Spring 2022 Introduction to Deep Learning

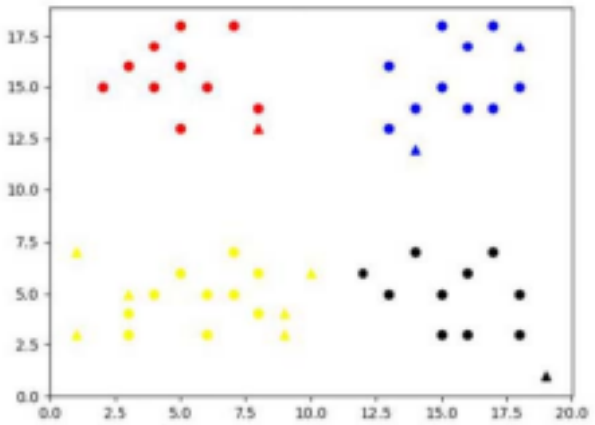
Homework Assignment 1 **Due dates: Feb. 16**

Problem 1 (Practice the computation of KNN). You are required to use KNN to classify a 3-dimension data. The training dataset contains 12 pairs of (data , label) as follows:

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)

What is classified label for test data (1,0,1) when K=1, 2, and 3, respectively? Choose L2 distance as the measurement metric.

Problem 2 (KNN for simple data). There are 40 2-dimension training data and corresponding labels (0~3) have been saved in the "knn_minitrain.npy" and "knn_minitrain_label.npy". Write a KNN classifier with filename "miniknn.py" to classify 10 random generated 2-dimension test data. Visualized result is illustrated as follows, where round and triangle indicate train and test data, respectively. The value of *k* can be chosen between 3~10 .



Note: Part of "miniknn.py" for data loading and plotting has been given. You can utilize them or write your own codes. Also, choose L2 distance as the measurement metric.

Problem 3 (KNN for handwriting digit recognition). In this problem you will

use KNN to recognize handwritten digits.
First, use “download_mnist.py” file to download the MNIST database. This file will make data to following numpy arrays and save it as Pickle. (“mnist.pkl”)

x_train : 60,000x784 numpy array that each row contains flattened version of training images.

y_train : 1x60,000 numpy array that each component is true label of the

corresponding training images.

x_test : 10,000x784 numpy array that each row contains flattened version of test images.

y_test : 1x10,000 numpy array that each component is true label of the

corresponding test images.

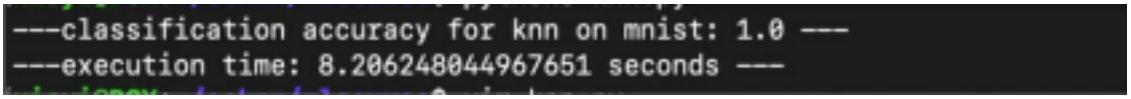
Notice: Once you get “mnist.pkl”, you don't need to call init() anymore.
Everything you need to do is to locate “download_mnist.py” and “mnist.pkl” in your working directory and to call load(). Then you can load the MNIST database in “knn.py”

Notice: Due to the high computational complexity of KNN, you do not need to classify all 10000 test images. Instead, you can select how many test images to classify in line51 and line52 of “knn.py” (e.g. 20 images).

Note: Part of “knn.py” for data loading and plotting has been given. You can utilize them or write your own codes. You can choose the value of *k*. Also, choose L2 distance as the measurement metric.

The final accuracy should be over 95% depending on your test dataset.

Execution time may be different due to the hardware platform and size of test data.



What to submit:

Submit a PDF-version report for three problems, including source codes and print screen results.

Question 1:

Nicolás Rubert

172007365

nar126 HW #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: $(-1,-1,-1), (0,-1,-2), (0,-1,1), (-1,-2,1)$

