Intro To Deep Learning
Homework 1
Report

Bingqian Li

bl604

February 16, 2020

1 Practice the computation of KNN

1.1 problem description

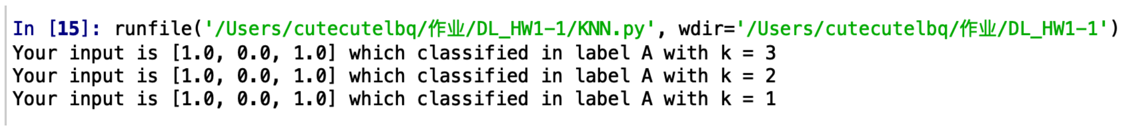
In this problem, I use KNN to classify a 3-dimension data, and the professor provide us the training dataset contains 12 pairs of data and labels. I choose L2 distance as the measurement metric which I have to write a program to implement the KNN classification function. In this function, I compare each feature of the data to be tested with each feature of the training sample data, and then calculate the L2 distance in each pairs and extract the k nearest training sample data, and count the classification labels of the k training sample data. The most frequently occurring labels represent the Category is the category of data to be tested. The Figure 1 shows the whole program.

1.2 Source code description

```
import math
from numpy import *
import operator
import time

def createDataSet():
    # Generate a matrix, each row represents a sample
    group = array([[0.0, 1.0, 0.0], [0.0, 1.0, 1.0], [1.0, 2.0, 1.0], [1.0, 2.0, 0.0],
            [1.0, 2.0, 2.0], [2.0, 2.0, 2.0], [1.0, 2.0, -1.0], [2.0, 2.0, 3.0],
            [-1.0, -1.0, -1.0], [0.0, -1.0, -2.0], [0.0, -1.0, 1.0], [-1.0, -2.0, 1.0]])
    labels = ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C']    #Categories
                                                        to which the samples belong
    return group, labels

def kNNClassify(newInput, dataSet, labels, k):

    numSamples = dataSet.shape[0]

    diff = tile(newInput, (numSamples, 1)) - dataSet  # Difference by element
    squaredDiff = diff ** 2  # Square the difference
    squaredDist = sum(squaredDiff, axis = 1)   # Accumulate by row
    distance = squaredDist ** 0.5  # Sum the squared difference and get the distance

    sortedDistIndices = argsort(distance)   # Sort the distance, return the sorted index
                                                        value
    classCount = {} # define a dictionary
    for i in range(k):
        voteLabel = labels[sortedDistIndices[i]]  # Select k nearest neighbors
        classCount[voteLabel] = classCount.get(voteLabel, 0) + 1   # Count the number of
                                                            occurrences of each category in the
                                                            k nearest neighbors
    maxCount = 0       # Returns the most frequent category labels
    for key, value in classCount.items():
        if value > maxCount:
            maxCount = value
            maxIndex = key


    return maxIndex


dataSet, labels = createDataSet()
testX = ([1.0, 0.0, 1.0])

k = 3
outputLabel = kNNClassify(testX, dataSet, labels, k)
print("Your input is",testX,"which classified in label",outputLabel,"with k = 3")
k = 2
outputLabel = kNNClassify(testX, dataSet, labels, k)
print("Your input is",testX,"which classified in label",outputLabel,"with k = 2")
k = 1
outputLabel = kNNClassify(testX, dataSet, labels, k)
print("Your input is",testX,"which classified in label",outputLabel,"with k = 1")
```

1.3 Screen Results

With the classified label for test data (1,0,1), when K=1, 2, and 3, respectively



```
In [15]: runfile('/Users/cutecutelbq/作业/DL_HW1-1/KNN.py', wdir='/Users/cutecutelbq/作业/DL_HW1-1')
Your input is [1.0, 0.0, 1.0] which classified in label A with k = 3
Your input is [1.0, 0.0, 1.0] which classified in label A with k = 2
Your input is [1.0, 0.0, 1.0] which classified in label A with k = 1
```

Figure 1: The classified label for test data (1,0,1) when K=1, 2, and 3

In the conclusion, the classified label for test data (1,0,1) when K=1 is label A. The classified label for test data (1,0,1) when K=2 is label A. The classified label for test data (1,0,1) when K=3 is also label A.

