MTH448 Project1

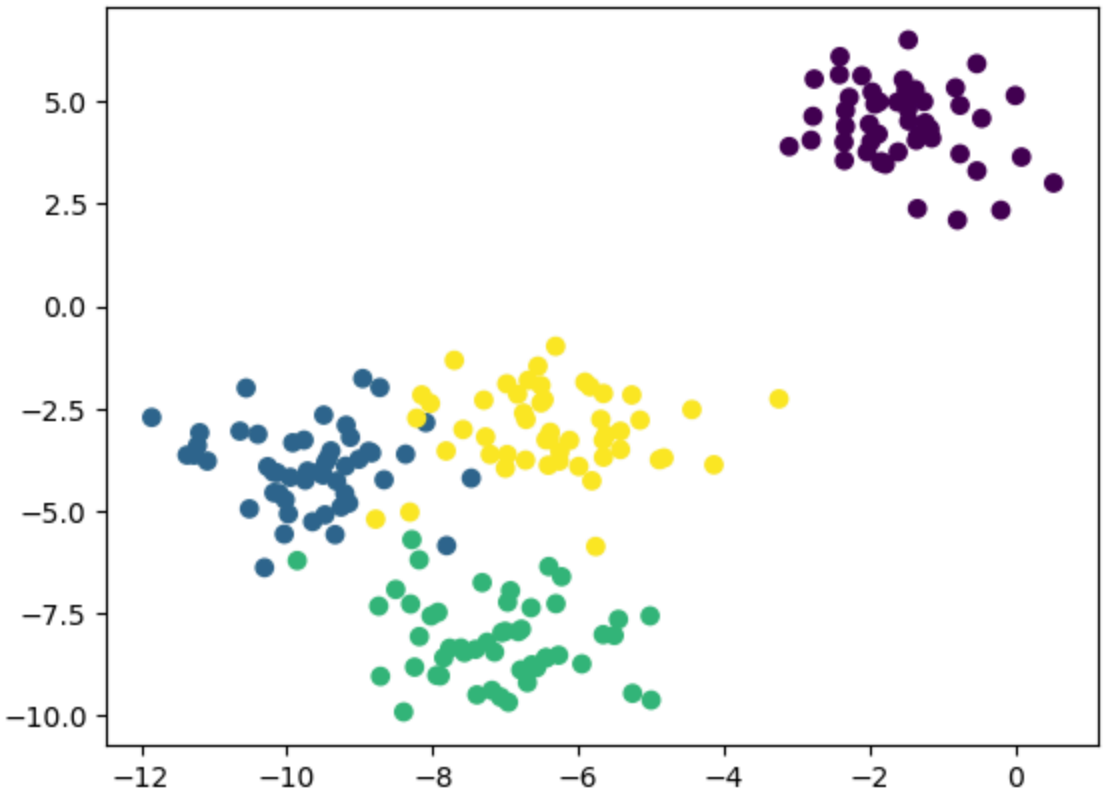
For the porject 1, we will use one algorithm from Machine learning named KNN(K Nearest Neighbors) to classify hand-written digits.Moreover, We will test the algroithm testing by MINST dataset. Hence, before we start, we have to know what is KNN algorithm and what is MINST dataset.

First we need to know what the three letters KNN stand for, KNN is K Nearest Neighbor. Below I will explain how to understand this algorithm with a diagram.

