

Deep Learning Homework I Report

Name:Dazhi Li

RUID:197007456

Problem 1

Source Code:

```
import numpy as np
import heapq
from collections import Counter

#Input the training_data
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)]
sample=(1,0,1)
L2=[]

#Calculate the distance and remember the training_data
for i in class_A:
    L2_dist=np.sqrt((i[0]-sample[0])**2+(i[1]-sample[1])**2+(i[2]-sample[2])**2)
    new={'class':'class_A','L2_dist':L2_dist}
    L2.append(new)
for i in class_B:
    L2_dist=np.sqrt((i[0]-sample[0])**2+(i[1]-sample[1])**2+(i[2]-sample[2])**2)
    new={'class':'class_B','L2_dist':L2_dist}
    L2.append(new)
for i in class_C:
    L2_dist=np.sqrt((i[0]-sample[0])**2+(i[1]-sample[1])**2+(i[2]-sample[2])**2)
    new={'class':'class_C','L2_dist':L2_dist}
    L2.append(new)

#print(L2)

#Define the KNN classifier to get the output label
def KNN_classifier(k):
    if k==1:
        print("\nWhen K=1,")
        shortest=heapq.nsmallest(k+1, L2, key=lambda s: s['L2_dist'])
        if shortest[0]['L2_dist']==shortest[1]['L2_dist']:
            print("Sample point "+str(sample)+" can be classified to :"+\
                  shortest[0]['class']+"/"+shortest[1]['class'])
        else:
            print("Sample point "+str(sample)+" can be classified to :"+\
                  shortest[0]['class'])
    elif k==3:
        print("\nWhen K=3,")
        shortest=heapq.nsmallest(k, L2, key=lambda s: s['L2_dist'])
        A=[]
```

```

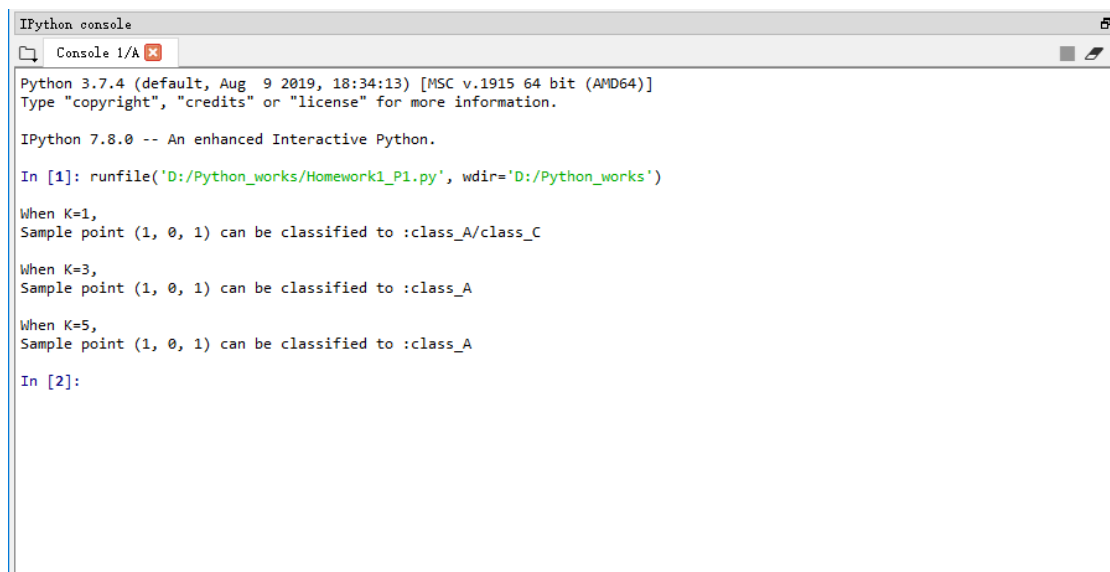
        A.append(shortest[0]['class'])
        A.append(shortest[1]['class'])
        A.append(shortest[2]['class'])
        B=Counter(A).most_common(1)
        if B[0][1]==1:
            print("Sample point "+str(sample)+" can not be classified")
        else:
            print("Sample point "+str(sample)+" can be classified to :"+\
                  B[0][0])
    elif k==5:
        print("\nWhen K=5,")
        shortest=heapq.nsmallest(k, L2, key=lambda s: s['L2_dist'])
        A=[]
        A.append(shortest[0]['class'])
        A.append(shortest[1]['class'])
        A.append(shortest[2]['class'])
        A.append(shortest[3]['class'])
        A.append(shortest[4]['class'])
        B=Counter(A).most_common(1)
        if B[0][1]==1:
            print("Sample point "+str(sample)+" can not be classified")
        else:
            print("Sample point "+str(sample)+" can be classified to :"+\
                  B[0][0])
KNN_classifier(1)
KNN_classifier(3)
KNN_classifier(5)

