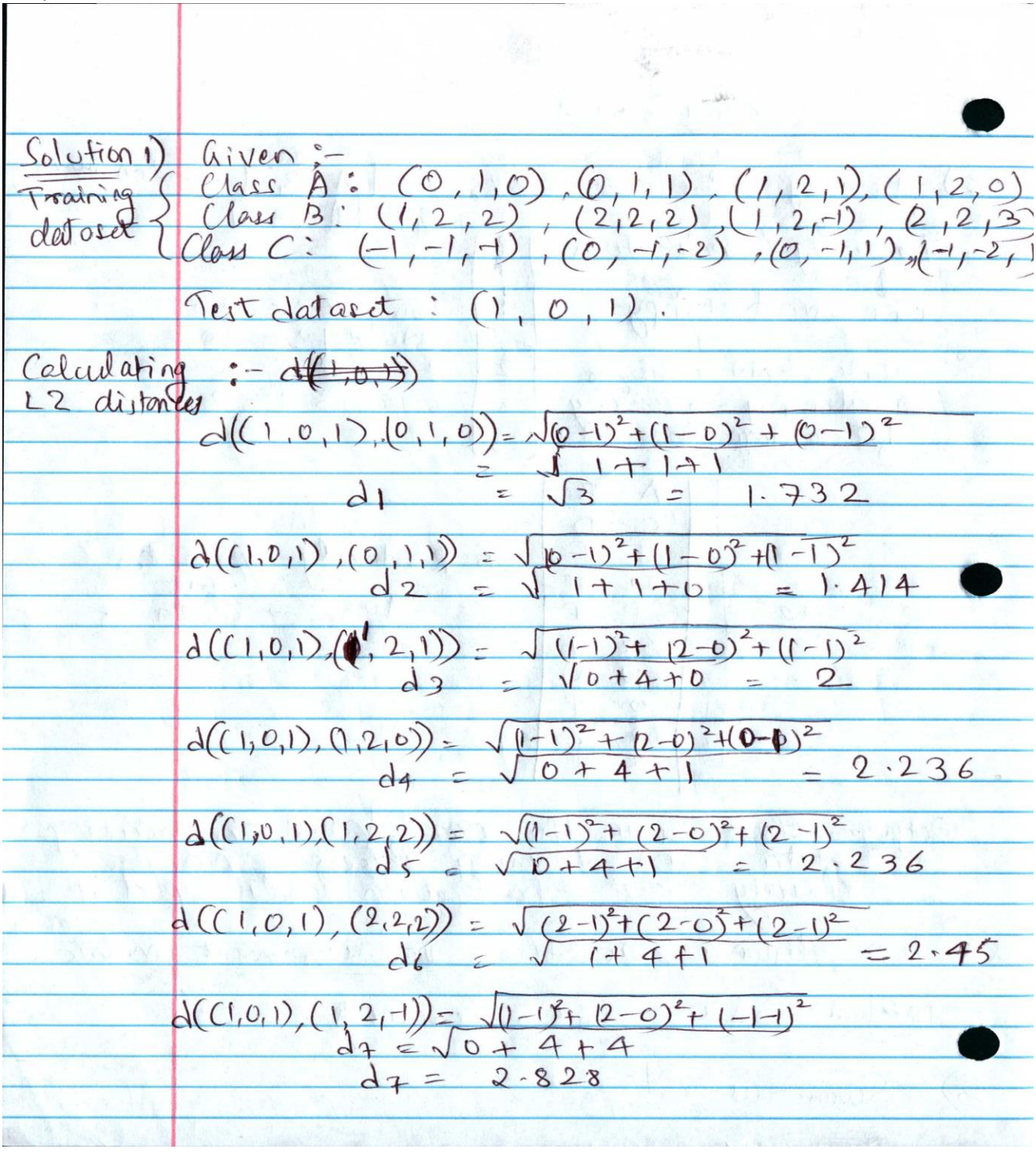
Homework 1: Intro to Deep Learning (Spring 2020)

Name: Pratik Mistry
Net ID: pdm79
RUID: 194008675

Solution 1: Please find the attached images as part of the solution. It consists of calculations of the computation of KNN in handwritten format.