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
```

In [15]:
```
coords,labs = make_blobs(n_samples=200,
                         n_features=2,
                         centers = 4,
                        cluster_std=1,
                        random_state=1)
plt.scatter(coords[:,0],coords[:,1],c=labs)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x2a49be410>
```



We can see four different color dots on the graph, distributed in different places depending on the color. So how do we understand KNN from these different color groups? K nearest neighbor is the simple algorithm to classify by looking at the neighbors or things around k.

First of all, the letter K is just a number that tells the algorithm how many neighbors it can look at.

On the way, we can see that the yellow dots also appear in the blue dots. So let's say we choose the point with coordinates (-8, -2.5), and let's say K = 3, then the algorithm looks for the three nearest neighbors at that coordinate. If there are three neighbors of that coordinate, two of them are yellow and one is blue. Then the algorithm identifies the point as yellow. Conversely, if there are three neighbors, two of which are blue and one of which is yellow, then the algorithm determines that the point is blue.

MINST Database

Once we understand what the KNN algorithm is, we need to understand what the MINST databse is. The MINST database is a collection of digits used by machine learning for training and testing of handwriting recognition. A whole dataset is divided into two parts, one with 60,000 training images and the other with 10,000 test images. Thus, there are a total of 70,000 images with 784 attributes, each of which is a grayscale image of size 28x28 pixels.

Here is to show the image looks like. We have to import the csv.file for MINST databse.

In [16]:
```
import csv
```

In [17]:
```
File = open('mnist_train.csv')
Reader = csv.reader(File)
inside = []
for line in Reader:
    inside.append(line)
File.close()
Data = np.array(inside,dtype=int)
```

In [18]:
```
Data.shape
```

Out[18]:
```
(60000, 785)
```

Data.shape is help us to check how many rows and columns from the databse. Here looks 60,000 rows and 785 columns.

In [19]:
```
Labs = Data[:,0]
Dms = Data[:,1:]
```

In [20]:
```
Dm = Dms[784,:].reshape(28,28)
```

In [21]:
```
Dm
```

Out[21]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,  24,
        108, 180, 253,  76,  19,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  45, 252,
        252, 235, 206, 207, 117,   0,   0,  43,  22,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 170, 252,
        170,  44,   0,   0,   0,   0,  68, 246, 199,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 159, 252,
        111,   0,   0,   0,   0,  26, 203, 252, 188,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  34, 252,
        246, 146,   0,   0,  74, 205, 252, 252,  63,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  95,
        247, 253, 201,  34, 212, 253, 234,  21,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         67, 202, 252, 253, 244, 123,  17,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  93, 252, 253, 206,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        100, 224, 252, 253, 206,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  22,
        215, 252,  95, 253, 248,  63,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  13, 212,
        247,  94,   0, 255, 253, 131,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 118, 252,
        110,   0,   0, 253, 252, 183,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  85, 253, 240,
         50,   0,   0, 253, 252, 183,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 157, 245,  79,
          0,   0,   0, 253, 252,  89,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  85, 250, 230,   0,
          0,   0, 106, 253, 210,   6,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  51, 243, 244,  50,
          0,   9, 233, 255, 144,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
```

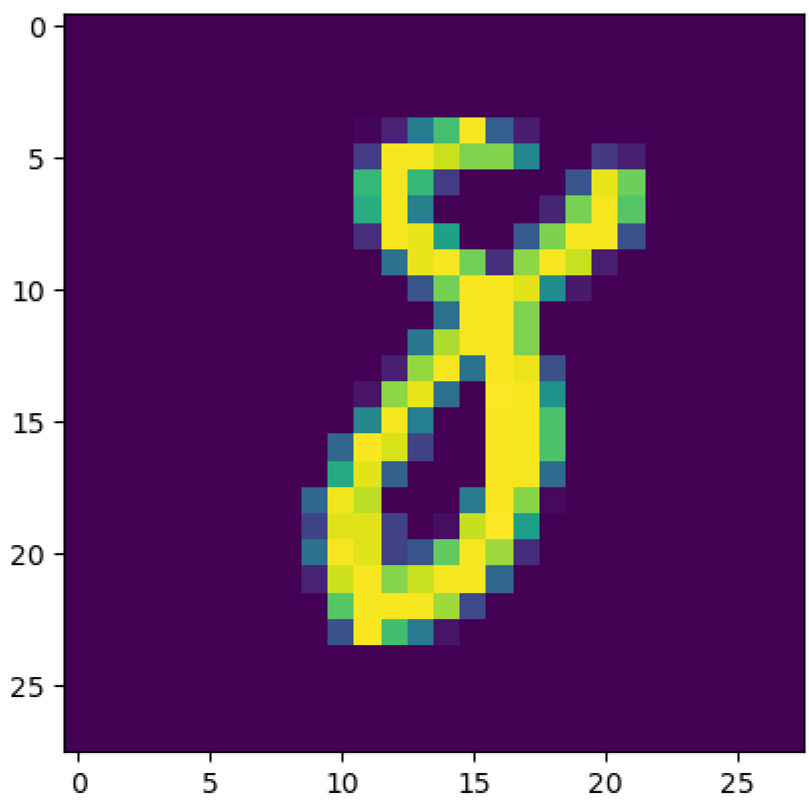
```
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  93, 252, 243,  50,
         66, 194, 252, 218,  33,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  25, 236, 253, 209,
        234, 252, 252,  84,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 188, 253, 252,
        252, 218,  56,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  63, 253, 178,
        106,  14,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]])
```

