מטלה 3

אילן מאיר סופיר : 342615648

שי משה : 318160108

המטלה נעשתה ביחד והתרומה שווה

שאלה 1:

הריצה של הקוד בשאלה הזו היא על דאטא סט הברמן.

תוצאות :

```
Haberman data set:
KNN with k = 1 and p = 1: empirical error: 0.00908496732026143, true error: 0.3341176470588231
KNN with k = 1 and p = 2: empirical error: 0.009215686274509794, true error: 0.32372549019607805
KNN with k = 1 and p = inf: empirical error: 0.010326797385620909, true error: 0.3360784313725486
KNN with k = 3 and p = 1: empirical error: 0.16457516339869258, true error: 0.29614379084967285
KNN with k = 3 and p = 2: empirical error: 0.16359477124182983, true error: 0.297254901960784
KNN with k = 3 and p = inf: empirical error: 0.16686274509803897, true error: 0.28954248366013036
KNN with k = 5 and p = 1: empirical error: 0.1984313725490192, true error: 0.2798692810457513
KNN with k = 5 and p = 2: empirical error: 0.20320261437908457, true error: 0.27673202614379055
KNN with k = 5 and p = inf: empirical error: 0.2001960784313722, true error: 0.2709803921568624
KNN with k = 7 and p = 1: empirical error: 0.2149019607843133, true error: 0.26300653594771206
KNN with k = 7 and p = 2: empirical error: 0.21078431372548978, true error: 0.26601307189542456
KNN with k = 7 and p = inf: empirical error: 0.21098039215686235, true error: 0.26006535947712384
KNN with k = 9 and p = 1: empirical error: 0.22888888888888853, true error: 0.25882352941176434
KNN with k = 9 and p = 2: empirical error: 0.22274509803921527, true error: 0.25627450980392125
KNN with k = 9 and p = inf: empirical error: 0.21856209150326758, true error: 0.25790849673202587
```

1) התוצאות הכי טובות מבחינת הטעות האמיתית התקבלו עם הפרמטרים **k=9** and **p= 2**
2) אפשר לראות כי :
   - הטעות האמיתית יורד והטעות אמפירית עולה ככל שה k עולה עם אותו p=[1,2, ∞].
   - בכללי הטעות האמיתית יורד והטעות אמפירית עולה ככל שה k עולה.
3) בכללי אפשר להגיד כש ה k עולה, המודל הכליל את הדאטא בצורה טובה יותר.
4) אפשר לראות שיש Overfitting כש k=1 כי אנחנו רואים שהטעות האמפירית היא הקטנה ביותר והטעות האמיתית היא הגדולה ביותר.

הריצה של הקוד בסעיף השני הזו היא על דאטא סט של מטלה 2: circle separator.

תוצאות :

```
circle_separator data set:
KNN with k = 1 and p = 1: empirical error: 0.0, true error: 0.06666666666666671
KNN with k = 1 and p = 2: empirical error: 0.0, true error: 0.06866666666666674
KNN with k = 1 and p = inf: empirical error: 0.0, true error: 0.08080000000000005
KNN with k = 3 and p = 1: empirical error: 0.0332, true error: 0.08506666666666671
KNN with k = 3 and p = 2: empirical error: 0.032, true error: 0.08186666666666671
KNN with k = 3 and p = inf: empirical error: 0.03293333333333335, true error: 0.0873333333333334
KNN with k = 5 and p = 1: empirical error: 0.05453333333333335, true error: 0.10893333333333338
KNN with k = 5 and p = 2: empirical error: 0.049600000000000026, true error: 0.10453333333333337
KNN with k = 5 and p = inf: empirical error: 0.051200000000000016, true error: 0.09893333333333328
KNN with k = 7 and p = 1: empirical error: 0.07000000000000005, true error: 0.12186666666666673
KNN with k = 7 and p = 2: empirical error: 0.06653333333333337, true error: 0.12253333333333329
KNN with k = 7 and p = inf: empirical error: 0.07373333333333337, true error: 0.12559999999999996
KNN with k = 9 and p = 1: empirical error: 0.08906666666666671, true error: 0.1497333333333333
KNN with k = 9 and p = 2: empirical error: 0.08493333333333337, true error: 0.13546666666666668
KNN with k = 9 and p = inf: empirical error: 0.09840000000000003, true error: 0.15999999999999995
```

1) התוצאות הכי טובות מבחינת הטעות האמיתית התקבלו עם הפרמטרים **k=1** and **p= 1**
2) אפשר לראות כי :
   - הטעות האמיתית עולה ככל שה k עולה.
   - והטעות אמפירית היא 0 ב k=1 ואז עולה ככל שה k עולה.
   - אם אנחנו עכשיו מנסים להשוות את התוצאות האלה עם התוצאות של הסעיף הקודם אז בסעיף הקודם היה לנו בכללי הטעות האמיתית יורד והטעות אמפירית עולה ככל שה k עולה. ועכשיו בסעיף הזה שתיהן עולות ככל שה k עולה. אני חושב שזה בגלל שפה ה train ו- ה test דומים.
3) אפשר לראות שאין Overfitting כי הטעות המדגמית עולה ביחד עם הטעות האמיתית.

