Spring 2022 Introduction to Deep Learning

Homework Assignment 1

Name: Lohitanvita Rompicharla NetID: lr701

Problem 1 (Practice the computation of KNN):

You are required to use KNN to classify a 3-dimenion 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.

Python Code File: Q1_KNN_3-D.py

```
from math import sqrt


# finding L2 (euclidean) distance between two arrays and returning their
distances
def euclidean_distance(point1, point2):
    distance = 0.0
    for i in range(len(point1)):
        distance += (point1[i] - point2[i]) ** 2
    return sqrt(distance)


# calls euclidean_distance function by iterating through elements of
train_data
# and stores the respective euclidean distance into a list called
nearest_distances
def nearest_neighbors(train_data, test_data, num_neighbors):
    nearest_distances = []
    for train_row in train_data:
        line_dist = euclidean_distance(test_data, train_row)
        nearest_distances.append((train_row, line_dist))
    # sorts the distances in ascending order
    nearest_distances.sort(key=lambda tup: tup[1])
    neighbors = []
    # storing nearest distance compared to k nearest neighbors
    for i in range(num_neighbors):
        neighbors.append(nearest_distances[i][0])
    return neighbors


# calls nearest_neighbors function by passing all the test data, train data
and k-value
```

```
# and classifies the test data into nearest labeled class
def predict_classification(train_data, test_data, num_neighbors):
    k_neighbors = nearest_neighbors(train_data, test_data, num_neighbors)
    result = [row[-1] for row in k_neighbors]
    prediction = max(set(result), key=result.count)
    return prediction


# driver function
if __name__ == '__main__':
    dataset = [[0, 1, 0, "ClassA"], [0, 1, 1, "ClassA"], [1, 2, 1, "ClassA"],
               [1, 2, 0, "ClassA"],
               [1, 2, 2, "ClassB"], [2, 2, 2, "ClassB"], [1, 2, -1, "ClassB"],
               [2, 2, 3, "ClassB"],
               [-1, -1, -1, "ClassC"], [0, -1, -2, "ClassC"],
               [0, -1, 1, "ClassC"], [-1, -2, 1, "ClassC"]]

    test = [1, 0, 1]

    k_value1 = 1

    print("The given test data  with k value as 1 "
          "is classified into: ", predict_classification(dataset, test,
k_value1))

    k_value2 = 2

    print("The given test data  with k value as 2 "
          "is classified into: ", predict_classification(dataset, test,
k_value2))

    k_value3 = 3

    print("The given test data  with k value as 3 "
          "is classified into: ", predict_classification(dataset, test,
k_value3))
```

OUTPUT:

The given test data with k value as 1 is classified into: ClassA

The given test data with k value as 2 is classified into: ClassA

The given test data with k value as 3 is classified into: ClassA

Process finished with exit code 0

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 file name "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.

Python Code File: miniknn.py

```
import numpy as np
import matplotlib as mpl

mpl.use('Agg')
import matplotlib.pyplot as plt


class KNNClassifier:
    # finding L2 (euclidean) distance between two numpy arrays and returning
their distances
    def euclidean_distance(self, point1, point2):
        distance = np.sum((np.square(point1 - point2)))
        return np.sqrt(distance)

    # calls euclidean_distance function by iterating through elements of
train_data
    # and stores the respective euclidean distance and labels into a list
called nearest_distances
    def nearest_neighbors(self, train_data, test_row, train_labels,
num_neighbors):
        nearest_distances = []
        for i in range(len(train_data)):
            line_dist = self.euclidean_distance(test_row, train_data[i])
            nearest_distances.append((train_labels[i], line_dist))
        # sorts the distances in ascending order
        nearest_distances.sort(key=lambda tup: tup[1])
        neighbors = []
        # storing nearest distance compared to k nearest neighbors
        for i in range(num_neighbors):
            neighbors.append(nearest_distances[i][0])
        return neighbors

    def predict_classification(self, train_data, test_data, train_labels,
num_neighbors):
        output_labels = []
        # calls the nearest neighbors function by passing each element of
test data
        for test_row in test_data:
            k_neighbors = self.nearest_neighbors(train_data, test_row,
train_labels, num_neighbors)
            # classifying the test data by counting maximum number of labels
around it
            prediction = max(set(k_neighbors), key=k_neighbors.count)
            output_labels.append(prediction)
        return output_labels
```

```
def plot_knn_classification():
    mini_train = np.load('C:\\Users\\rlohi\PycharmProjects\\pythonProject\\'
                         'Introduction to Deep Learning\\knn_minitrain.npy')
    mini_train_label =
np.load('C:\\Users\\rlohi\PycharmProjects\\pythonProject\\'
                            'Introduction to Deep
Learning\\knn_minitrain_label.npy')
    print("Training Data\n", mini_train)
    # randomly generate test data
    mini_test = np.random.randint(20, size=20)
    mini_test = mini_test.reshape(10, 2)
    k_value = np.random.randint(3, 10)
    print("Testing Data\n",mini_test)

    knn_obj = KNNClassifier()  # object of the class to call its function
    # passing the input to the funtcion
    outputlabels = knn_obj.predict_classification(train_data=mini_train,
                                                  test_data=mini_test,

train_labels=mini_train_label,
                                                  num_neighbors=k_value)

    # plotting the train data in circle shape
    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]

    # plotting the classified test data in triangle shape
    outputlabels = np.array(outputlabels)
    print("Classification of Test Data into classes 1 to 4 respectively :
",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")
```

```
# calling the main function
plot_knn_classification()
```

