import numpy as np  
import matplotlib.pyplot as plt  
import time  
*#Load the training set:*train\_data = np.load('C:/Users/POOJA SARKAR/Downloads/data/train\_data.npy')  
train\_labels = np.load('C:/Users/POOJA SARKAR/Downloads/data/train\_labels.npy')  
*#Load the testing set:*test\_data = np.load('C:/Users/POOJA SARKAR/Downloads/data/test\_data.npy')  
test\_labels = np.load('C:/Users/POOJA SARKAR/Downloads/data/test\_labels.npy')  
*#Print out data dimensions:*print("Training dataset dimensions: ", np.shape(train\_data))  
print("Number of training labels: ", len(train\_labels))  
print("Testing dataset dimensions: ", np.shape(test\_data))  
print("Number of testing labels: ", len(test\_labels))  
*#Compute the number of examples of each digit:*train\_digits, train\_counts = np.unique(train\_labels, return\_counts=True)  
print("Training set distribution:")  
print(dict(zip(train\_digits, train\_counts)))  
test\_digits, test\_counts = np.unique(test\_labels, return\_counts=True)  
print("Test set distribution:")  
print(dict(zip(test\_digits, test\_counts)))  
*#2. Visualizing the data  
#Define a function that displays a digit given its vector representation:*def show\_digit(x):  
 plt.axis('off')  
 plt.imshow(x.reshape((28,28)), cmap=plt.cm.gray)  
 plt.show()  
 return  
*#Define a function that takes an index into a particular data set ("train" or "test")  
# and displays that image.*def vis\_image(index, dataset="train"):  
 if dataset == "train":  
 show\_digit(train\_data[index,])  
 label = train\_labels[index]  
 else:  
 show\_digit(test\_data[index,])  
 label = test\_labels[index]  
  
 print("Label " + str(label))  
 return  
vis\_image(0, "train")  
vis\_image(0, "test")  
*#Function that computes squared Euclidean distance between two vectors.*def squared\_dist(x,y):  
 return np.sum(np.square(x-y))  
*#Compute distance between a seven and a one in our training set.*vis\_image(4, "train")  
vis\_image(5, "train")  
print("Distance from 7 to 1: ", squared\_dist(train\_data[4,],train\_data[5,]))  
*#Compute distance between a seven and a two in our training set.*vis\_image(4, "train")  
vis\_image(1, "train")  
print("Distance from 7 to 2: ", squared\_dist(train\_data[4,],train\_data[1,]))  
*#Compute distance between two seven's in our training set.*vis\_image(4, "train")  
vis\_image(7, "train")  
print("Distance from 7 to 7: ", squared\_dist(train\_data[4,],train\_data[7,]))  
*#4. Computing nearest neighbors*def find\_NN(x):  
 distances = [squared\_dist(x, train\_data[i,]) for i in range(len(train\_labels))]  
 return np.argmin(distances)  
def NN\_classifier(x):  
 index = find\_NN(x)  
 return train\_labels[index]  
*#A success case:*print("A success case:")  
print("NN classification: ", NN\_classifier(test\_data[0,]))  
print("True label: ", test\_labels[0])  
print("The test image:")  
vis\_image(0, "test")  
print("The corresponding nearest neighbor image:")  
vis\_image(find\_NN(test\_data[0,]), "train")  
*#A failure case:*print("A failure case:")  
print("NN classification: ", NN\_classifier(test\_data[39,]))  
print("True label: ", test\_labels[39])  
print("The test image:")  
vis\_image(39, "test")  
print("The corresponding nearest neighbor image:")  
vis\_image(find\_NN(test\_data[39,]), "train")  
  
t\_before = time.time()  
test\_predictions = [NN\_classifier(test\_data[i]) for i in range(len(test\_labels))]  
t\_after: float = time.time() *#Compute the error:*err\_positions = np.not\_equal(test\_predictions, test\_labels)  
error = float(np.sum(err\_positions))/len(test\_labels)  
print("Error of nearest neighbor classifier: ", error)  
print("Classification time (seconds): ", t\_after - t\_before)  
*#6. Faster nearest neighbor methods*from sklearn.neighbors import BallTree  
*# Build nearest neighbor structure on training data*t\_before = time.time()  
ball\_tree = BallTree(train\_data)  
t\_after = time.time()  
*# Compute training time*t\_training = t\_after - t\_before  
print("Time to build data structure (seconds): ", t\_training)  
*# Get nearest neighbor predictions on testing data*t\_before = time.time()  
test\_neighbors = np.squeeze(ball\_tree.query(test\_data, k=1, return\_distance=False))  
ball\_tree\_predictions = train\_labels[test\_neighbors]  
t\_after = time.time()  
*# Compute testing time*t\_testing = t\_after - t\_before  
print("Time to classify test set (seconds): ", t\_testing)  
*# Verify that the predictions are the same*print("Ball tree produces same predictions as above? ", np.array\_equal(test\_predictions, ball\_tree\_predictions))  
from sklearn.neighbors import KDTree  
*# Build nearest neighbor structure on training data*t\_before = time.time()  
kd\_tree = KDTree(train\_data)  
t\_after = time.time()  
*# Compute training time*t\_before = time.time()  
  
*# Build KD tree using training data*kd\_tree = KDTree(train\_data)  
  
*# Measure time after building KD tree*t\_after = time.time()  
  
*# Compute training time*t\_training = t\_after - t\_before  
print("Time to build data structure (seconds): ", t\_training)  
  
*# Measure time before querying KD tree for nearest neighbors*t\_before = time.time()  
  
*# Get nearest neighbor predictions on testing data*test\_neighbors = np.squeeze(kd\_tree.query(test\_data, k=1, return\_distance=False))  
kd\_tree\_predictions = train\_labels[test\_neighbors]  
  
*# Measure time after querying KD tree*t\_after = time.time()  
  
*# Compute testing time*t\_testing = t\_after - t\_before  
print("Time to classify test set (seconds): ", t\_testing)  
  
*# Verify that the predictions are the same as previously obtained predictions  
# Assuming test\_predictions is already defined*print("KD tree produces same predictions as above? ", np.array\_equal(test\_predictions, kd\_tree\_predictions))