<u>Code :</u>

KNN.py :

```
from itertools import combinations
import Utility

import numpy as np

class KNN:
    def __init__(self, k: int, distance_lp: float = 2):
        self.k = k
        self.T = []
        self.distance_lp = distance_lp

    def fit(self, data: np.ndarray):
        # find the minimum distance between red and blue points
        min_distance = np.inf
        for p1, p2 in combinations(data, 2):
            if (not np.array_equal(p1[:-1], p2[:-1])) and p1[-1] != p2[-1]:
                if Utility.lp_distance(p1[:-1], p2[:-1], self.distance_lp) < min_distance:
                    min_distance = Utility.lp_distance(p1[:-1], p2[:-1], self.distance_lp)

        # add random points to the set T from self.X
        self.T.append(data[0])

        for p in data[1:]:
            for t in self.T:
                if Utility.lp_distance(p, t, self.distance_lp) >= min_distance:
```

```
                    self.T.append(p)
                    break

    def predict(self, X: np.ndarray) -> np.ndarray:
        y = np.zeros(len(X))
        count = 0
        # find the k nearest neighbors of each point in X over the set T
        for p in X:
            distances = []
            for t in self.T:
                distances.append(Utility.lp_distance(p, t[:-1], self.distance_lp))
            distances = np.array(distances)
            k_nearest_neighbors = np.take(self.T, np.argsort(distances)[:self.k], axis=0)
            # predict the label of the point in X by majority voting
            # by counting the number of red and blue points in the k nearest neighbors
            # and assign the label of the majority
            y[count] = np.argmax(np.bincount(k_nearest_neighbors[:, -1].astype(int).tolist()))
            count += 1
        return y
```

Utility.py :

```
from scipy.io.arff import loadarff
import numpy as np

def read_data(filename: str) -> np.ndarray:
    with open(filename) as f:
        lines = f.readlines()
    data = np.zeros((len(lines), 3))
    for i in range(len(lines)):
        data[i] = np.array([float(x) for x in lines[i].split()])
    return data


def read_data_from_csv(filename: str) -> np.ndarray:
    data = loadarff(filename)[0]
    data = np.array([list(x) for x in data])
    data = np.char.decode(data)
    return data

# split the data into train and test in ratio 1:1
def random_train_test_split(data: np.ndarray) -> (np.ndarray, np.ndarray):
    shuffled_data = np.random.permutation(data)
    train_data = shuffled_data[0: int(len(data) * 0.5)]
    test_data = shuffled_data[int(len(data) * 0.5):]
    return train_data, test_data


# calculate distance using Lp norm
def lp_distance(p1: np.ndarray, p2: np.ndarray, p: float = 2) -> float:
    p1 = p1.astype(float)
    p2 = p2.astype(float)
    if p == np.inf:
        return np.max(np.abs(p1 - p2))
    elif p == 0:
        return np.sum(p1 != p2)
    else:
        return np.sum(np.abs(p1 - p2) ** p) ** (1 / p)
```

Main.py :

```
import numpy as np
from KNN import KNN
from Utility import read_data, random_train_test_split, read_data_from_csv

def knn_validation(data: np.ndarray, num_of_iter: int, k: int, distance_lp: float = 2):
    empirical_error = 0
    true_error = 0
    for i in range(num_of_iter):
        knnClassifier = KNN(k=k, distance_lp=distance_lp)
        # split the data into train and test in ratio 1:1
        train_data, test_data = random_train_test_split(data)
        # train the model
        knnClassifier.fit(train_data)
```

```
        # predict the labels of the test data
        y_pred_train = knnClassifier.predict(train_data[:, :-1]).astype(int)
        y_pred_test = knnClassifier.predict(test_data[:, :-1]).astype(int)
        # calculate the error of the model on the train and test data
        empirical_error += np.sum(y_pred_train != train_data[:, -1].astype(int)) / len(train_data)
        true_error += np.sum(y_pred_test != test_data[:, -1].astype(int)) / len(test_data)

    # print the average error of the model on the train and test data
    print(f"KNN with k = {k} and p = {distance_lp}: empirical error: {empirical_error / num_of_iter}"
          f", true error: {true_error / num_of_iter}")


if __name__ == '__main__':
    random_seed = 42
    np.random.seed(random_seed)
    print("Haberman data set:")
    data = read_data_from_csv("Haberman.csv.arff")
    for k in {1, 3, 5, 7, 9}:
        for p in {1, 2, np.inf}:
            knn_validation(data, 100, k, p)

    print("=============================================")
    print("circle_separator data set:")
    data = read_data("circle_separator.txt")
    # replace the labels from {-1, 1} to {0, 1}
    data[:, -1] = (data[:, -1] + 1) / 2
    for k in {1, 3, 5, 7, 9}:
        for p in {1, 2, np.inf}:
            knn_validation(data, 100, k, p)
```