2 KNN for simple data

2.1 problem description

In this problem, there are 40 2-dimension training data and corresponding labels (0 3) provided by professor in the training data file and corresponding label file. Then, I generate 10 random 2-dimension test data to test the training data, and then do the visualization. Also, I choose L2 distance as the measurement metric and the procedure is very similar to the question 1. However, the most different between question1 and question2 is that I use a for loop to generate 10 epoch, in each epoch, I use each random generated data to do the classification and generate the corresponding label for the specific test data.

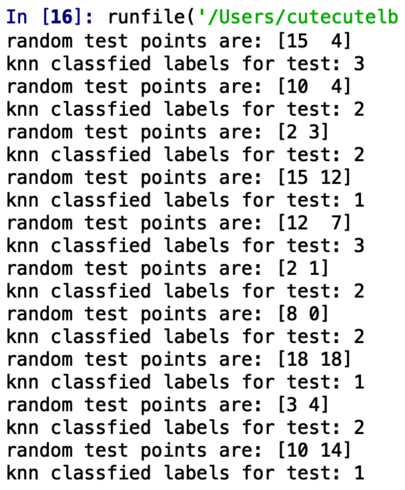
2.2 Source code description

```
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')
```

```python
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):

    numSamples = dataSet.shape[0]
    diff = tile(newInput, (numSamples, 1)) - dataSet  # Difference by element
    squaredDiff = diff ** 2  # Square the difference
    squaredDist = sum(squaredDiff, axis = 1)   # Accumulate by row
    distance = squaredDist ** 0.5  # Sum the squared difference and get the distance

    sortedDistIndices = argsort(distance)   # Sort the distance, return the sorted index
                                                value
    classCount = {} # define a dictionary
    for i in range(k):
        voteLabel = labels[sortedDistIndices[i]]  # Select k nearest neighbors
        classCount[voteLabel] = classCount.get(voteLabel, 0) + 1   # Count the number of
                                                    occurrences of each category in the
                                                    k nearest neighbors

    maxCount = 0      # Returns the most frequent category labels
    for key, value in classCount.items():
        if value > maxCount:
            maxCount = value
            maxIndex = key


    return maxIndex

result = []
for mini_test1 in mini_test:
    outputlabels=kNNClassify(mini_test1,mini_train,mini_train_label,4)
    print ('random test points are:', mini_test1)
    print ('knn classfied labels for test:', outputlabels)
    result.append(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(result)
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")
```

2.3 Screen Results

I use the random generated 10 test data to test the training data and figure out each specific test data corresponding label. The Figure 2 is the result, and the Figure 3 is the visualization of the result, where round and triangle indicate train and test data, respectively. Specifically, I use k = 4 to test the data.



```
In [16]: runfile('/Users/cutecutelb
random test points are: [15  4]
knn classfied labels for test: 3
random test points are: [10  4]
knn classfied labels for test: 2
random test points are: [2 3]
knn classfied labels for test: 2
random test points are: [15 12]
knn classfied labels for test: 1
random test points are: [12  7]
knn classfied labels for test: 3
random test points are: [2 1]
knn classfied labels for test: 2
random test points are: [8 0]
knn classfied labels for test: 2
random test points are: [18 18]
knn classfied labels for test: 1
random test points are: [3 4]
knn classfied labels for test: 2
random test points are: [10 14]
knn classfied labels for test: 1
```

Figure 2: classify 10 random generated 2-dimension test data

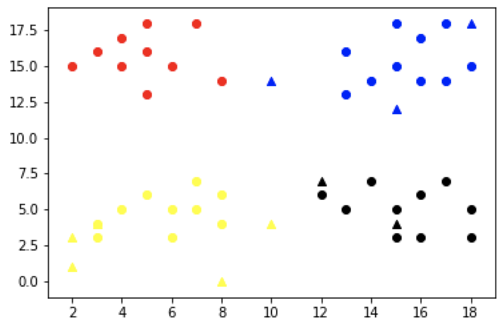


Figure 3: The Visualization of the result

In the conclusion, the classified label for specific 10 random generated test data have different classified labels from 1 to 3. It will take majority vote from K closest points. There zero points classified to label 0 which marks in red color, three points classified to label 1 which marks in blue color, five points classified to label 2 which marks in yellow color and two points classified to label 3 which marks in black color. And the classified label will be changed simultaneous when the random generated number changed.