Solution 1) Given :-

Training dataset {
Class A : (0, 1, 0) , (0, 1, 1) , (1, 2, 1), (1, 2, 0)
Class B : (1, 2, 2) , (2, 2, 2) , (1, 2, -1) , (2, 2, 3)
Class C : (-1, -1, -1) , (0, -1, -2) , (0, -1, 1) , (-1, -2, 1)

Test dataset : (1, 0, 1).

Calculating L2 distances :-

$d((1,0,1),(0,1,0)) = \sqrt{(0-1)^2+(1-0)^2+(0-1)^2}$
$= \sqrt{1+1+1}$
$d_1 = \sqrt{3} = 1.732$

$d((1,0,1),(0,1,1)) = \sqrt{(0-1)^2+(1-0)^2+(1-1)^2}$
$d_2 = \sqrt{1+1+0} = 1.414$

$d((1,0,1),(1,2,1)) = \sqrt{(1-1)^2+(2-0)^2+(1-1)^2}$
$d_3 = \sqrt{0+4+0} = 2$

$d((1,0,1),(1,2,0)) = \sqrt{(1-1)^2+(2-0)^2+(0-1)^2}$
$d_4 = \sqrt{0+4+1} = 2.236$

$d((1,0,1),(1,2,2)) = \sqrt{(1-1)^2+(2-0)^2+(2-1)^2}$
$d_5 = \sqrt{0+4+1} = 2.236$

$d((1,0,1),(2,2,2)) = \sqrt{(2-1)^2+(2-0)^2+(2-1)^2}$
$d_6 = \sqrt{1+4+1} = 2.45$

$d((1,0,1),(1,2,-1)) = \sqrt{(1-1)^2+(2-0)^2+(-1-1)^2}$
$d_7 = \sqrt{0+4+4}$
$d_7 = 2.828$

$$d((1,0,1),(2,2,3)) = \sqrt{(2-1)^2+(2-0)^2+(3-1)^2}$$
$$d_8 = \sqrt{1+4+4} = 3$$

$$d((1,0,1),(-1,-1,-1)) = \sqrt{(-1-1)^2+(-1-0)^2+(-1-1)^2}$$
$$d_9 = \sqrt{4+1+4} = 3$$

$$d((1,0,1),(0,-1,-2)) = \sqrt{(0-1)^2+(-1-0)^2+(-2-1)^2}$$
$$= \sqrt{1+1+9}$$
$$d_{10} = 3.317$$

$$d((1,0,1),(0,-1,0)) = \sqrt{(0-1)^2+(-1-0)^2+(1-1)^2}$$
$$d_{11} = \sqrt{1+1+0} = 1.414$$

$$d((1,0,1),(-1,-2,1)) = \sqrt{(-1-1)^2+(-2-0)^2+(1-1)^2}$$
$$= \sqrt{4+4}$$
$$d_{12} = 2.828$$

∴ L2 Distance between test data and all the training data are:

~~[1.732, 1.414, 2.236, 2.236, 2.45, 2.828,~~

Class A Distances | Class B distances

[1.732, 1.414, 2, 2.236], [2.236, 2.45,

2.828, 3], [3, 3.317, 1.414, 2.828]]

Class B | Class C distances.

∴ Now, 1st four distances as shown are L2 distances between test data and class A pairs, next four are distances between test data and class B pairs and last four with that of class C pairs,

∴ To find the 'K' nearest neighbours and label of respective test data:-

1) When K=1, the minimum distance is $d_2 = 1.414$ (chosen randomly as it is same as $d_{11} = 1.414$). This distance $d_2 = 1.414$ is distance of test data with respect to data of class A. Thus, label of test data when K=1 is 'A'.

2) When K=2, the two nearest neighbours i.e two minimum distances are:
$d_2 = 1.414$ ... distance from class A pair
$d_{11} = 1.414$ ... distance from class C pair.
Thus, since there is tie in the distances between two class ~~pair~~ labels – A and C. We can choose randomly, here I choose label of test data as 'A'.

3) When K=3, the three minimum distances are:
$d_2 = 1.414$ ... distance from class A pair
$d_{11} = 1.414$ ... distance from class C pair
$d_1 = 1.732$ ... distance from class A pair

Since, it is more closer to class A data set as count/majority of votes seen above, than class C pair/dataset. The label for the test data when K=3 is A, because of majority votes.

Thus, when K=1, ... label of test data is 'A', when K=2, label is A and when K=3, label is A.

Conclusion:

- When **k=1** the nearest neighbor can be the pair of Class A and Class C as distance is same i.e. 1.414 and to randomly select the minimum neighbor, I have selected distance with respect to Class A and hence the label of test data is **Class A**.
- When **k=2** the nearest neighbor is the pair of Class A and Class C and it is a tie when counting the majority votes. But, then for simplifying we consider the 1NN solution (randomly choosing the either of the class label) and according to that Class A pair is chosen and hence label of the test data would be **Class A**.
- When **k=3** the nearest neighbor are pairs from Class A, Class C and Class A. Thus, if we count the majority votes, test data is nearest to neighbor with Class A and hence the label would be **Class A.**

Solution 2: The program for the problem 2 is written below:
Code:

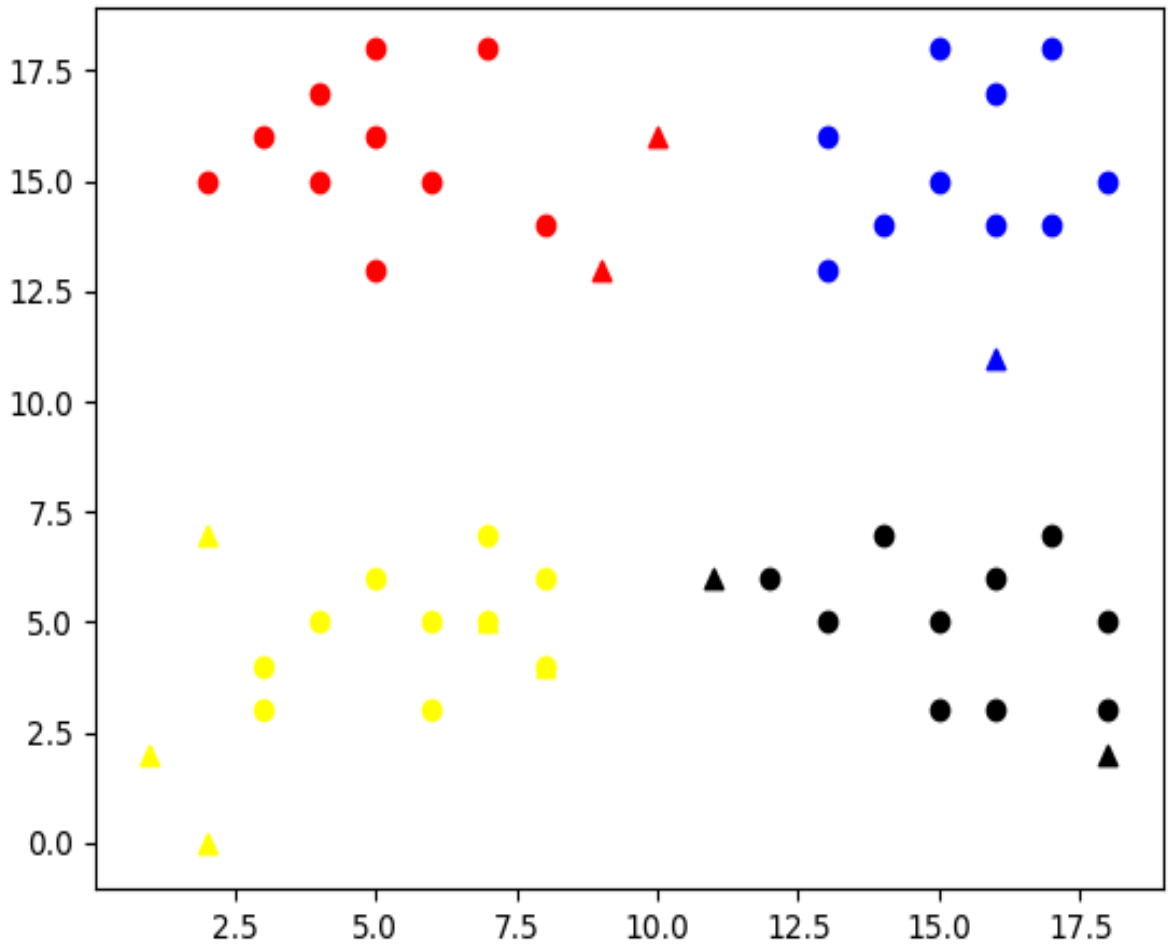
```
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt

# load mini training data and labels
mini_train = np.load('knn_minitrain.npy')
mini_train_label = np.load('knn_minitrain_label.npy')

# randomly generate test data
mini_test = np.random.randint(20, size=20)
mini_test = mini_test.reshape(10,2)



# Define knn classifier
def kNNClassify(newInput, dataSet, labels, k):
  result=[]
  ########################
  # Input your code here #
  ########################
  l2_distance=np.zeros((10,40))                    # Array for storing L2 distance for all test data w.r.t
training data
  for i in range(len(newInput)):                   # Finding L2 distance of each random test data
    for j in range(len(dataSet)):
      distance = np.linalg.norm(dataSet[j]-newInput[i])  # Calculating the L2 Distance
      l2_distance[i,j] = distance                # Storing the L2 distance for each test data
```

```
    for i in range(len(newInput)):                  # Finding the true label for each random test data
        label_count = np.zeros(4)                   # Creating array for counting the label values for KNN - 0,1,2,3
        knn_indices = np.argsort(l2_distance[i])[:k]        # Sorting in ascending order and finding indices of "K" nearest neighbours
        for j in range(len(knn_indices)):           # Finding label of each "K" nearest neighbour
            label = labels[knn_indices[j]]          # Getting the label of each "K" nearest neighbour and incrementing the count
            label_count[label]+=1
        result.append(np.argmax(label_count))       # Appending the label (index) having max count to the result list

    ######################
    # End of your code #
    ######################
    return result

outputlabels=kNNClassify(mini_test,mini_train,mini_train_label,10)


print ('random test points are:', mini_test)
print ('knn classfied labels for test:', outputlabels)

# plot train data and classfied test data
train_x = mini_train[:,0]
train_y = mini_train[:,1]
fig = plt.figure()
plt.scatter(train_x[np.where(mini_train_label==0)], train_y[np.where(mini_train_label==0)], color='red')
plt.scatter(train_x[np.where(mini_train_label==1)], train_y[np.where(mini_train_label==1)], color='blue')
plt.scatter(train_x[np.where(mini_train_label==2)], train_y[np.where(mini_train_label==2)], color='yellow')
plt.scatter(train_x[np.where(mini_train_label==3)], train_y[np.where(mini_train_label==3)], color='black')

test_x = mini_test[:,0]
test_y = mini_test[:,1]
outputlabels = np.array(outputlabels)
plt.scatter(test_x[np.where(outputlabels==0)], test_y[np.where(outputlabels==0)], marker='^', color='red')
plt.scatter(test_x[np.where(outputlabels==1)], test_y[np.where(outputlabels==1)], marker='^', color='blue')
```

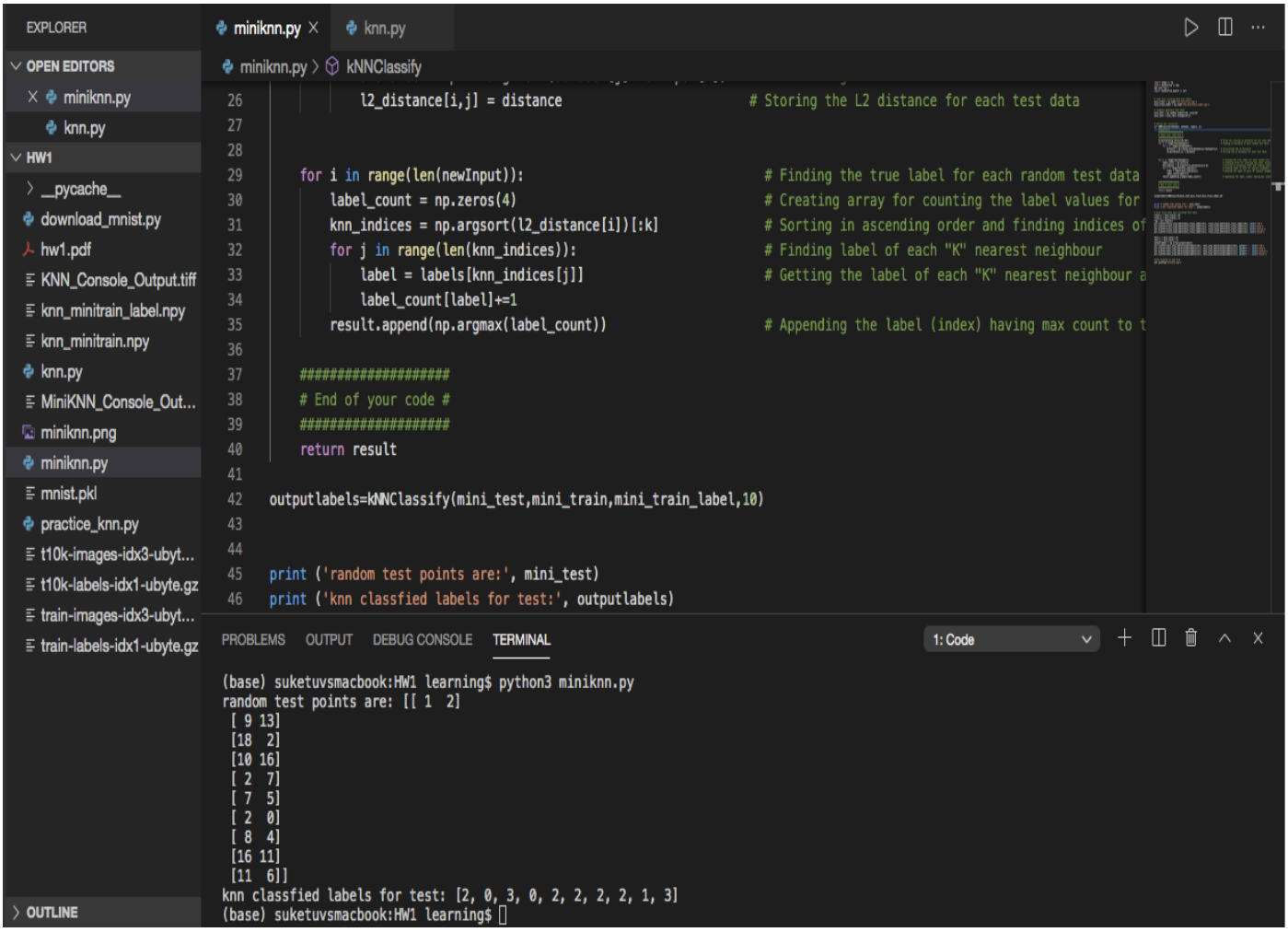
```
plt.scatter(test_x[np.where(outputlabels==2)], test_y[np.where(outputlabels==2)], marker='^',
color='yellow')
plt.scatter(test_x[np.where(outputlabels==3)], test_y[np.where(outputlabels==3)], marker='^',
color='black')

#save diagram as png file
plt.savefig("miniknn.png")
```

Output:

1. Plotted image



2. Console Output Image:



Conclusion:

- As seen in the output screenshots of the plotted image as well as console output, it can be verified that the solution of KNN Classifier using L2 Distance as the measurement metric to find K-nearest neighbors correctly classifies the test data into correct labels.
- In the program, value of k chosen is 10 i.e. we have considered choosing 10 nearest neighbors to find the correct label of each 10 random test data generated and K=10 gives us the best accuracy.
- There are 2 Red Triangles, 1 Blue Triangle, 5 Yellow Triangle and 2 Black Triangle as Label (Two out of 5 yellow triangles nearly overlaps the circle i.e. training data and hence can be barely visible. Thus, if attached image is zoomed can be viewed up to certain level).

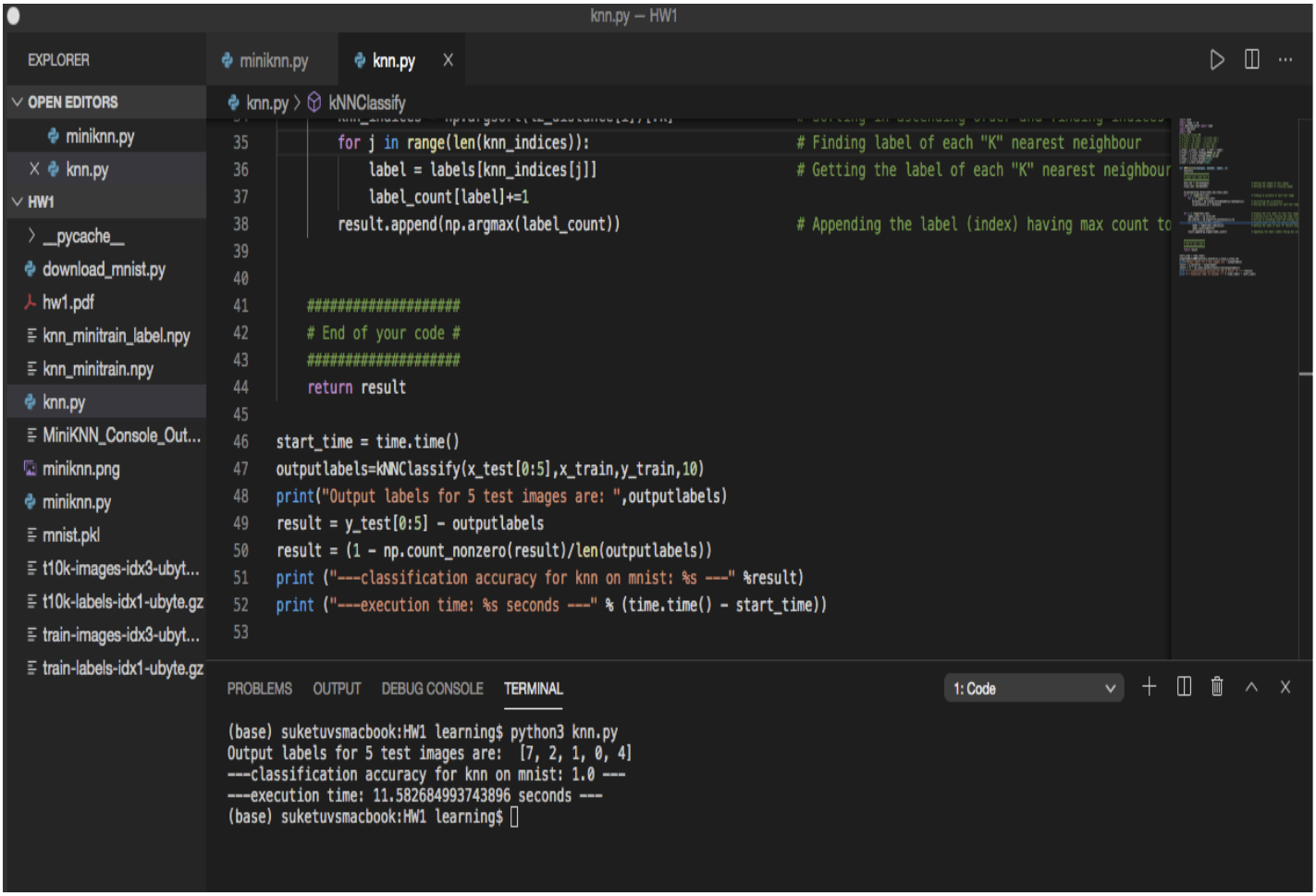
Solution 3: The program for the problem 2 is written below:
Code:

```
import math
import numpy as np
from download_mnist import load
import operator
import time
# classify using kNN
# x_train = np.load('../x_train.npy')
# y_train = np.load('../y_train.npy')
# x_test = np.load('../x_test.npy')
# y_test = np.load('../y_test.npy')
x_train, y_train, x_test, y_test = load()
x_train = x_train.reshape(60000,28,28)
x_test  = x_test.reshape(10000,28,28)
x_train = x_train.astype(float)
x_test = x_test.astype(float)

def kNNClassify(newInput, dataSet, labels, k):
   result=[]
   ########################
   # Input your code here #
   ########################
   test_len = len(newInput)                          # Getting the length of test images
   train_len= len(dataSet)                           # Getting the length of training images

   l2_distance=np.zeros((test_len,train_len))
   for i in range(test_len):                         # Finding L2 distance of each test image
      for j in range(train_len):
         distance = np.linalg.norm(dataSet[j]-newInput[i])      # Calculating the L2 Distance
         l2_distance[i,j] = distance                  # Storing the L2 distance for each test image


   for i in range(test_len):                         # Finding the true label for each test image
      label_count = np.zeros(10)                     # Creating array for counting the label values for numbers
- 0-9
      knn_indices = np.argsort(l2_distance[i])[:k]             # Sorting in ascending order and finding indices
of "K" nearest neighbours
      for j in range(len(knn_indices)):                # Finding label of each "K" nearest neighbour
         label = labels[knn_indices[j]]                # Getting the label of each "K" nearest neighbour and
incrementing the count
         label_count[label]+=1
```

result.append(np.argmax(label_count)) # Appending the label (index) having max count to the result list

```
    ######################
    # End of your code #
    ######################
    return result

start_time = time.time()
outputlabels=kNNClassify(x_test[0:5],x_train,y_train,10)
print("Output labels for 5 test images are: ",outputlabels)
result = y_test[0:5] - outputlabels
result = (1 - np.count_nonzero(result)/len(outputlabels))
print ("---classification accuracy for knn on mnist: %s ---" %result)
print ("---execution time: %s seconds ---" % (time.time() - start_time))
```

Output: Below is the console output image



Conclusion:

- As seen in the output screenshot of the console output, it can be verified that the solution of KNN Classifier using L2 Distance as the measurement metric to find K-nearest neighbors correctly classifies the test images into correct labels. ***The accuracy of the model as seen in image is 1 i.e. 100%.***
- In the program, we have chosen 5 test images since taking images more than 5 takes more time to execute the program. For 5 images, it takes around 11.5 seconds as seen in image while for 10 images it takes around 26 seconds for complete execution.
- In the program, value of k chosen is 10 i.e. we have considered choosing 10 nearest neighbors to find the correct label of each test image generated and K=10 gives us the best accuracy.