```

Program Procedure:

At first I input those training data and sample point into my program. Then I calculate the L2 distance between the sample point and all the training data. I use a list-dictionary structure named L2 to store all the L2 distances and their classifications. At first I just printed all the L2 data and found that if K=1, there would be two smallest L2 distances belongs to two different class. So I add some logical judge in KNN_classifier method. There would be some special situations that there would be a point belongs to no one class as what we have seen the "white part" in our slides. Finally, I use heapq library to get the N smallest distances from the L2 structure. And I count how often a label of a class will occur in it as my final KNN output. The program is not so perfect that every kind of datasets would work. But it is still enough to work in most of the situations. Finally, the output screen will be shown below in **figure 1**.

Output of Problem 1:



```
IPython console
Console 1/A
Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.8.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/Python_works/Homework1_P1.py', wdir='D:/Python_works')

When K=1,
Sample point (1, 0, 1) can be classified to :class_A/class_C

When K=3,
Sample point (1, 0, 1) can be classified to :class_A

When K=5,
Sample point (1, 0, 1) can be classified to :class_A

In [2]:
```

Figure 1

Problem 2

Source Code:

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

# 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):
    Inf=999
    result=[]
    #####
    # Input your code here #
    #####
    for i in mini_test:
        L2=[]
        for j in mini_train:
```

```

        L2_dist=np.sqrt((i[0]-j[0])**2+(i[1]-j[1])**2)
        L2.append(L2_dist)
    Min_list=[]
    for a in range(k):
        Min_list.append(L2.index(min(L2)))
        L2[L2.index(min(L2))]=Inf
    classifier=[]
    for b in Min_list:
        classifier.append(mini_train_label[b])
    #print(classifier)
    result.append(Counter(classifier).most_common(1)[0][0])
#####
# End of your code #
#####
    return result

outputlabels=kNNClassify(mini_test,mini_train,mini_train_label,6)

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")

```

Program Procedure:

I use the template of the python code on sakai. But I import counter from collections library and then all my codes are in the defining KNN classifier area. I set an unreachable large number 999 to get the smallest L2 distance and then set it as 999. I use 2 for loop to calculate

the L2 distance between the random generated sample point and the training data. Then I choose the K smallest data from those L2 distances and remember its index. I use the index to get the class of the closest training points. Finally I count the number of the most occurred class as the final result of where this sample point belongs to. The screen shots of output will be posted in **figure 2** and **figure 3**.

Output of Problem2:

```
IPython console
Console 1/A

In [6]: runfile('D:/Dazhi-Project/DeepLearning/Homework1_P2/miniknn.py', wdir='D:/Dazhi-Project/DeepLearning/Homework1_P2')
random test points are: [[16 14]
[18 13]
[12 3]
[10 4]
[14 10]
[ 4 2]
[15 1]
[11 6]
[10 10]
[ 8 16]]
knn classified labels for test: [1, 1, 3, 2, 3, 2, 3, 3, 2, 0]

