**CS 7267- HW 1 Seoyoon Park**

1. Import packages and Open files

|  |
| --- |
| **import** **csv**  **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  tmpfile = open("MNIST\_training.csv", "rt", encoding='utf-8')  training = np.array(list(csv.reader(tmpfile)))  trainingY = np.array(training[1:, 0], dtype=int)  trainingX = np.array(training[1:, 1:], dtype=int)  tmpfile = open("MNIST\_test.csv", "rt", encoding='utf-8')  test = np.array(list(csv.reader(tmpfile)))  testY = np.array(test[1:, 0], dtype=int)  testX = np.array(test[1:, 1:], dtype=int) |

1. Calculate distance between test data and training data

|  |
| --- |
| Distance = np.zeros((len(testX), len(trainingX)))  **for** i **in** range (len(testX)):  **for** j **in** range (len(trainingX)):  Distance[i, j] = (np.sum((testX[i]-trainingX[j])\*\*2))\*\*(1/2) |

Matrix ‘Distance’ indicates Euclidean distance between i-th test sample and j-th training sample as Distance[i, j].

1. Make some matrix to help prediction

|  |
| --- |
| *# count the number of each label of KNN of test samples.*  KNNlabel = np.zeros((len(testY), 10), dtype=int)  *# predict which label testY is.*  Prediction = np.zeros((len(testY), 1), dtype=int)  *# accuracy of KNN algorithm based on size of K.*  Accuracy = np.zeros((95, 1), dtype=np.float16) |

Matrix ‘KNNlabel’ indicates the number of K-nearest training samples’ label (from test samples to training samples) based on K. For example, when the labels of 5 nearest neighbor of i-th sample are 2, 5, 3, 3, 2 respectively, KNNlabel[i] is [0, 0, 1, 3, 0, 1, 0, 0, 0, 0]. This matrix varies depending on the number K.

Matrix ‘Prediction’ represents the prediction of i-th test sample’s label in Prediction[i]. From the matrix ‘KNNlabel’, it is possible to predict test sample’s label. This matrix varies depending on the number K.

Matrix ‘Accuracy’ represents the accuracy of KNN algorithm when K equals i+1. Accuracy[i] is calculated by this formula:

The number of correctly classified on my prediction / The number of total test data

For each K, I compared matrix ‘Prediction’ with the label of test data and if each elements is identical, I counted it as “correctly classified.” From this, matrix ‘Accuracy’ is calculated.

1. Calculate Accuracy based on the number of K

|  |
| --- |
| Distance2 = np.copy(Distance) |

First, copy ‘Distance’ to ‘Distance2’. ‘Distance’ is original distance matrix and ‘Distance2’ is for getting prediction and accuracy. The values of ‘Distance2’ will be modified.

|  |
| --- |
| **for** k **in** range(1, 96): *# k is the number of nearest neighbor*  **for** i **in** range(0, 50): *# for i-th sample in test data*  **for** j **in** range(0, k):  index = np.argmin(np.array(Distance2[i]))  label = trainingY[index]    *# count which label i-th test sample is predicted as*  KNNlabel[i, label] = (KNNlabel[i, label]) + 1    *# exclude this sample in training data for next loop*  Distance2[i, index] = np.max(Distance2[i])    Prediction[i] = np.argmax(np.array(KNNlabel[i]))    *# the number of correctly classificated*  correct = 0  **for** x **in** range(0, 50):  **if** int(Prediction[x]) == int(testY[x]):  correct = correct + 1    *# calculate Accuracy*  Accuracy[k-1, 0] = format((correct/50), '.2f')    *# put the original distance in Distance2 array*  Distance2 = np.copy(Distance)    **print**('k = ', k)  **print**('correct = ', correct) |

I set the range of K from 1 to 96. According to my speculation, in the ideal case, a test sample’s K-nearest neighbors have the same label and the prediction is correct. In this case, the maximum K is 95 because the number of training data of each classes is 95. It is the reason why I set the maximum K as 95.

The code for calculating ‘Prediction’ and ‘Accuracy’ is iterated over the number of K.

For each K, I got the K-nearest neighbors and predict which label the i-th test data is. From this prediction, I got the accuracy of my prediction based on K.

1. Plot the result

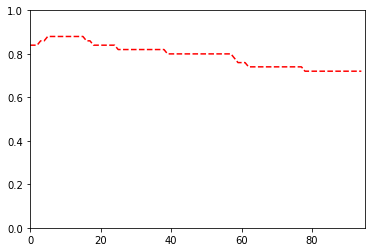
|  |
| --- |
| **print**(Accuracy)  **print**('optimum K = ', np.argmax(Accuracy))  **print**('Accuracy = ', np.max(Accuracy))  plt.plot(Accuracy, 'r--')  plt.axis([0, 95, 0, 1])  plt.show() |

We can see the result of the maximum accuracy and the optimal K.

Here is the result.

optimum K = 5

Accuracy = 0.87988



(x axis = K, y axis = accuracy)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K = 1~20 | K = 21~40 | K = 41~60 | K = 61~80 | K = 81~95 |
| [[ 0.83984375]  [ 0.83984375]  [ 0.83984375]  [ 0.85986328]  [ 0.85986328]  [ 0.87988281]  [ 0.87988281]  [ 0.87988281]  [ 0.87988281]  [ 0.87988281]  [ 0.87988281]  [ 0.87988281]  [ 0.87988281]  [ 0.87988281]  [ 0.87988281]  [ 0.87988281]  [ 0.85986328]  [ 0.85986328]  [ 0.83984375]  [ 0.83984375] | [ 0.83984375]  [ 0.83984375]  [ 0.83984375]  [ 0.83984375]  [ 0.83984375]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.81982422]  [ 0.79980469] | [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.79980469]  [ 0.77978516]  [ 0.75976562] | [ 0.75976562]  [ 0.75976562]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.74023438]  [ 0.72021484]  [ 0.72021484] | [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]  [ 0.72021484]] |

This table represents the accuracy of my prediction based on K.