Name: Pranav Shivkumar

RUID: 194007405

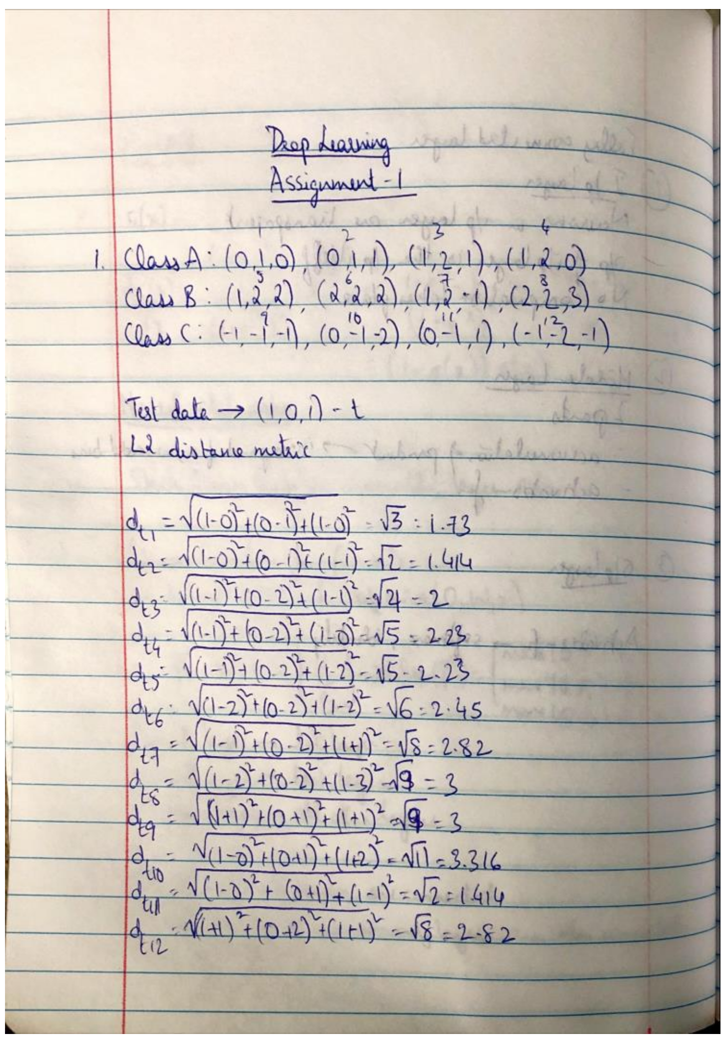
NetID: ps1029

Intro to Deep Learning

Assignment 1

1. Computation of KNN

   Based on the given data, the computed L2 distances between the test data and the labelled samples are shown below:



Deep Learning
Assignment-1

1. 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 data → (1,0,1) - t
L2 distance metric

$d_{t1} = \sqrt{(1-0)^2+(0-1)^2+(1-0)^2} = \sqrt{3} = 1.73$

$d_{t2} = \sqrt{(1-0)^2+(0-1)^2+(1-1)^2} = \sqrt{2} = 1.414$

$d_{t3} = \sqrt{(1-1)^2+(0-2)^2+(1-1)^2} = \sqrt{4} = 2$

$d_{t4} = \sqrt{(1-1)^2+(0-2)^2+(1-0)^2} = \sqrt{5} = 2.23$

$d_{t5} = \sqrt{(1-1)^2+(0-2)^2+(1-2)^2} = \sqrt{5} = 2.23$

$d_{t6} = \sqrt{(1-2)^2+(0-2)^2+(1-2)^2} = \sqrt{6} = 2.45$

$d_{t7} = \sqrt{(1-1)^2+(0-2)^2+(1+1)^2} = \sqrt{8} = 2.82$

$d_{t8} = \sqrt{(1-2)^2+(0-2)^2+(1-3)^2} = \sqrt{9} = 3$

$d_{t9} = \sqrt{(1+1)^2+(0+1)^2+(1+1)^2} = \sqrt{9} = 3$

$d_{t10} = \sqrt{(1-0)^2+(0+1)^2+(1+2)^2} = \sqrt{11} = 3.316$

$d_{t11} = \sqrt{(1-0)^2+(0+1)^2+(1-1)^2} = \sqrt{2} = 1.414$

$d_{t12} = \sqrt{(1+1)^2+(0+2)^2+(1+1)^2} = \sqrt{8} = 2.82$

a. K = 1

For K = 1, the test data is classified based on the 1 labeled sample to which it is closest. From the distances, the minimum distance is d_{t2} and d_{t11}, each of which are at a distance of 1.414 from the test sample. This implies that the test sample can be either classified into Class A or Class C.

b. K = 2

For K = 2, the test data is classified based on the 2 closest labeled samples. Once again, the two closest samples are at a distance of 1.414 from the test sample, so the test sample can be classified into Class A or Class C.

c. K = 3

In this case, the test sample is classified based on the 3 closest labelled samples. From the distances, the 3 least distances are d_{t2}, d_{t11} and d_{t1}, with distances of 1.414, 1.414 and 1.73 respectively. Since d_{t1} and d_{t2} correspond to class A and these 2 distances form the majority of the k nearest distances, the test sample will be labelled as Class A.

