СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“

Факултет по математика и информатика
Катедра „Компютърна информатика“

Дисциплина: СОЗ (3ти курс ИС, зимен семестър 2020/2021)

ЗАДАНИЕ ЗА ДОМАШНА РАБОТА №3

Изготвил: Кристина Георгиева Цекова, Група 1, ФН: 71852

1. Описание на използвания метод за решаване на задачата в свободен формат.

Методът на най-близкия съсед се състои в класификацията на даден тестов пример в зависимост от степента на принадлежност с единствен пример на понятие – този, който се намира на най - близко разстояние от него (чрез Евклидово разстояние). При метода на k най – близки съседи числото k определя броя на екземпляри на класове, които участват в определянето на решението за класификация на тестовия пример. Тестовият пример се класифицира в съответствие с най – често срещания клас измежду неговите k най – близки съседи.

Работим в двумерното пространство, чиято размерност е по – голяма от 0. В таблицата е дадено множество от вече класифицирани индивиди (разположени в пространството). Задачата е да се предвиди вида на покупките на нов индивид, който трябва да бъде разположен в това пространство, т.е. да се класифицира нов тестов пример. Най – напред си създавам клас – IndividualPoint, чиято цел е да създаде индивид по негови координати (спрямо абсцисата – осъществени разходи на клиентите и спрямо ординатата – честота на покупките) и да определи неговата класификация, която приемам, че е Boolean (true – 1 – покупки от специалните оферти в каталога, false – 0 – покупки от стандартната ценова листа). Тоест ако се класифицира с 1, на стандартния изход ще се извежда true, и обратно. Създавам си конструктори, методи за достъп до член – данните на класа, методи за промяна на стойностите на член – данните, както и метод, който да извежда информация за индивидите. Освен това задавам стойности по подразбиране на x и y, като на x задавам 1000, тъй като това е приблизително средата на всички стойности, а на y – 2, което означава, че преобладава “Very often”.

След това създавам друг клас – KNearestNeighbors, който описва същността на метода. Той притежава множество от обучаващи примери, които вече са дадени в таблицата и число – k, което показва спрямо колко негови най – близки съседи ще бъде класифициран тестовия пример. В този клас най – напред си създавам метод – enumerate, чиято цел е да преобразува String в число (честотата от таблицата), за да може да бъде прочетена. След това имам метод, който намира Евклидовото разстояние между два индивида в пространството – euclideanDistance. Методът fit прочита таблицата от .csv файла, разполагайки обучаващите примери в пространството. И последният метод, който създавам в този клас, е методът predict. Той приема като аргументи осъществените разходи на клиентите и честотата на покупките и спрямо тях се намира класификацията на тестовия пример. Накрая имам метод toString, който извежда информация за k и множеството от обучаващи примери.

2. Описание на реализацията с псевдокод.

2.1. Метод enumerate.

```
// create a method which transforms the Frequency from the table with given examples in Integer
private Integer enumerate(String frequency) {
    switch (frequency.strip()) {
        case "Rarely": {
            return 0;
        }
        case "Sometimes": {
            return 1;
        }
        case "Often": {
            return 2;
        }
        case "Very often": {
            return 3;
        }
        default: {
            return null;
        }
    }
}
```

Псевдокод:

```
function enumerate(frequency) returns Integer
    switch(frequency)
    case “Rarely“ then return 0
    case “Sometimes” then return 1
    case “Often” then return 2
    case “Very often” then return 3
    default return null
```

2.2. Метод predict.

Този метод пресъздава методът за класификация спрямо k най – добрите съседи. Идеята на метода е следната: Най – напред създавам индивид, който представлява тестовия пример, като по подразбиране класификацията му е 0 – false. След това създавам приоритетна опашка, в която запазвам всички Евклидови разстояния на класифицираните индивиди. Така те ще бъдат сортирани, като най – добрите разстояния ще бъдат най – отпред в опашката, след което избираме само първите k от тях и класифицираме тестовия пример спрямо тях.

```
// this method make the classification of the testing example
public Boolean predict(Integer amountPayed, String frequency) {
    IndividualPoint testingExample = new IndividualPoint(amountPayed, enumerate(frequency), classification: false);

    // collect all euclidean distances of all classified individuals in a priority queue,
    // so then we can choose only the best k of them
    PriorityQueue<IndividualPoint> allDistances = new PriorityQueue<>((p1, p2) -> {
        Double eucDist1 = euclideanDistance(testingExample, p1); // find one euclidean distance
        Double eucDist2 = euclideanDistance(testingExample, p2); // find second euclidean distance

        // compare both of the distances
        if (eucDist1 < eucDist2) {
            return -1; // if the first is smaller than the second
        } else if (eucDist1.equals(eucDist2)) {
            return 0; // return 0 if they are equal
        }
        return 1; // if the second is smaller than the first distance
    });

    // collect the sorted distances in the queue
    allDistances.addAll(trainingData);

    // make an ArrayList of the first k classified individuals
    ArrayList<Boolean> classifications = new ArrayList<>();

    // iterate from 0 to k and add the classifications of the polled individual
    for (int i = 0; i < k; i++) {
        classifications.add(Objects.requireNonNull(allDistances.poll()).getClassification());
    }

    // finally, compare the classifications - true and false,  and return true if it predominates
    return classifications.stream().filter((x) -> x).count() > classifications.stream().filter((x) -> !x).count();
}
```