Classified label for test data $(1,0,1)$

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Calculation for Class A

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

$= \sqrt{1+1+1}$

$= \sqrt{3}$

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

$= \sqrt{1+0+1}$

$= \sqrt{2}$

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

$= \sqrt{0+4+0}$

$= 2$

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

$= \sqrt{0+4+1}$

$= \sqrt{5}$

Calculation for Class B

⑤ $(1,2,2)$ $= \sqrt{(1-1)^2 + (0-2)^2 + (1-2)^2}$

$= \sqrt{0+4+1}$

$= \sqrt{5}$

⑥ $(2,2,2)$ $= \sqrt{(1-2)^2 + (0-2)^2 + (1-2)^2}$

$= \sqrt{1+4+1}$

$= \sqrt{6}$

⑦ $(1,2,-1)$ $= \sqrt{(1-1)^2 + (0-2)^2 + (1-(-1))^2}$

$= \sqrt{0+4+4}$

$= \sqrt{8}$

⑧ $(2,2,3)$ $= \sqrt{(1-2)^2 + (0-2)^2 + (1-3)^2}$

$= \sqrt{1+4+4}$

$= \sqrt{9}$

$= 3$

Calculations for Class C

⑨ $(-1, -1, -1)$

$$= \sqrt{(1-(-1))^2 + (0-(-1))^2 + (1-(-1))^2}$$
$$= \sqrt{4+1+4}$$
$$= \sqrt{9}$$
$$= 3$$

⑩ $(0, -1, -2)$

$$= \sqrt{(1-0)^2 + (0-(-1))^2 + (1-(-2))^2}$$
$$= \sqrt{1+1+9}$$
$$= \sqrt{11}$$

⑪ $(0, -1, 1)$

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

⑫ $(-1, -2, 1)$

$$= \sqrt{(1-(-1))^2 + (0-(-2))^2 + (1-1)^2}$$
$$= \sqrt{4+4+0}$$
$$= \sqrt{8}$$

K = 1

Class A (0, 1, 1) & Class C (0, -1, 1) = $\sqrt{2}$

K = 2

Class A (0, 1, 1) & Class C (0, -1, 1) = $\sqrt{2}$

K = 3

Class A (0, 1, 1) & Class C (0, -1, 1) = $\sqrt{2}$

Class A (0, 1, 0) = $\sqrt{3}$

Question 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=[]

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

    # Range is (0,10) because of the testing size
    for x in range(0, 10):

        # Array to hold the distances
        distance=np.array([])
```

```
        # Calculate the distance between two given points =
sqrt((x-y)^2 + .....)
        for y in range(0, 40):
            # Subtract Testing Point - Training Point
            sub = np.subtract(newInput[x],dataSet[y])
            # Square the result (Testing Point - Training Point) ^ 2
            square= np.power(sub,2)
            # Sum x^2 + y ^ 2
            sum = np.sum(square)
            # Take the resulting value distance and append it to the
array
            distance = np.append(distance,np.array([sum*sum]), axis=0)
            # Store the corresponding label to the
            distance = np.append(distance,np.array([labels[y]]),axis=0)

        # Reshape array into a 2d array
        distance = distance.reshape(40, 2)

        # Sort the distances in the array
        distance = distance[distance[:, 0].argsort(kind='quicksort')]

        # Assign labels for classification
        neighbors = [0] * k

        # Convert the labels into integers
        label = distance[:, 1]
        label = label.astype('int64')

        # Count the labels of the k smallest/nearest distance
        for i in range(0, k):
            neighbors[label[i]] += 1
            i += 1
```

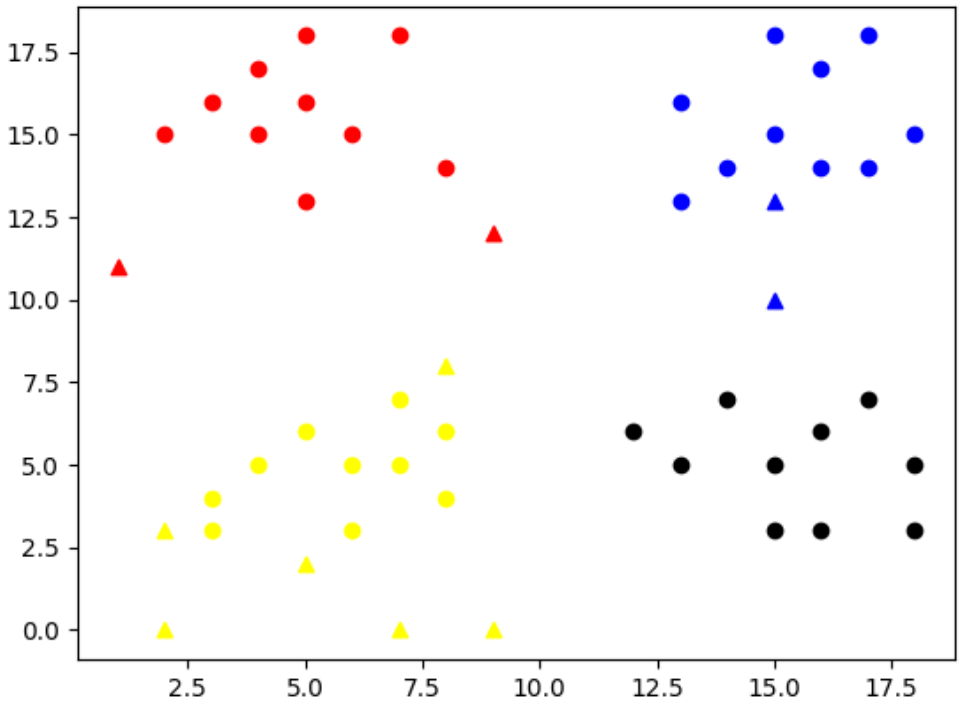
```
        # Max
        maximum = neighbors[0]
        index = 0