In [22]:

```
plt.imshow(Dm)
```

Out[22]:

```
<matplotlib.image.AxesImage at 0x2e14f1dd0>
```



After we know about KNN Algorithm and MINST databse, we have to back our project.

First of all, we can apply KNN algorithm to compute distances from the given point to all training data points. Then we can select the k-value nearest neighbors based on these

distances. After this, we can through the distances to determine the most coomon label among the neighbors.

Hence, we have to untilze Euclidean distance formula to calculate the distances from the points. The formula for Euclidean Distance formula: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

In [81]:
```
def euclideanDistance(x1,x2):
    return np.sqrt(np.sum((x1-x2)** 2,axis=1))
```

When we try to run the KNN algorithm first, we have to load the MINST dataset to make sure we have enough training set and testing set. Hence, we should import the MINST dataset first to let Python know we will use this dataset.

In [82]:
```
# We try to load the training set MINST dataset
import csv
import statistics as st
```

At first, we use Eulidean Distance to compute the distance from the point K. After we can compare the distances to determine the most common lbel among the neighbors. Then, return the predicted label and the index of the k nearest neighbors.

In [37]:
```
# Here are the parameters and use KNN algorithm
def KNN(training_data, training_labels, point, k):
    # Then we have to find the nearest distance from the point k,

    Dist = euclideanDistance(point, training_data)

    nearest_coords = np.argsort(Dist)[:k]
    nearest_labels = training_labels[nearest_coords]
    # The distance untill the kth closest point


    Common_labels = st.mode(nearest_labels)
    return Common_labels
```

Here is testing part, we can pick 50,000 training data and 1000 testing data.

In [39]:
```
Data.shape
```

Out[39]:
```
(60000, 785)
```

In [73]:
```
# This is show the coordinate of Data
Coords = Data[:,1:]
Labs = Data[:,0]
```

Testing Set

When we are done with the coding part, we can find the predicted labels and the index of the k nearest neighbors. We have to use the test set from the MINST dataset, so we can try to using 50,000 training sets and 10,00 test sets. Therefore, we can create a function called

TestKNN that has five arguments: train_data,train_labs,test_data,test_labs,k. “train_data” and ‘train_labs’ correspond to the coordinates (x,y) of the training dataset, ‘test_data’ and “test_labs” correspond to the coordinates (x,y) of the test dataset. Coordinates of the test dataset (x,y).

Confusion Matirx

A confusion matrix helps asses classification model performance. It can compare predicted values against actual values for a dataset. It also a visulaization method for classifier algroithm results. Thus, we will use confusion matrix inside the testing function.