Псевдокод:

function predict(amountPayed, frequency) returns the classification
declare and initialize testingExample <- IndividualPoint(amountPayed,
enumerate(frequency), false)
declare and initialize priority queue allDistances <- PriorityQueue((p1, p2) -> compare
two euclideanDistances and sort them)
add the training data (sorted) in the queue allDistances
declare and initialize ArrayList classifications which contains all classifications
for each value i of k do
classification.add{allDistances.getClassification()}
add the classifications of the examples from allDistances to the ArrayList
return the most common classification

2.3. Метод fit.

Този метод прочита данните за таблицата от файла.

```
// this method fills the training data and locate them in the space (reading the .csv file)
public void fit(String filename) {
    BufferedReader openFile; // open the file for reading
    try {
        // get the file
        openFile = new BufferedReader(new FileReader(filename));
        //Read to skip the header
        openFile.readLine();

        String line = "";
        //Reading from the second line
        while ((line = openFile.readLine()) != null) {
            String[] lineData = line.split( regex: ";");

            if (lineData.length > 0) {
                //Save the values in the training data
                trainingData.add(new IndividualPoint(
                                    Integer.parseInt(lineData[0]),
                                    enumerate(lineData[1]),
                            classification: Integer.parseInt(lineData[2]) != 0));
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Псевдокод:

```
function fit(filename)
        declare BufferReader openFile to open the file
        try
                initialize openFile with the file name
                read the file starting from the second line
                while the line is not null
                        save the values in the training data
        catch (exception)
```

2.4. Метод euclideanDistance

Намиране на Евклидовото разстояние между два обекта.

```
// method which calculate the euclidean distance between two individual points
private Double euclideanDistance(IndividualPoint firstIndividual, IndividualPoint secondIndividual) {
    return Math.sqrt(Math.pow((firstIndividual.getX() - secondIndividual.getX()), 2)
            + Math.pow((firstIndividual.getY() - secondIndividual.getY()), 2));
}
```

Псевдокод:

function(firstIndividual, secondIndividual) returns Double
return Euclidean distance

3. Инструкции за компилиране на програмата.

За да тествам вече създадените класове, си правя тестов клас – KNearestNeighborsTest. Задавам стойност на k = 3, тъй като имаме само два класа – 0 и 1, спрямо които трябва да класифицираме новия тестов пример. Ако две от класификациите са 0, то и тестовият пример ще бъде класифициран като 0 – false. И обратното. Прочитам таблицата след което си задавам стойности за 10 различни тестови примера, като в коментар към тях описвам техните 3 най – близки съседи, както и класификацията, която се получава.

```
// this class tests the algorithm
public class KNearestNeighborsTest {
    public static void main(String[] args) {
        // create an object of type KNearestNeighbors with k = 3
        KNearestNeighbors knn = new KNearestNeighbors( k: 3);

        // read the .csv file
        knn.fit( filename: "training_data.csv");
        System.out.println(knn);

        // testing examples which are not in the table with training data
        System.out.println(knn.predict( amountPayed: 650,  frequency: "Rarely")); // 3 nearest neighbors: 680, 691, 703 -> all of them are 0 => false
        System.out.println(knn.predict( amountPayed: 740,  frequency: "Often")); //  3 nearest neighbors: 703, 708, 725 -> all are 0 => false
        System.out.println(knn.predict( amountPayed: 890,  frequency: "Sometimes")); // 3 nearest neighbors: 925, 952, 959 -> common is 0 => false
        System.out.println(knn.predict( amountPayed: 1600,  frequency: "Rarely")); // 3 nearest neighbors: 1562, 1569, 1569 -> all are 1 => true
        System.out.println(knn.predict( amountPayed: 1700,  frequency: "Sometimes")); // 3 nearest neighbors: 1562, 1569, 1569 -> all are 1 => true
        System.out.println(knn.predict( amountPayed: 500,  frequency: "Often")); // 3 nearest neighbors: 600, 602, 604 -> all are 0 => false
        System.out.println(knn.predict( amountPayed: 724,  frequency: "Very often")); // 3 nearest neighbors: 703, 308, 725 -> all are 0 => false
        System.out.println(knn.predict( amountPayed: 1218,  frequency: "Rarely")); // 3 nearest neighbors: 1213, 1213, 1215 -> common is 0 => false
        System.out.println(knn.predict( amountPayed: 1080,  frequency: "Rarely")); // 3 nearest neighbors: 982, 1071, 1076 -> common is 0 => false
        System.out.println(knn.predict( amountPayed: 1238,  frequency: "Rarely")); // 3 nearest neighbors: 1256, 1256, 1256 -> all are 1 => true
        System.out.println(knn.predict( amountPayed: 1360,  frequency: "Sometimes")); // 3 nearest neighbors: 1315, 1377, 1404 -> all are 1 => true
        System.out.println(knn.predict( amountPayed: 1570,  frequency: "Very often")); // 3 nearest neighbors: 1562, 1569, 1569 -> all are 1 => true
    }
}
```

4. Примерни резултати.

Най – напред извеждам информацията от таблицата:

