Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчёт
по лабораторной работе № 1 «Метод k ближайших соседей»

Выполнил:

студент группы ИП-013

Копытина Татьяна Алексеевна
ФИО студента

Работу проверил: Дементьева Кристина Игоревна
ФИО преподавателя

Новосибирск 2023 г.

Задание

Суть лабораторной работы заключается в написании классификатора на основе метода k ближайших соседей. Данные из файла необходимо разбить на две выборки, обучающую и тестовую, согласно общепринятым правилам разбиения. На основе этих данных необходимо обучить разработанный классификатор и протестировать его на обеих выборках. В качестве отчёта требуется представить работающую программу и таблицу с результатами тестирования для каждого из 10 разбиений. Разбиение выборки необходимо выполнять программно, случайным образом, при этом, не нарушая информативности обучающей выборки. Разбивать рекомендуется по следующему правилу: делим выборку на 3 равных части, 2 части используем в качестве обучающей, одну в качестве тестовой. Кроме того, обучающая выборка должна быть сгенерирована таким образом, чтобы минимизировать разницу между количеством представленных в ней объектов разных классов, т.е. $abs\left(\left|\{(x_i, y_i) \in X^l | y_i = -1\}\right| - \left|\{(x_i, y_i) \in X^l | y_i = 1\}\right|\right) \to min$.

Первый пункт отвечает за выбор типа классификатора:

Метод парзеновского окна с фиксированным h.

По второму пункту:

В данном варианте используем функцию ядра K(z):

T – треугольное K(x) = (1 – r)[r ≤ 1]

По третьему пункту выбираем 1 базу данных. Вариант (data1.csv)