OUTPUT:

Training Data

[[5 13]

[4 15]

[7 18]

[5 16]

[6 15]

[4 17]

[5 18]

[3 16]

[2 15]

[8 14]

[15 15]

[13 16]

[14 14]

[15 18]

[16 17]

[17 14]

[13 13]

[17 18]

[18 15]

[16 14]

[5 6]

[3 4]

[8 4]

[7 5]

[6 5]

[3 3]

[4 5]

[6 3]

[8 6]

[7 7]

[13 5]

[14 7]

[16 3]

[16 6]

[18 5]

[15 3]

[18 3]

[17 7]

[12 6]

[15 5]]

Testing Data

[[14 6]

[18 18]

[6 13]

[0 5]

[1 15]

[11 11]

[7 2]

[13 19]

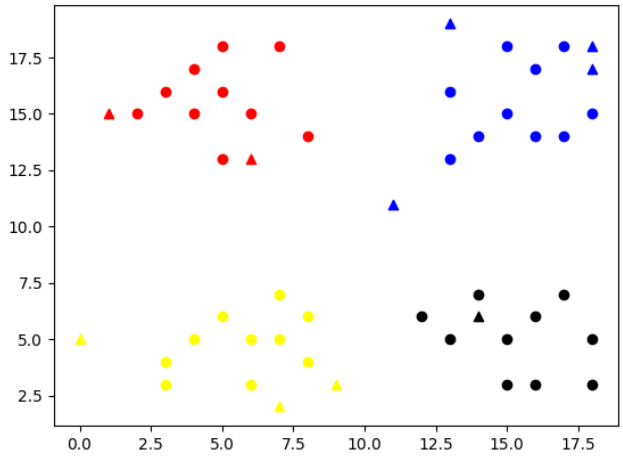
[9 3]

[18 17]]

Classification of Test Data into classes 1 to 4 respectively : [3 1 0 2 0 1 2 1 2 1]

Process finished with exit code 0

Fig: miniknn.png



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.

Python Code File: knn.py

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


class KNNClassifier:
    # finding L2 (euclidean) distance between two numpy arrays and returning
their distances
    def euclidean_distance(self, point1, point2):
        return np.sqrt(np.sum((np.square(point1 - point2))))

    # calls euclidean_distance function by iterating through elements of
```

```
train_data
    # and stores the respective euclidean distance and labels into a list
called nearest_distances
    def nearest_neighbors(self, train_data, test_row, train_labels,
num_neighbors):
        nearest_distances = []
        for i in range(len(train_data)):
            distances = self.euclidean_distance(test_row, train_data[i])
            nearest_distances.append((train_labels[i], distances))
        # sorts the distances in ascending order
        nearest_distances.sort(key=lambda tup: tup[1])
        neighbors = []
        # storing nearest distance compared to k nearest neighbors
        for i in range(num_neighbors):
            neighbors.append(nearest_distances[i][0])
        return neighbors

    def predict_classification(self, train_data, test_data, train_labels,
num_neighbors):
        output_labels = []
        for test_row in test_data:
            # calls the nearest neighbors function by passing each element of
test data
            k_neighbors = self.nearest_neighbors(train_data, test_row,
train_labels, num_neighbors)
            # classifying the test data by counting maximum number of labels
around it
            prediction = max(set(k_neighbors), key=k_neighbors.count)
            output_labels.append(prediction)
        return output_labels


def plot_knn_classification():
    # 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)

    start_time = time.time()  # calculating execution time
    knn_obj = KNNClassifier()  # object of class
    outputlabels = knn_obj.predict_classification(train_data=x_train,
                                                  test_data=x_test[0:40],
                                                  train_labels=y_train,
                                                  num_neighbors=2)
    # calculating the accuracy by comparing with the known test sata
    result = y_test[0:40] - outputlabels
    result = (1 - np.count_nonzero(result) / len(outputlabels))
    print( " classification of 40 images with above algorithm has below
result:")
    print("---classification accuracy for knn on mnist: %s ---" % result)
    print("---execution time: %s seconds ---" % (time.time() - start_time))
```

```
plot_knn_classification()
```

OUTPUT:

classification of 40 images with above algorithm has below result:

---classification accuracy for knn on mnist: 0.95 ---

---execution time: 18.02318286895752 seconds ---

Process finished with exit code 0