        # Find the largest/furthest distance
        for i in range(0, k):
            # Compare distances
            if neighbors[i] > maximum:
                index = i
                # If new max is found replace existing value
                maximum = neighbors[i]
            i += 1

        # Store the result in the result array
        result.append(index)

        # Increase x by 1
        x += 1

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

    return result

outputlabels=kNNClassify(mini_test,mini_train,mini_train_label,4)

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

# plot train data and classified 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 From Terminal:

```
/Users/nico/PycharmProject/DeepLearning/HW1/venv/bin/python /Users/nico/PycharmProject/DeepLearning/HW1/miniknn.py
random test points are: [[ 2  0]
 [ 9 12]
 [15 13]
 [15 10]
 [ 5  2]
 [ 1 11]
 [ 8  8]
 [ 2  3]
 [ 7  0]
 [ 9  0]]
knn classified labels for test: [2, 0, 1, 1, 2, 0, 2, 2, 2, 2]

Process finished with exit code 0
```

Output Image:



Question 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=[]
   ########################
   # Input your code here #
   ########################

   for x in range(0, len(newInput)):

       # Hold the distances
       distance = np.array([])

       # Calculate the distance between two given points =
sqrt((x-y)^2 + .....)
       for y in range(0, 60000):
```

```
            # Subtract Testing Point - Training Point
            sub = np.subtract(newInput[x],dataSet[y])
            # Square the result (Testing Point - Training Point) ^ 2
            square= np.power(sub,2)
            # Sum x^2 + y ^ 2
            sum = np.sum(square)
            # Take the resulting value distance and append it to the
array
            distance = np.append(distance,np.array([sum*sum]), axis=0)
            # Store the corresponding label to the
            distance = np.append(distance,np.array([labels[y]]),axis=0)

        # Reshape array into a 2d array
        distance = distance.reshape(60000, 2)

        # Sort the distances in the array
        distance = distance[distance[:, 0].argsort(kind='quicksort')]

        # Labels are the digits 0 through 9
        neighbors = [0] * 10

        # Convert the labels into integers
        label = distance[:, 1]
        label = label.astype('int64')

        # Count the labels of the k smallest/nearest distance
        for i in range(0, k):
            neighbors[label[i]] += 1
            i += 1

        # Max
        maximum = neighbors[0]
        index = 0
```

```
        # Find the largest/furthest distance
        for i in range(0, 10):
            # Compare distances
            if neighbors[i] > maximum:
                index = i
                # If new max is found replace existing value
                maximum = neighbors[i]
            i += 1

        # Store the result in the result array
        result.append(index)

        # Increase x by 1
        x += 1

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

start_time = time.time()
outputlabels=kNNClassify(x_test[0:20],x_train,y_train,20)
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))
```

Output After Running 20 Images:

I don’t know how it has the accuracy at 100 percent

```
/Users/nico/PycharmProject/DeepLearning/HW1/venv/bin/python /Users/nico/PycharmProject/DeepLearning/HW1/knn.py
---classification accuracy for knn on mnist: 1.0 ---
---execution time: 99.6183009147644 seconds ---

Process finished with exit code 0
```