In [7]:
```

Figure 2

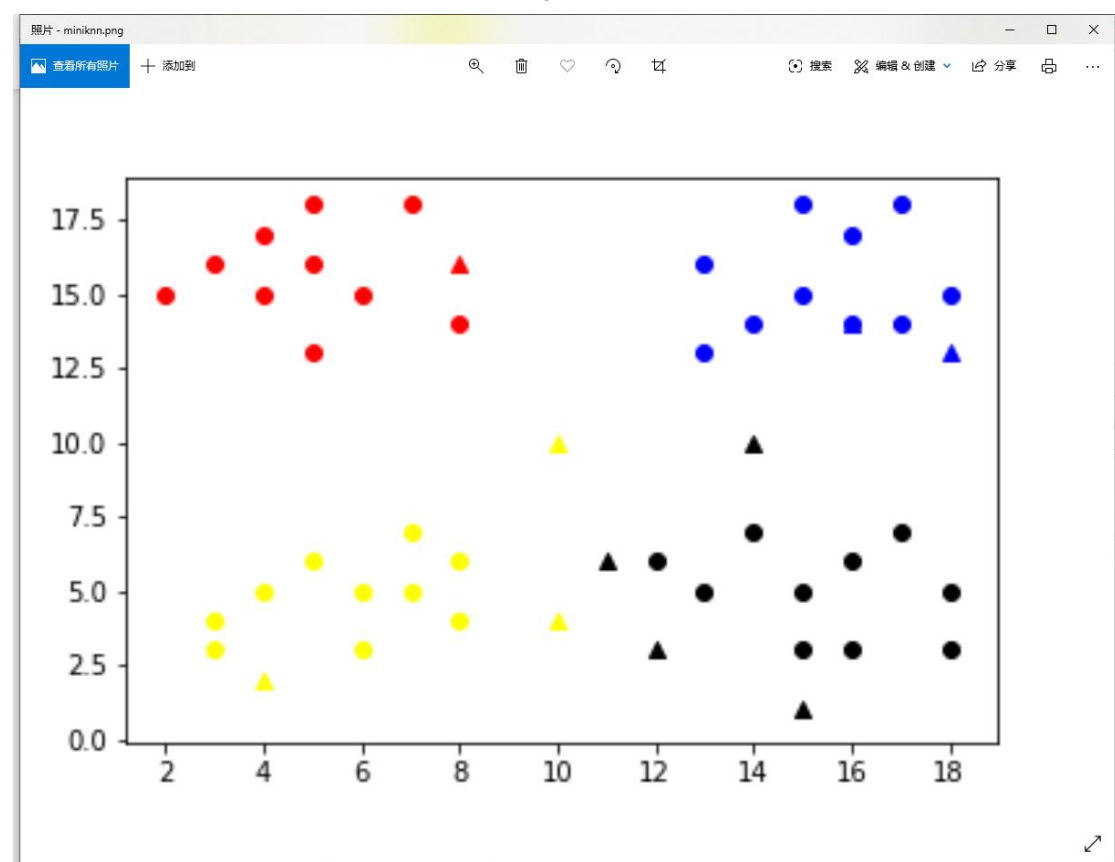


Figure 3

Problem 3

Source Code:

```
#import math

import numpy as np

from download_mnist import load

#import operator

import time

from collections import Counter

# 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 i in newInput:

        L2_dist=0

        L2=[]

        for j in dataSet:

            '''

            This kind of method is low_efficient and low_accurate

            for row in range(28):

                for column in range(28):

                    L2_dist+=np.sqrt((i[row][column]-j[row][column])**2)

            '''

            L2_dist=np.sum(np.sqrt((i-j)**2))

            L2.append(L2_dist)

        Min_list=[]

        for a in range(k):

            Min_list.append(L2.index(min(L2)))

            L2[L2.index(min(L2))]=float("inf")

        classifier=[]

        for b in Min_list:

            classifier.append(labels[b])
```

```

        #print(classifier)

        result.append(Counter(classifier).most_common(1)[0][0])