In [55]:
```
def TestKNN(train_data,train_labs,test_data,test_labs,k):
    testrows,testcols = test_data.shape
    # we will create a confusion matrix
    conf = np.zeros((10,10))
    # we have to create a loop to run the each testing dataset
    for row in range(testrows):
        pt = test_data[row,:]
        pred_lab = KNN(train_data, train_labs, pt, k)
        true_lab = test_labs[row]
        conf[true_lab,pred_lab]+=1
    return conf
```

We will choose the 100 testing data set and 59900 training data set, also we will pick k = 5. The confusion matrix shows here are some "1" and "2" coming out, that means the accuracy is changing. We can try change the number of traing data set.

In [61]:
```
c = TestKNN(Dms[:100,:],Labs[:100],Dms[59900:,:],Labs[59900:],5)
```

In [62]:
```
c
```

Out[62]:
```
array([[ 6.,  0.,  0.,  0.,  1.,  2.,  0.,  1.,  0.,  0.],
       [ 0.,  9.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  6.,  0.,  1.,  2.,  0.,  0.,  0.,  0.],
       [ 0.,  2.,  0.,  8.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  1.,  0.,  0.,  7.,  1.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  1.,  7.,  0.,  0.,  0.,  1.],
       [ 0.,  1.,  0.,  0.,  0.,  1.,  8.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  8.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 10.,  2.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  9.]])
```

In [68]:
```
D = TestKNN(Dms[:1000,:],Labs[:1000],Dms[59900:,:],Labs[59900:],5)
```

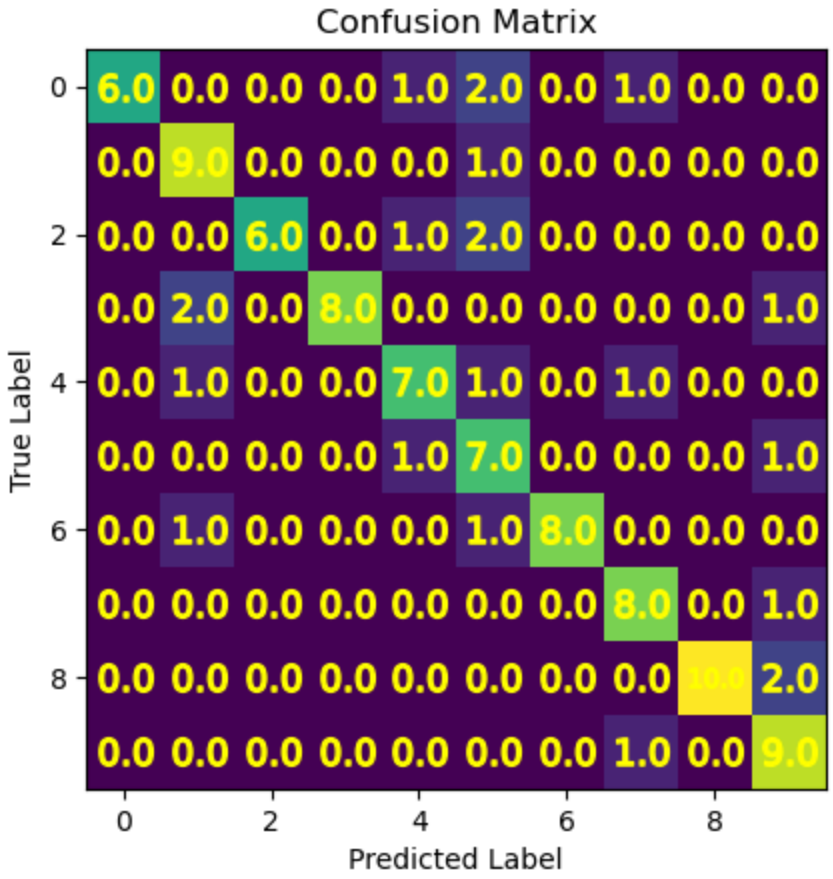
In [69]:
```
D
```

Out[69]:
```
array([[ 9.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.],
       [ 0., 10.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  9.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0., 10.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  1.,  0.,  0.,  8.,  0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  9.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0., 10.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  9.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 11.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  9.]])
```