```
"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" "-javaagent:C:
Displaying results for k = 3 nearest neighbors
Training data:
[(Amount_payed: 982, Frequency: 3, Classification: false)
, (Amount_payed: 1304, Frequency: 3, Classification: true)
, (Amount_payed: 1256, Frequency: 3, Classification: true)
, (Amount_payed: 1562, Frequency: 3, Classification: true)
, (Amount_payed: 703, Frequency: 3, Classification: false)
, (Amount_payed: 1213, Frequency: 3, Classification: false)
, (Amount_payed: 1471, Frequency: 3, Classification: true)
, (Amount_payed: 1315, Frequency: 3, Classification: true)
, (Amount_payed: 691, Frequency: 3, Classification: false)
, (Amount_payed: 1439, Frequency: 3, Classification: true)
, (Amount_payed: 1377, Frequency: 2, Classification: true)
, (Amount_payed: 675, Frequency: 2, Classification: false)
, (Amount_payed: 1458, Frequency: 2, Classification: true)
, (Amount_payed: 1294, Frequency: 2, Classification: true)
, (Amount_payed: 611, Frequency: 2, Classification: false)
, (Amount_payed: 959, Frequency: 2, Classification: false)
, (Amount_payed: 1301, Frequency: 2, Classification: true)
, (Amount_payed: 1076, Frequency: 2, Classification: true)
, (Amount_payed: 1569, Frequency: 2, Classification: true)
, (Amount_payed: 680, Frequency: 2, Classification: false)
, (Amount_payed: 978, Frequency: 1, Classification: true)
, (Amount_payed: 600, Frequency: 1, Classification: false)
, (Amount_payed: 654, Frequency: 1, Classification: false)
, (Amount_payed: 708, Frequency: 1, Classification: false)
, (Amount_payed: 604, Frequency: 1, Classification: false)
, (Amount_payed: 925, Frequency: 1, Classification: false)
, (Amount_payed: 1213, Frequency: 1, Classification: false)
, (Amount_payed: 1471, Frequency: 1, Classification: true)
, (Amount_payed: 1215, Frequency: 1, Classification: true)
, (Amount_payed: 1071, Frequency: 1, Classification: false)
, (Amount_payed: 1569, Frequency: 0, Classification: true)
, (Amount_payed: 680, Frequency: 0, Classification: false)
, (Amount_payed: 952, Frequency: 0, Classification: true)
, (Amount_payed: 602, Frequency: 0, Classification: false)
, (Amount_payed: 654, Frequency: 0, Classification: false)
, (Amount_payed: 725, Frequency: 0, Classification: false)
, (Amount_payed: 604, Frequency: 0, Classification: false)
, (Amount_payed: 1256, Frequency: 0, Classification: true)
, (Amount_payed: 1404, Frequency: 0, Classification: true)
, (Amount_payed: 1256, Frequency: 0, Classification: true)
]
```

Резултат от тестването на подадените (некласифицирани) тестови примери.

```
(650, "Rarely") // 3 nearest neighbors: 680, 691, 703 -> all of them are 0 => false
(740, "Often") // 3 nearest neighbors: 703, 708, 725 -> all are 0 => false
(890, "Sometimes") // 3 nearest neighbors: 925, 952, 959 -> common is 0 => false
(1600, "Rarely") // 3 nearest neighbors: 1562, 1569, 1569 -> all are 1 => true
(1700, "Sometimes") // 3 nearest neighbors: 1562, 1569, 1569 -> all are 1 => true
(500, "Often") // 3 nearest neighbors: 600, 602, 604 -> all are 0 => false
(724, "Very often") // 3 nearest neighbors: 703, 308, 725 -> all are 0 => false
(1218, "Rarely") // 3 nearest neighbors: 1213, 1213, 1215 -> common is 0 => false
(1080, "Rarely") // 3 nearest neighbors: 982, 1071, 1076 -> common is 0 => false
(1238, "Rarely") // 3 nearest neighbors: 1256, 1256, 1256 -> all are 1 => true
(1360, "Sometimes") // 3 nearest neighbors: 1315, 1377, 1404 -> all are 1 => true
(1570, "Very often") // 3 nearest neighbors: 1562, 1569, 1569 -> all are 1 => true
```

```
false
false
false
true
true
false
false
false
false
true
true
true
```