#####

# End of your code #

#####

return result

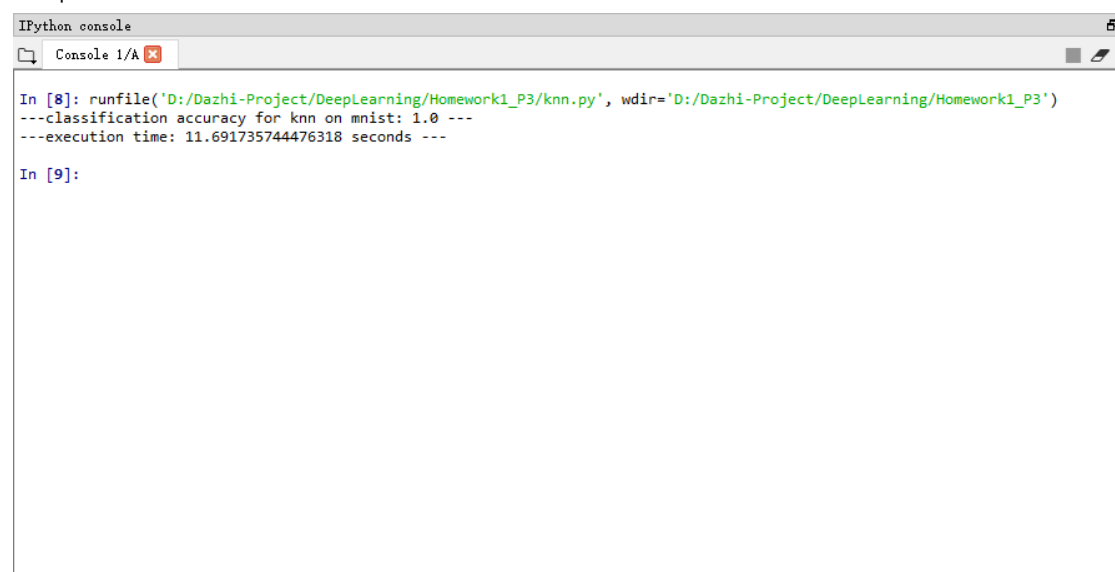
start_time = time.time()
outputlabels=kNNClassify(x_test[0:20],x_train,y_train,10)
result = y_test[0:20] - outputlabels
result = (1 - np.count_nonzero(result)/len(outputlabels))
print("---classification accuracy for knn on mnist: %s ---" %result)
print("---execution time: %s seconds ---" % (time.time() - start_time))

```

Program Procedure:

I also use the template of python codes from the sakai. But I import some packages which is needed and interpret some unused packages. Since there are 60000 training data sets, I calculated there would be a lot of computation about $60000 * 28 * 28 * \text{number_of_testdata}$, which is a huge computation capacity. At first I chose the code as what it is in problem 2 but I found that the final output needed too much time about 1 min for a test data and accuracy was so low about 0.15. So I use the numpy library which could compute array very fast. The way to get K smallest L2 distances and its label is the same as what I do in problem 2. The final output will be shown in **figure 4**(20 test datasets) and **figure 5**(100 test datasets).

Output of Problem 3:



```

IPython console
Console 1/A
In [8]: runfile('D:/Dazhi-Project/DeepLearning/Homework1_P3/knn.py', wdir='D:/Dazhi-Project/DeepLearning/Homework1_P3')
---classification accuracy for knn on mnist: 1.0 ---
---execution time: 11.691735744476318 seconds ---
In [9]:

```

Figure 3-First 20 test datasets

```
28     for row in range(25):
29         for column in range(25):
30             L2_dist=np.sqrt((i[row][column]-j[row][column])**2)
31             ...
32             L2_dist=np.sum(np.sqrt((1-j)**2))
33             L2.append(L2_dist)
34         Min_list=[]
35         for s in range(4):
36             Min_list.append(L2.index(min(L2)))
37             L2[L2.index(min(L2))]=float("inf")
38         classifier=[]
39         for b in Min_list:
40             classifier.append(labels[b])
41         print(classifier)
42         result.append(Counter(classifier).most_common(1)[0][0])
43     43
44     44
45     45
46     =====
47     # End of your code #
48     =====
49     return result
50
51 start_time = time.time()
52 outputLabels=kNNClassifier(y_test[0:100],x_train,y_train,10)
53 result = y_test[0:100] - outputLabels
54 result = (1 - np.count_nonzero(result))/len(outputLabels)
55 print ("---classification accuracy for knn on mnist: %s ---" % result)
56 print ("---execution time: %s seconds ---" % (time.time() - start_time))
57
```

Python console

Console 1/A

In [10]: runfile('D:/Dzhi-Project/DeepLearning/Homework1_3/knn.py', wdir='D:/Dzhi-Project/DeepLearning/Homework1_3')

Related modules: download print

---classification accuracy for knn on mnist: 0.98 ---

---execution time: 58.20815587043762 seconds ---

In [11]:

Figure 5-First 100 test datasets