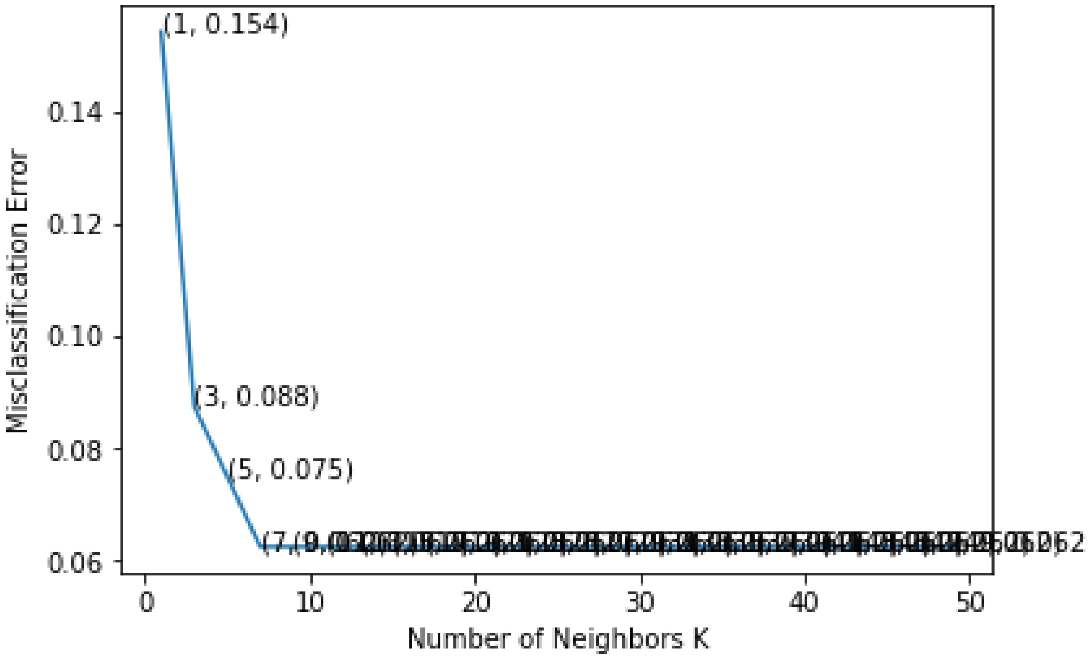
1.5.2 using 10 fold

In [184]:
```
# 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, X_tr, y_tr, 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))
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_vali
```

```
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
```

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




```
the misclassification error for each k value is :  [0.154 0.088 0.075 0.062 0.062 0.062 0.062 0
 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062 0.062
 0.062]
```

```
In [185]: # ============================== 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_tr, y_tr)

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

          # 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 = 7 is 86.000000%
```

1.5.3 using kd-tree

```
In [187]: from sklearn.feature_extraction.text import CountVectorizer
```

```
# split the data set into train and test
X_test, y_test = np.array(tsne_data1), np.array(label1[0:100])

# 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, y, test_size=0.3)

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

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

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

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

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

```
CV accuracy for k = 1 is 86%

CV accuracy for k = 3 is 86%

CV accuracy for k = 5 is 93%

CV accuracy for k = 7 is 93%

CV accuracy for k = 9 is 93%

CV accuracy for k = 11 is 93%

CV accuracy for k = 13 is 93%

CV accuracy for k = 15 is 93%

CV accuracy for k = 17 is 93%

CV accuracy for k = 19 is 93%
```

```
CV accuracy for k = 21 is 93%

CV accuracy for k = 23 is 93%

CV accuracy for k = 25 is 93%

CV accuracy for k = 27 is 93%

CV accuracy for k = 29 is 93%

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

```
In [188]: knn = KNeighborsClassifier(5,algorithm='kd_tree')
          knn.fit(X_tr,y_tr)
          pred = knn.predict(X_test)
          acc = accuracy_score(y_test, pred, normalize=True) * float(100)
          print('\n****Test accuracy for k = 5 is %d%%' % (acc))
```

```
****Test accuracy for k = 5 is 86%
```

10-fold

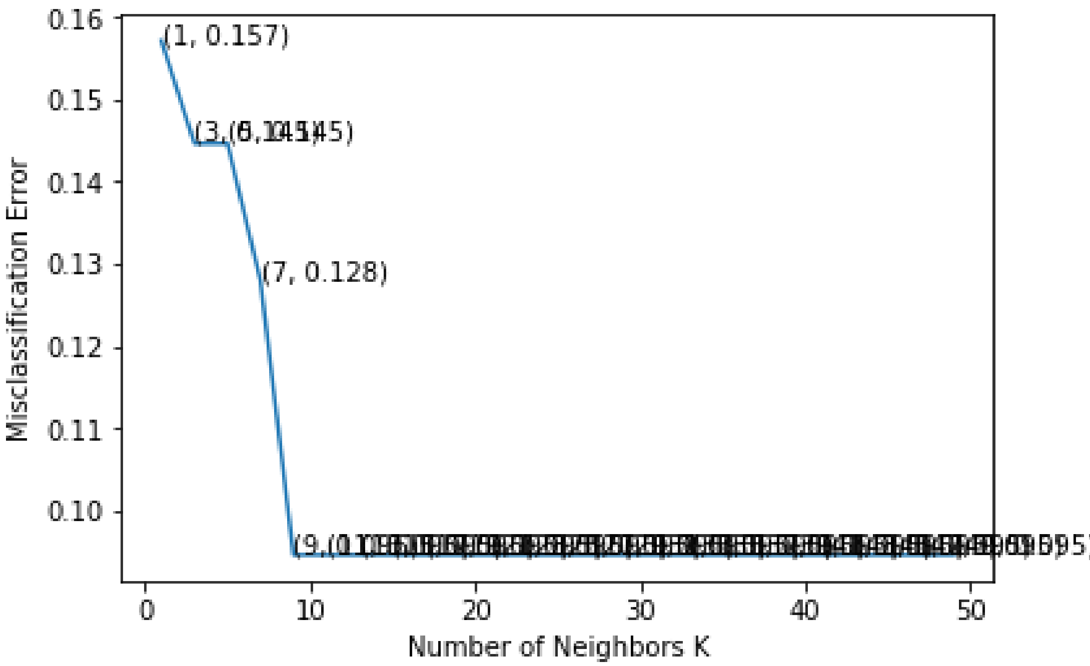
```
In [190]: # creating odd list of K for KNN
          myList = list(range(0,50))
          neighbors = list(filter(lambda x: x % 2 != 0, myList))

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

          # perform 10-fold cross validation
          for k in neighbors:
              knn = KNeighborsClassifier(n_neighbors=k,algorithm='kd_tree')
              scores = cross_val_score(knn, X_tr, y_tr, 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)
```

```python
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))
```

```
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
```

```
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
c:\users\vishal\appdata\local\programs\python\python36-32\lib\site-packages\sklearn\cross_valic
  % (min_labels, self.n_folds)), Warning)
```

```
The optimal number of neighbors is 9.
```




```
the misclassification error for each k value is :  [0.157 0.145 0.145 0.128 0.095 0.095 0.095 0
 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095 0.095
 0.095]
```

```
In [193]: # ============================== KNN with k = optimal_k ==============================
          # instantiate learning model k = optimal_k
          knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k,algorithm='kd_tree')
```

```python
# fitting the model
knn_optimal.fit(X_tr, y_tr)

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

# 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 = 9 is 86.000000%
```