2. KNN for simple data

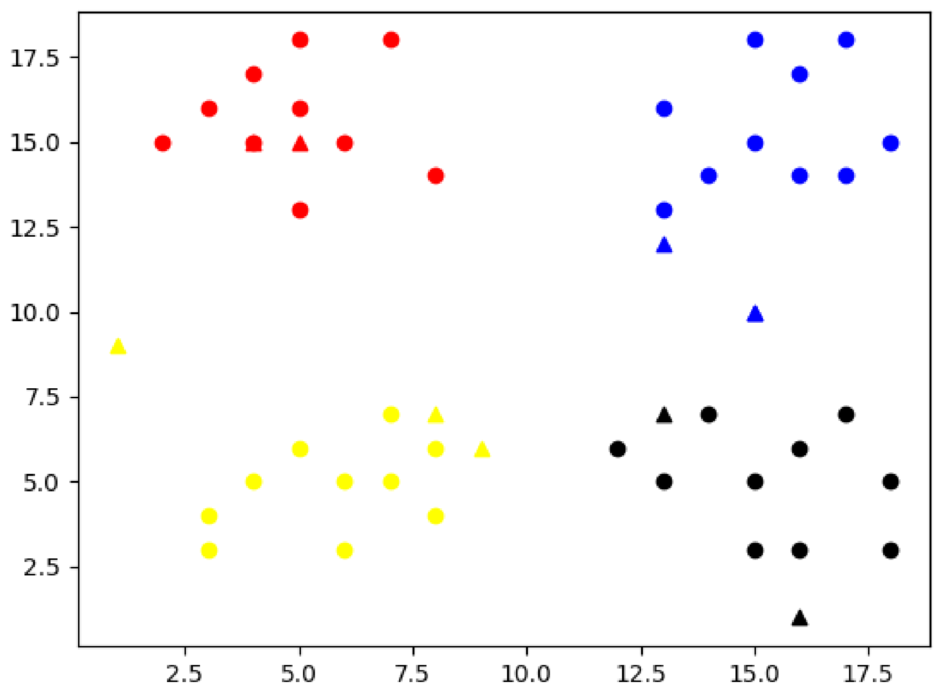
The code for the KNN algorithm is shown below:

```
#########################
# Input your code here #
#########################

# create a 2D array of zeros to store the L2 distances
dist = np.zeros((40, 10))
# run for all samples in the train dataset
for i in range(len(dataSet)):
    # compute the L2 distance for each training sample and test sample
    d = np.sqrt(np.sum((newInput - dataSet[i, :])**2, axis=1))
    # store each resulting value in each row of the distance matrix
    dist[i, :] = d

# in order to find the k-smallest distances over each row, transpose the matrix
dist = np.transpose(dist)

for i in range(len(dist)):
    # obtains the k-smallest distances' indices
    indices = np.argsort(dist[i])[:k]
    # create an array to store the closest class to the test sample
    classes = [0] * 4
    # for each index in the indices
    for j in range(len(indices)):
        # find the label corresponding to that index
        label = mini_train_label[indices[j]]
        # increment that class by 1, indicating the class for the test sample
        classes[label] += 1
    # add the most frequent class to result
    result.append(np.argmax(classes))

####################
# End of your code #
####################
```

The output of the code, with the classified labels for the test samples is shown below:

```
PS D:\Rutgers NB\Courses\Deep Learning\HW1> python miniknn.py
random test points are: [[13  7]
 [15 10]
 [ 9  6]
 [16  1]
 [15 10]
 [ 5 15]
 [13 12]
 [ 1  9]
 [ 4 15]
 [ 8  7]]
knn classfied labels for test: [3, 1, 2, 3, 1, 0, 1, 2, 0, 2]
```

The visualized result of the KNN algorithm for the randomly generated test samples for k=9 is shown below:



3. KNN for Handwritten Digit Recognition

The code for the KNN algorithm is shown below:

```
#########################
# Input your code here #
#########################
# create a 2D array of zeros to store the L2 distances
dist = np.zeros((len(dataSet), len(newInput)))
# run for all samples in the train dataset
for i in range(len(dataSet)):
    for j in range(len(newInput)):
        # compute the L2 distance for each training sample and test sample
        d = np.sqrt(np.sum((newInput[j] - dataSet[i])**2))
        # store each resulting value in each row of the distance matrix
        dist[i, j] = d

# in order to find the k-smallest distances over each row, transpose the matrix
dist = np.transpose(dist)

for i in range(len(dist)):
    # obtains the k-smallest distances' indices from each row of the dist array
    indices = np.argsort(dist[i])[:k]
    # create an array to store the closest class to the test sample
    classes = [0] * 10
    # for each index in the indices
    for j in range(len(indices)):
        # find the label corresponding to that index
        label = labels[indices[j]]
        # increment that class by 1, indicating the class for the test sample
        classes[label] += 1
    # add the most frequent class to result
    result.append(np.argmax(classes))


####################
# End of your code #
####################
```

To determine the number of classes, I found the maximum value of y_train, which was 9. From this, I was able to deduce that there were 10 classes in which the test sample could be classified into.

On running the KNN algorithm for 25 test samples and k=9, the obtained accuracy was 100% with an execution time of about 15s, as shown in the screenshot below:

```
PS D:\Rutgers NB\Courses\Deep Learning\HW1> python knn.py
---classification accuracy for knn on mnist: 1.0 ---
---execution time: 15.481516361236572 seconds ---
```