In [1]:
```
%matplotlib inline
```

Python Code for Digit Recognition using K-Nearest Neighbors

Author: Sri Sravya Tirupachur Comerica
Date: 9/12/2019
Instructor: Dr. Mark Albert
Description: The code implements K-NN for the given data set by using euclidean distance as the distance metric to find the nearest neighbors and later predicts the class based on the value of K and the most frequent classes of its neighbors.

In [2]:
```
# import packages
from __future__ import print_function
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn import datasets
from sklearn import metrics
from skimage import exposure
import numpy as np
import math
import cv2
import matplotlib.pyplot as plt

mnist = datasets.load_digits()

#splitting the train data and the test data into 50% each
(trainData, testData, trainLabels, testLabels) = train_test_split(np.array(mni
st.data),
mnist.target, test_size=0.50, random_state=4)
```

In [3]:
```
#intializing the k-values and the corresponding arrays to store the accuracies
kVals = [1, 3, 5, 100, 500]
accuracies = []
test = len(testData)
train = len(trainData)
predictions = []
highest = 0
best_k = 0
Test_Distances = []

#finds the euclidean distance for each vector in a test set with the train set
for i in range(0, test):
    distances = []
    j=0
    while j < train:
            size = 0
            euclidean = 0
            while size < 64:
                euclidean = euclidean + ((testData[i][size] - trainData[j][siz
e]) ** 2)
                size = size+1
            euclidean = math.sqrt(euclidean)
            distances.append(euclidean)
            j = j+1
#sorts the found dustances in ascending order
    sorted_distances = np.argsort(distances)
    Test_Distances.append(sorted_distances)
```

In [22]:
```
# loop over various values of `k` for the k-Nearest Neighbor classifier
for k in kVals:
    predictions1 = []
    test_len=0
    while test_len<len(Test_Distances):
        x = 0
        t = k
        pred_class = []
        while t > 0:
            pred_class.append(trainLabels[Test_Distances[test_len][x]])
            x = x+1
            t = t-1
        res = max(set(pred_class), key = pred_class.count)
        predictions1.append(res)
        test_len=test_len+1
#gets the accuracy of the predictions made on the given value k
    g = accuracy_score(testLabels, predictions1)
    print("\n k=%d, accuracy=%.2f%%" % (k, g * 100))
    print("\n Confusion Matrix:")
    print(confusion_matrix(testLabels,predictions1))
#stores the highest accuracy recorded up until that point
    accuracy_percentage = g * 100
    if accuracy_percentage > highest:
        highest = accuracy_percentage
        predictions = predictions1
        best_k = k
    accuracies.append(g*100)

print("\n The K with Maximum Accuracy k=%d, Accuracy=%.2f%%" % (best_k, highes
t))

print("\n Evaluating the Performance of k=%d" %best_k)
print(classification_report(testLabels, predictions))
```

```
k=1, accuracy=98.44%

Confusion Matrix:
[[91  0  0  0  0  0  0  0  0  0]
 [ 0 84  0  0  0  0  0  0  0  0]
 [ 0  0 86  1  0  0  0  0  0  0]
 [ 0  0  0 97  0  0  0  0  0  0]
 [ 0  2  0  0 88  0  0  0  0  0]
 [ 0  0  0  0  0 89  1  0  0  2]
 [ 0  0  0  0  0  0 92  0  1  0]
 [ 0  0  0  0  0  0  0 92  0  0]
 [ 0  3  0  0  0  0  0  0 79  0]
 [ 0  1  0  1  0  0  0  0  2 87]]

k=3, accuracy=98.44%

Confusion Matrix:
[[91  0  0  0  0  0  0  0  0  0]
 [ 0 84  0  0  0  0  0  0  0  0]
 [ 0  0 86  0  0  0  0  1  0  0]
 [ 0  0  0 95  0  1  0  1  0  0]
 [ 0  1  0  0 89  0  0  0  0  0]
 [ 0  0  0  0  0 90  1  0  0  1]
 [ 0  0  0  0  0  0 92  0  1  0]
 [ 0  0  0  0  0  0  0 92  0  0]
 [ 0  3  0  0  0  0  0  0 79  0]
 [ 0  2  0  1  0  0  0  0  1 87]]

k=5, accuracy=98.00%

Confusion Matrix:
[[91  0  0  0  0  0  0  0  0  0]
 [ 0 84  0  0  0  0  0  0  0  0]
 [ 0  0 86  0  0  0  0  1  0  0]
 [ 0  0  0 95  0  1  0  1  0  0]
 [ 0  1  0  0 89  0  0  0  0  0]
 [ 0  0  0  0  1 89  1  0  0  1]
 [ 0  0  0  0  0  0 92  0  1  0]
 [ 0  0  0  0  0  0  0 92  0  0]
 [ 0  5  0  0  0  0  0  0 77  0]
 [ 0  3  0  1  0  0  0  0  1 86]]

k=100, accuracy=89.99%

Confusion Matrix:
[[91  0  0  0  0  0  0  0  0  0]
 [ 0 67 11  1  0  0  1  0  1  3]
 [ 0  4 77  2  0  0  0  1  2  1]
 [ 1  0  0 84  0  1  0  4  2  5]
 [ 0  3  0  0 85  0  0  1  1  0]
 [ 0  0  0  0  1 81  1  0  0  9]
 [ 0  0  0  0  0  0 92  0  1  0]
 [ 0  0  0  0  0  0  0 92  0  0]
 [ 0  5  2  1  0  1  0  2 67  4]
 [ 1  5  0  1  3  2  0  3  3 73]]

k=500, accuracy=67.85%
```

```
 Confusion Matrix:
[[86  0  0  0  0  1  4  0  0  0]
 [ 1 26 15  1  1  0  5  0 26  9]
 [ 3  1 53  7  0  0  4  1 15  3]
 [ 2  0  8 73  0  1  1  1  7  4]
 [ 5  2  0  0 67  0 10  2  4  0]
 [ 9  0  0  1  1 68  3  0  1  9]
 [19  1  0  0  1  0 72  0  0  0]
 [ 0  0  0  0  5  5  0 51 31  0]
 [ 2  4  1  1  1  3  2  0 61  7]
 [22  1  2  2  3  2  0  1  5 53]]

 The K with Maximum Accuracy k=1, Accuracy=98.44%

 Evaluating the Performance of k=1
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        91
           1       0.93      1.00      0.97        84
           2       1.00      0.99      0.99        87
           3       0.98      1.00      0.99        97
           4       1.00      0.98      0.99        90
           5       1.00      0.97      0.98        92
           6       0.99      0.99      0.99        93
           7       1.00      1.00      1.00        92
           8       0.96      0.96      0.96        82
           9       0.98      0.96      0.97        91

    accuracy                           0.98       899
   macro avg       0.98      0.98      0.98       899
weighted avg       0.98      0.98      0.98       899
```

Error Evaluation:

1. Based on the given confusion matrix for k=1, 5 is being predicted as 9 about 2 times which matches my intutions since 5 and 9 are similar looking numbers and might have a very closely similar distance values because of which sometimes the model tends to predict 5 as 9.
2. Also 8 is being predicted as 9 about 2 times since these two numbers are also very similar to each other in shape and the model tends to make a wrong preciction of these numbers, just like 2 being predicted as 4.
3. Based on the accuracies found from different values of K, k=1 tends to be be the k with highest accuracy and k=500 tends to be the lowest which might be the reason why they say the higher the value of k the more overfitted a model gets, k=1, 3 and 5 have almost the same amount of accuracies proving that the model does a better job when k is in one these values.