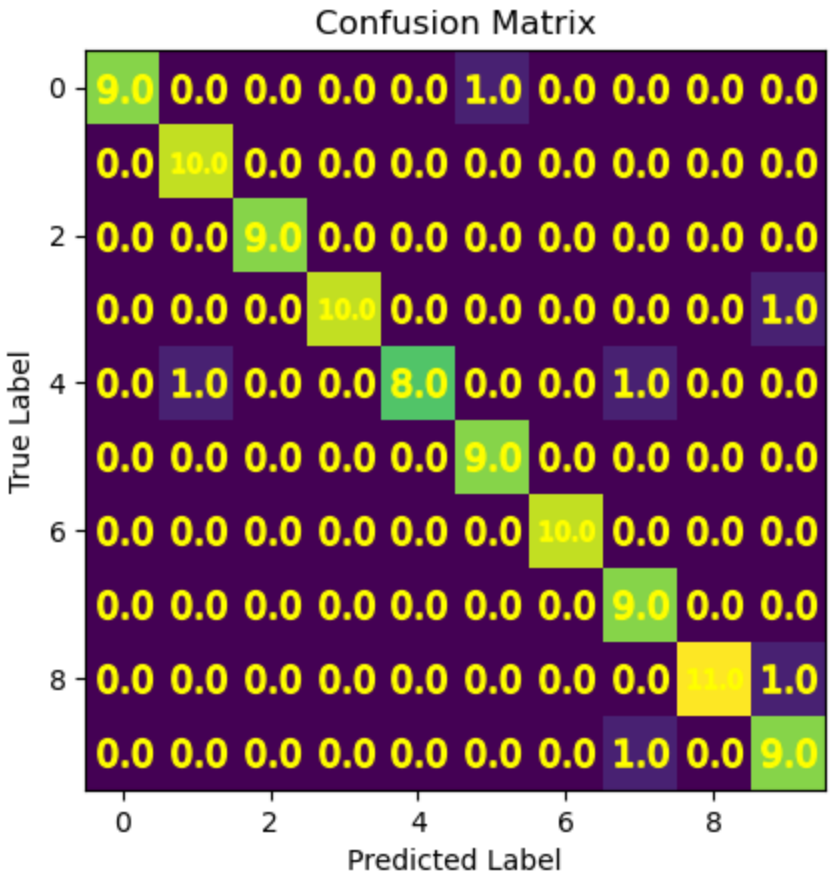
This function "ShowConf" is polt for confusion matrix, and True label.

In [65]:
```
def ShowConf(Arr):
    plt.imshow(Arr)
    nRow,nCol = Arr.shape
    # we will loop each row and each column to check the predicted label and Tr
    for row in range(nRow):
        for col in range(nCol):
            plt.plot(col,row,marker = r'$'+str(Arr[row,col])+'$',markersize = 7
    # When we polt the confusion matrix, we can put the title for x,y.
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    plt.show()
```

In [66]:
```
ShowConf(c)
```



In [70]:
```
ShowConf(D)
```



Here we can through the predicted label and True label to analysis the accuracy of the results is changed.

Follow the confusion matrix D and confusion matrix c, we will discover the "1" will change less. That's means the accuracy of the results is improved.

Furthermore, we also can use the percentage to indicate the difference bewteen two matrices.

In [41]:
```
def PercentRight(conf):
    return conf.trace()/np.sum(conf)
```

In [100…
```
PercentRight(c)
```

Out[100]:
```
0.78
```

In [74]:
```
PercentRight(D)
```

Out[74]:
```
0.94
```

In conclusion, through the percentage function, if the number is more larger, then the percentage will more higher. Hence, we can know the accuracy of the results is depends on the training data set and testing data set,k. When we pick more testing data, the accuracy will be imporved.

In [ ]: