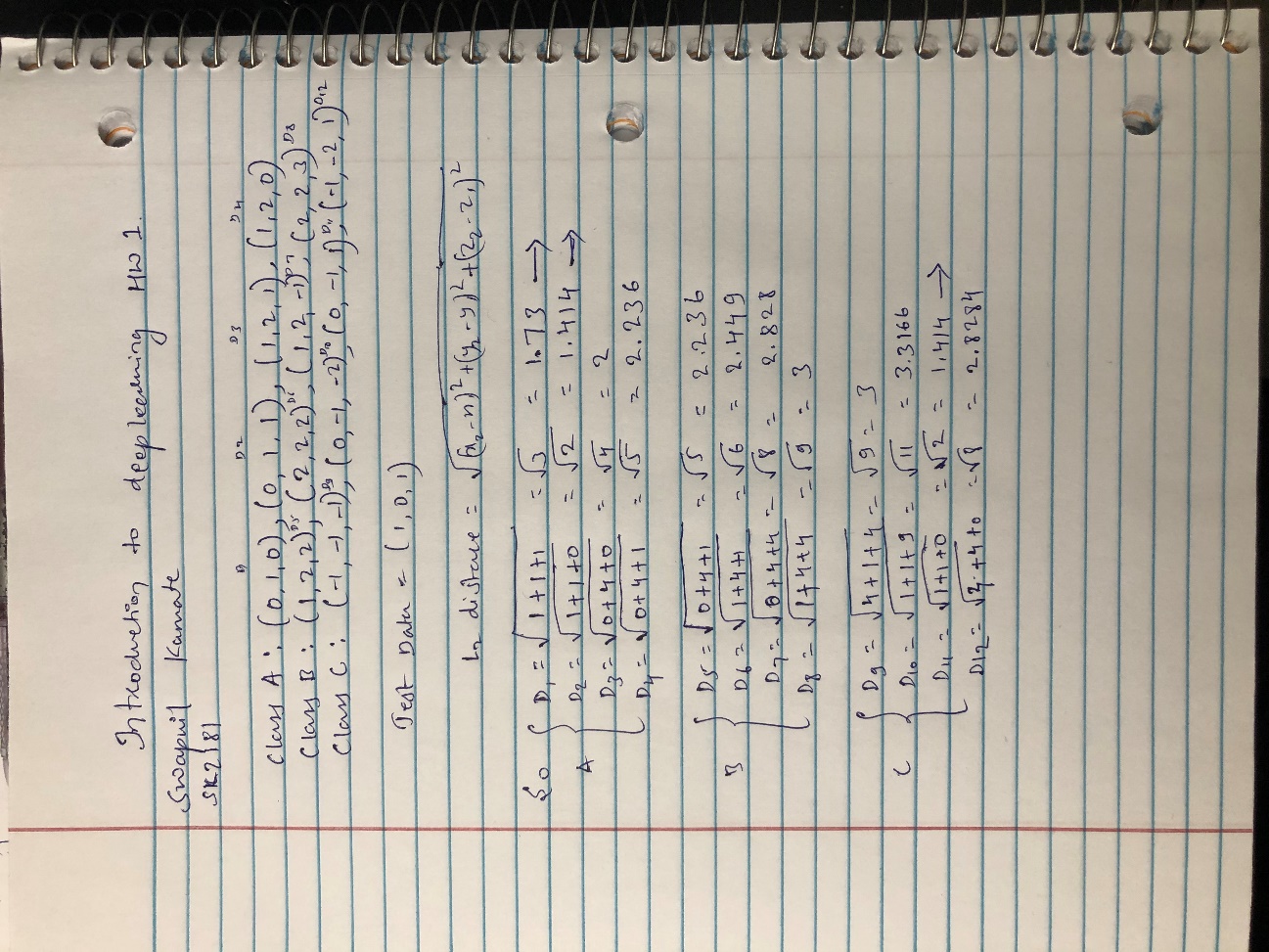
**Homework 1 Introduction to Deep Learning.**

**Swapnil Kamate**

**Sk2181**

**Solution 1:**



Result:

When K=1 We need to consider only one lowest distance. So, we could see the lowest distance(L2) is 1.414 which is both d2 and d11. So, when value of K =1 then our test data can fall into either Class A or Class C.

When K = 2 Again, in this case we need to consider two lowest distances which are again surprisingly 1.414 and 1.414 which are d2 and d11. So, when K = 2 our test data can fall into either Class A or Class C.

When K = 3, we need to consider 3 lowest distances and according to our results the three lowest distances are d2 = 1.414 -> Class A, d11 = 1.414 -> Class C and d1 = 1.73 -> Class A. We could see two of our distances fall in the Class A so, when K =3 I would say our test data can be classified as Class A result.

**Solution 2:**

**Source 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=[]

# compute L2 distance for all test and train samples

distances = []

for item1 in newInput:

d = []

for item2 in dataSet:

distance = np.sqrt(np.sum((item1-item2)\*\*2)) #calclulating L2 distance

d.append(distance)

distances.append(d)

print(distances)

# decide which class the test samples belong in

for i in range(len(newInput)):

label\_value = np.zeros(4) # creating an array for lable values

knn\_indices = np.argsort(distances[i])[:k]

for j in range(len(knn\_indices)):

label = labels[knn\_indices[j]] #incrementing the count after getting each 'K'

label\_value[label]+=1

result.append(np.argmax(label\_value))

return result

outputlabels=kNNClassify(mini\_test,mini\_train,mini\_train\_label,4)

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')

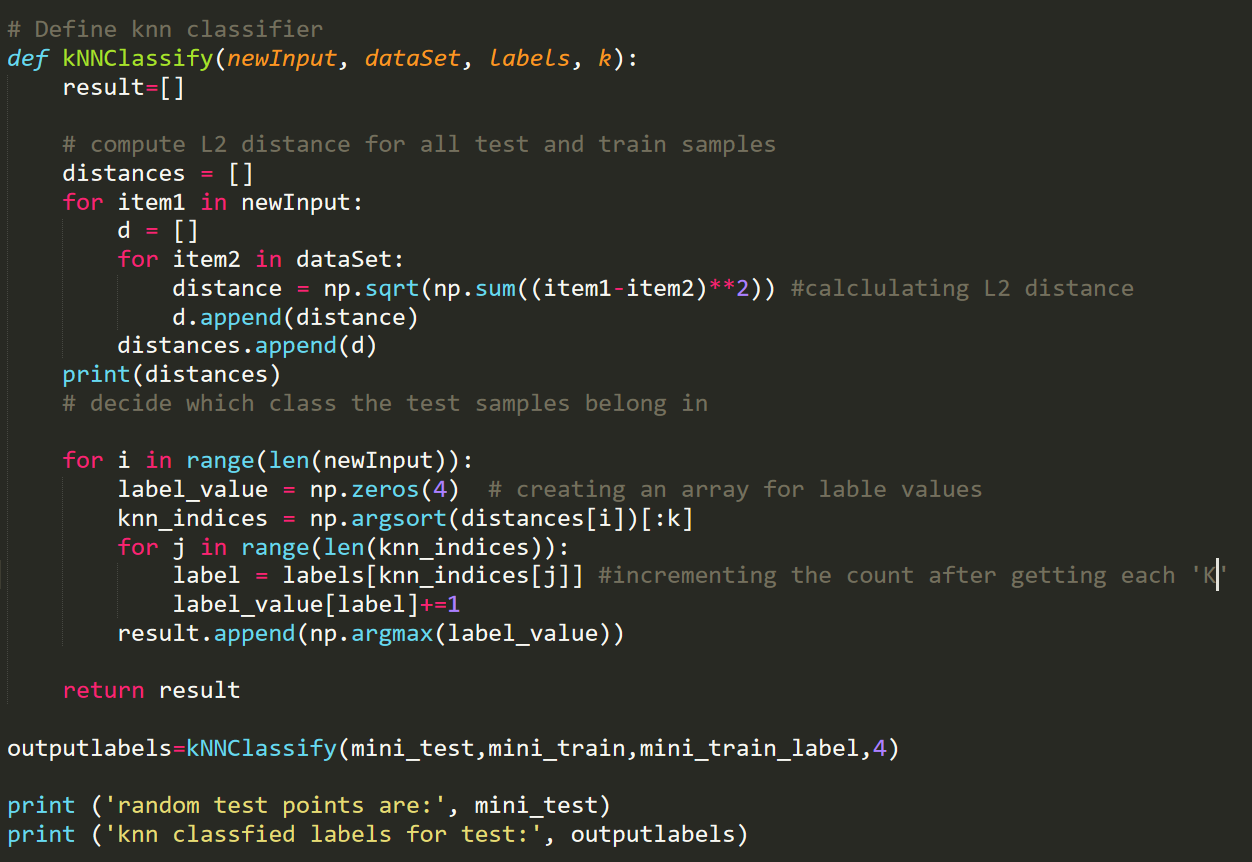
plt.scatter(test\_x[np.where(outputlabels==2)], test\_y[np.where(outputlabels==2)], marker='^', color='yellow')

plt.scatter(test\_x[np.where(outputlabels==3)], test\_y[np.where(outputlabels==3)], marker='^', color='black')

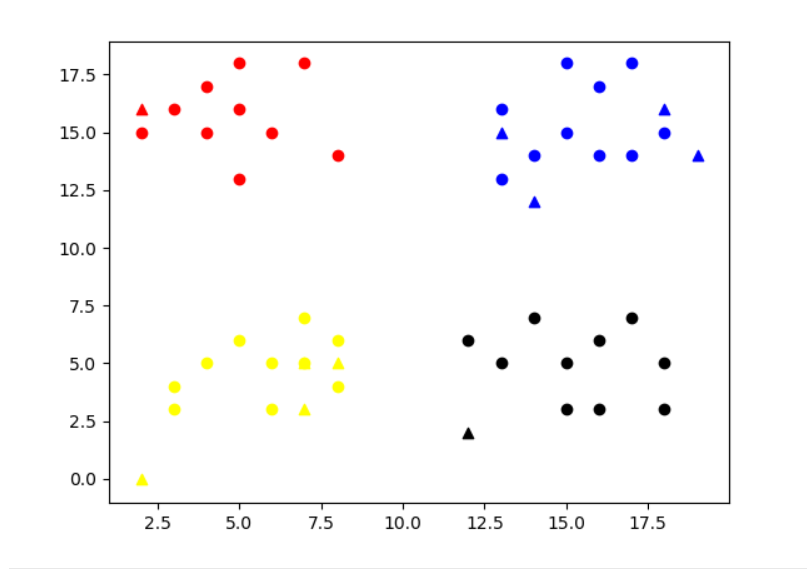
#save diagram as png file

plt.savefig("miniknn.png")

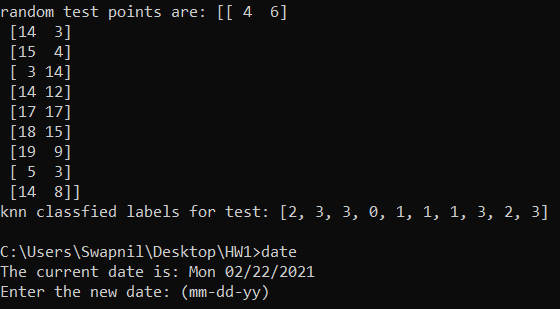
**Screenshot of code:**



**Output of plotted Image**



**CMD Output:**



**Solution 3:**

**Source 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=[]

# compute L2 distance for all test and train samples

distances = np.zeros((len(newInput), len(dataSet)))

for i in range(len(newInput)):

for j in range(len(dataSet)):

distance = np.sqrt(np.sum((newInput[i]-dataSet[j])\*\*2))

distances[i, j] = distance

# decide which class the test samples belong in

for i in range(len(newInput)):

label\_value = np.zeros(10)

knn\_indices = np.argsort(distances[i])[:k]

for j in range(len(knn\_indices)):

label = labels[knn\_indices[j]]

label\_value[label]+=1

result.append(np.argmax(label\_value))

return result

start\_time = time.time()

outputlabels=kNNClassify(x\_test[0:20],x\_train,y\_train,10)

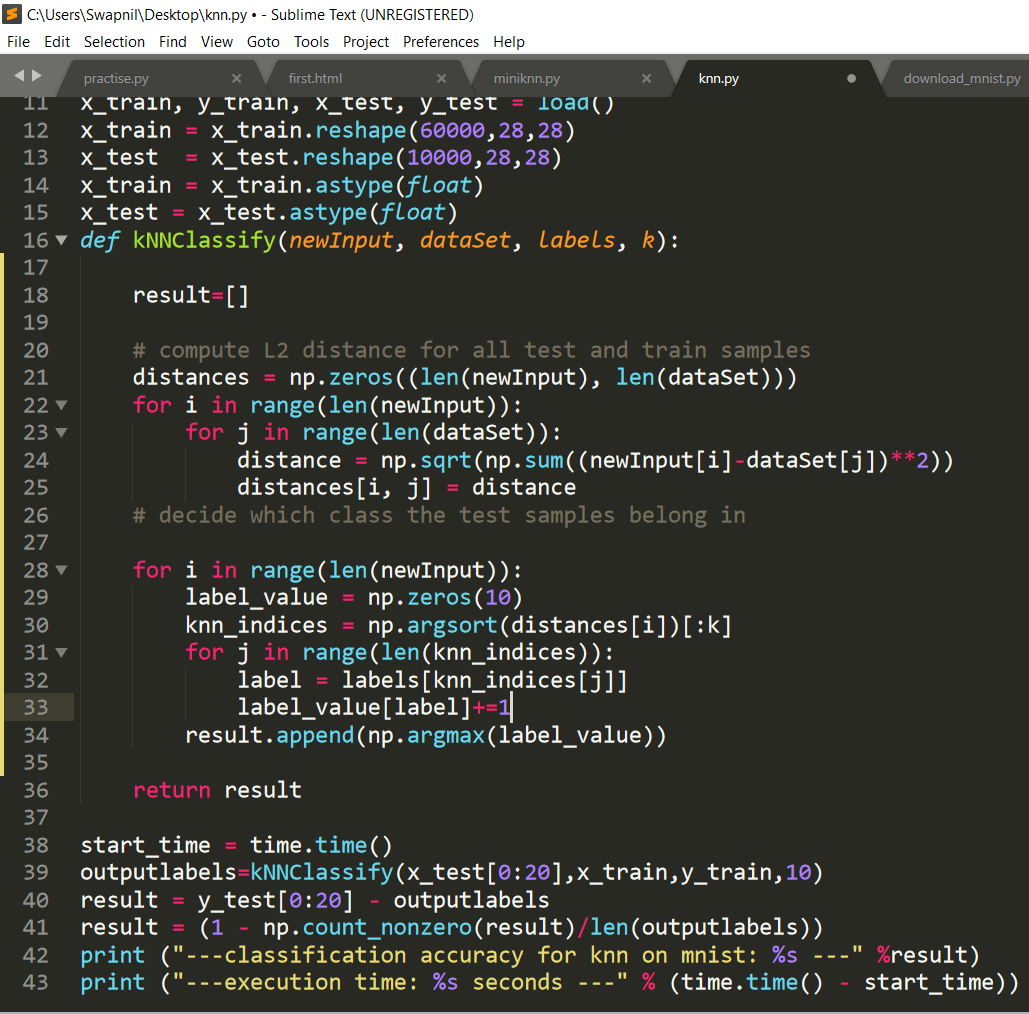
result = y\_test[0:20] - outputlabels

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))

**Screenshot of Code:**



**Output:**