Код программы

```
import csv
from math import sqrt
from sklearn.model_selection import train_test_split


def loo_h(X_train, h_max: int = 10):
    h_calc = 0
    h_accuracy = 0

    for h in range(2, h_max + 1):
        correct = 0
        incorrect = 0

        for dot in X_train:
            class_0 = 0
            class_1 = 0

            for influence_dot in X_train:
                if not dot == influence_dot:
                    if core(distance(dot, influence_dot) / h) <= 1:
                        if influence_dot[2] == 0:
                            class_0 += 1
                        else:
                            class_1 += 1

            if class_0 + class_1 == 0:
                continue

            if class_0 >= class_1:
                calc = 0
            else:
                calc = 1
            # print(calc)
```

```
            if dot[2] == calc:
                correct += 1
            else:
                incorrect += 1

        accuracy = correct / (correct + incorrect)
        print("Для {0} процент точности = {1}".format(h, accuracy * 100), "%")
        if h_accuracy < accuracy:
            h_accuracy = accuracy
            h_calc = h
    return h_calc

def loo_k(X_train, k_max: int = 10):
    k_calc = 0
    k_accuracy = 0

    for k in range(2, k_max + 1):
        correct = 0
        incorrect = 0

        for dot in X_train:
            class_0 = 0
            class_1 = 0
            with_calculated_distance = []  # [(dot, distance), (dot, distance), (dot, distance)]
            for neighbor_dot in X_train:
                dist = distance(dot, neighbor_dot)
                with_calculated_distance.append((neighbor_dot, dist))

            # with_calculated_distance_sorted = sorted(with_calculated_distance, key=lambda x: x[1])
            # k_div = with_calculated_distance_sorted[k + 1]
            k_div = sorted(with_calculated_distance, key=lambda x: x[1])[k + 1][1]

            for neighbor_dot, dist in with_calculated_distance:
                if not dot == neighbor_dot:
```

```
                if core(dist / k_div) <= 1:
                    if neighbor_dot[2] == 0:
                        class_0 += 1
                    else:
                        class_1 += 1

            if class_0 >= class_1:
                calc = 0
            else:
                calc = 1
            # print(calc)
            if dot[2] == calc:
                correct += 1
            else:
                incorrect += 1
        accuracy = correct / (correct + incorrect)
        print("Для {0} процент точности = {1}".format(k, accuracy * 100), "%")
        if k_accuracy < accuracy:
            k_accuracy = accuracy
            k_calc = k
    return k_calc

def parzen_h(X_test, X_train, h):
    correct = 0

    for dot in X_test:
        class_0 = 0
        class_1 = 0
        for influence_dot in X_train:
            if not (dot[0] == influence_dot[0] and dot[1] == influence_dot[1]):
                if core(distance(dot, influence_dot) / h) <= 1:
                    if influence_dot[2] == 0:
                        class_0 += 1
                    else:
```

```
                class_1 += 1
        if class_0 >= class_1:
            calc = 0
        else:
            calc = 1

        if dot[2] == calc:
            correct += 1
    return correct

def parzen_k(X_test, X_train, k):
    correct = 0

    for dot in X_test:
        class_0 = 0
        class_1 = 0
        with_calculated_distance = []  # [(dot, distance), (dot, distance), (dot, distance)]
        for neighbor_dot in X_train:
            dist = distance(dot, neighbor_dot)
            with_calculated_distance.append((neighbor_dot, dist))

        k_div = sorted(with_calculated_distance, key=lambda x: x[1])[k + 1][1]

        for neighbor_dot, dist in with_calculated_distance:
            if not dot == neighbor_dot:
                if core(dist / k_div) <= 1:
                    if neighbor_dot[2] == 0:
                        class_0 += 1
                    else:
                        class_1 += 1

        if class_0 >= class_1:
            calc = 0
        else:
```

```
            calc = 1
        # print(calc)
        if dot[2] == calc:
            correct += 1
    return correct

def distance(central_dot, influence_dot):
    return sqrt((central_dot[0] - influence_dot[0]) ** 2 + (central_dot[1] - influence_dot[1]) ** 2)

def core(r):
    return abs(1 - r)

X_data = []
with open("C:\MMO\data1.csv") as f:
    csv_iter = csv.reader(f, delimiter=',')
    next(csv_iter)
    for row in csv_iter:
        X_data.append(row)

for row in X_data:
    row[0] = int(row[0])
    row[1] = int(row[1])
    row[2] = int(row[2])

X_train, X_test = train_test_split(X_data, test_size=0.33)
#print("Поиск подходящего h")
h = loo_h(X_train, 3)
print(h)
acc = parzen_h(X_test, X_train, h)
#print("Поиск подходящего k")
#k = loo_k(X_train, 5)
#print(k)
#acc = parzen_k(X_test, X_train, k)
print("Точность на тестовой выборке {0}".format(acc / len(X_test) * 100), "%")
```

Результаты работы

```
PS C:\MMO> & C:/Users/Татьяна/AppData/Local/Programs/Python/Python310/python.exe c:/MMO/lab1.py
Для 2 процент точности = 91.91532772251976 %
Для 3 процент точности = 88.64285714285714 %
2
Точность на тестовой выборке 71.21212121212122 %
PS C:\MMO>
```

Рис1. Точность на выборке 3 h

```
Поиск подходящего h
Для 2 процент точности = 91.39528726502515 %
Для 3 процент точности = 87.91718722113153 %
Для 4 процент точности = 85.1291745431632 %
2
Точность на тестовой выборке 75.18181818181819 %
```

Рис2. Точность на выборке 4 h

```
PS C:\MMO> & C:/Users/Татьяна/AppData/Local/Programs/Python/Python310/python.exe c:/MMO/lab1.py
Поиск подходящего h
Для 2 процент точности = 92.72216547497446 %
2
Поиск подходящего k
Для 2 процент точности = 82.29850746268656 %
2
Точность на тестовой выборке 82.57575757575758 %
PS C:\MMO>
```

Ри3. Точность на выборке 2 h и 2 k

Выводы

В ходе работы я изучила и применила на практике метод k ближайших соседей, а именно метод парзеновского окна с фиксированным h, используя квартическое ядро.