3 KNN for handwriting digit recognition

3.1 problem description

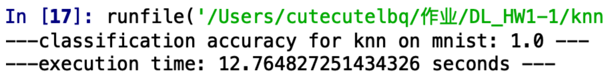
In this problem, I firstly use downloadmnist.py to download the dataset which contains training data and testing data.xtrain dataset contains 60,000x784 numpy array that each row contains flattened version, ytrain dataset contains 1x60,000 numpy array that each component is true label of the corresponding training images. xtest dataset contain 10,000x784 numpy array that each row contains flattened version of test images and ytest dataset contains 1x10,000 numpy array that each component is true label of the corresponding test images. I also use KNN and choose L2 distance as the measurement metric to recognize handwritten digits in the ytest dataset. The final accuracy will be over 95% depending on the test dataset.

3.2 Source code description

```
import math
import numpy as np
from download_mnist import load
import operator
import time

# classify using kNN
x_train, y_train, x_test, y_test = load()

def kNNClassify(newInput, dataset, labels, k):
    result = []
    newInput = newInput.astype(np.float)
    dataset = dataset.astype(np.float)
    labels = labels.astype(np.int)
    for textdata in newInput:
        # caucluate the distance
        numSamples = dataset.shape[0]   # Number of rows, that is, the number of training
                                          samples
        diff = tile(textdata, (numSamples, 1)) - dataset  # Difference by element, turn
                                             the input textdata into a matrix
                                             with the same number of rows and
                                             columns as dataset
        squaredDiff = diff ** 2  # Square the difference
        squaredDist = sum(squaredDiff, axis = 1)   # Accumulate by row
        distance = squaredDist ** 0.5  # Sum the squared difference and get the distance

        sortedDistIndices = argsort(distance)   # Sort the distance, return the sorted
                                             index value
        classCount = {} # define a dictionary
        for i in range(k):
            voteLabel = labels[sortedDistIndices[i]]  # Select k nearest neighbors
            classCount[voteLabel] = classCount.get(voteLabel, 0) + 1   # Count the number
                                                  of occurrences of each category
                                                  in the k nearest neighbors

        maxCount = 0      # Returns the most frequent category labels
        for key, value in classCount.items():
            if value > maxCount:
                maxCount = value
                maxIndex = key
        result.append(maxIndex)

    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))
```

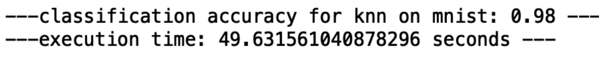
3.3 Screen Results

Due to the high computational complexity of KNN, I don't need to classify all 10000 test images. Instead, I only test 20 images in the test dataset and calculate classification accuracy for KNN on mnist. Specifically, I use k = 10 to test the data. The following Figure 4 and Figure 5 show the results.



```
In [17]: runfile('/Users/cutecutelbq/作业/DL_HW1-1/knn
---classification accuracy for knn on mnist: 1.0 ---
---execution time: 12.764827251434326 seconds ---
```

Figure 4: Using of KNN to recognize handwritten digits when using 20 test data



```
---classification accuracy for knn on mnist: 0.98 ---
---execution time: 49.631561040878296 seconds ---
```

Figure 5: Using of KNN to recognize handwritten digits when using 100 test data

The detailed implementation is that import the mnist data set including the training set and the validation set into the project file, then calculate the distance between the validation set and the training set, and sort the k nearest neighbors from small reach, and obtain the highest category by voting, and judge The pictures of the verification set belong to this category, then the labels of the category and the tags of the verification set are compared. If they match, they are correct, and finally the calculated accuracy rate is output.In the conclusion, I found that when I test 20 images in the test dataset, the classification accuracy for KNN on mnist is 1.0 and the execution time is 12,7648s. However, when I test 100 images in the test dataset, the classification accuracy for KNN on mnist is 0.98 and the execution time is 49.6315.