СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ"
Факултет по математика и информатика

Дисциплина: Системи, основани на знания

Задание за домашна работа №2

Изготвил: Весела Илиянова Стоянова
Факултетен номер: 71949
Група: 1

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 разработи програмна система, която да получава данни за нов пациент и да предвижда подходящото лекарство за пациента със същото заболяване, като за целта трансформира данните и прилага метода на k най-близки съседи (k-NN), оценява и анализира резултатите. Първоначално създавам клас KNN, чиято цел е да имплементира KNN алгоритъма. Имаме следните член-данни:

- **Private final int k** – показва спрямо колко негови най-близки съседи ще бъде класифициран тестовия пример.
- **Private List<List<Double>> X** – вектор, който съдържа данните от файла. Ще можем да го съпоставим с координатна система.
- **Private List<String> y** – вектор от класовете на X.

В класа KNN имам създаден конструктор, където се проверява дали k е положително число. Имам метод, който намира Евклидовото разстояние между firstSample и

secondSample – **Double euclideanDistance(List<Double> firstSample, List<Double> secondSample)**. След това имам метод **void fit(List<List<Double>> X, List<String> y)**, при който записваме подадените параметри в X и в y. И последният метод, който създавам е методът **public List<String> predict(List<List<Double>> X)**, който използва метода **String predictHelper(List<Double> x)**, който намира разстоянието до всички точки, взима най-близките k и прави majority vote върху тях.

Имам създаден клас Main. В него имам реализирани няколко метода за преобразуване на String в число.

- Методът **private static final Map<String, Double> genderEncoder** записва 1.0 вместо „F" и 0.0 вместо „M".
- Методът **private static final Map<String, Double> bpEncoder** записва 0 вместо „LOW", 1 вместо „NORMAL" и 2 вместо „HIGH".
- Методът **private static final Map<String, Double> cholesterolEncoder** записва 0.0 вместо „NORMAL" и 1.0 вместо „HIGH".
- Методът **void trainTestSplit(List<List<Double>> bps,**
  List<String> y,
  List<List<Double>> X_train,
  List<List<Double>> X_test,
  List<String> y_train,
  List<String> y_test), който връща данни за трениране и данни за тестване.
- Във **main** функцията четем csv файла, правим трансформация към вектор и го подаваме на нашия KNN клас.

2. Описание на реализацията с псевдокод

2.1. Метод **euclideanDistance** – намира евклидовото разстояние между два обекта.

```
28
29 @    public static Double euclideanDistance(List<Double>firstSample, List<Double> secondSample){
30          int size = firstSample.size();
31          double sum = 0;
32
33          for(int i = 0; i < size; i++){
34              sum += Math.pow(firstSample.get(i) - secondSample.get(i), 2);
35          }
36          return Math.sqrt(sum);
37      }
38
```

Псевдокод:
Function(firstSample, secondSample) returns Double
Return Euclidean distance

2.2. Метод **predict** - пресъздава метода за класификация спрямо k най-добрите съседи.

```java
//Find euclidean distance to all points and get the nearest k
private String predictHelper(List<Double> x){

    List<String> nearestNeighbours = X.stream()
            .sorted(Comparator.comparingDouble(s -> euclideanDistance(s, x)))
            .limit(k)
            .map(s -> y.get(X.indexOf(s)))
            .collect(Collectors.toList());

    //Majority vote
    List<String> distinct = nearestNeighbours.stream()
            .distinct()
            .sorted(Comparator.comparingInt(s -> Collections.frequency(nearestNeighbours, s)))
            .collect(Collectors.toList());
    return distinct.get(distinct.size() - 1);
}

public List<String> predict(List<List<Double>> X){
    return X.stream().map(this::predictHelper).collect(Collectors.toList());
}
}
```

Идеята на метода е следната: намираме най-близките k съседа като пресмятам евклидовото разстояние до всички точки и след това правя majority vote върху най-близките k.

2.3. Метод **fit** – записва подадените параметри в X и y.

```java
public void fit(List<List<Double>>X, List<String> y){
    this.X = X;
    this.y = y;
}
```

2.4. Методи за преобразуване на String в число

```java
private static final Map<String, Double> genderEncoder = new HashMap<>(){{
    put("F", 1.0);
    put("M", 0.0);
}};

private static final Map<String, Integer> bpEncoder = new HashMap<>(){{
    put("HIGH", 2);
    put("NORMAL", 1);
    put("LOW", 0);
}};

private static final Map<String, Double> cholesterolEncoder = new HashMap<>() {{
    put("HIGH", 1.0);
    put("NORMAL", 0.0);
}};
```

2.5. Метод за разделяне на данните на train-data и test-data

```java
public static void trainTestSplit(List<List<Double>> bps,
                                  List<String> y,
                                  List<List<Double>> X_train,
                                  List<List<Double>> X_test,
                                  List<String> y_train,
                                  List<String> y_test
) {
    int size = bps.size();
    int limit = (int) Math.round(size * 0.8);

    List<Integer> range = IntStream.range(0, size).boxed().collect(Collectors.toList());
    Collections.shuffle(range);

    for (int i = 0; i <= limit; i++) {
        X_train.add( bps.get(range.get(i)) );
        y_train.add( y.get(range.get(i)) );
    }

    for (int i = limit + 1; i < size; i++) {
        X_test.add( bps.get(range.get(i)) );
        y_test.add( y.get(range.get(i)) );
    }
}
```

3. Инструкции за компилиране на програмата

Във файла Main.java се намира main функцията, от където се стартира програмата. При стартиране на програмата на конзолата се отпечатват данните като матрица – на първия ред има информация за възрастта на всеки пациент, на втория ред – за пола, на третия ред за BP, на четвъртия за холестерола, на петия за Na_to_K и на шестия – за лекарството, което пациента е приемал. След това се отпечатва информация за пациентите, за които искаме да предвидим лекарство. В predicted се отпечатва информация, за това нашата програма кое лекарство би предвидила, а в actual – кое лекарство в действителност трябва да бъде прието. След това се отпечатва информация за броя правилни лекарства и за общия им брой и накрая се отпечатва съотношението между броя лекарства, които сме предвидили правилно и общия брой.

4. Примерни резултати

- [0.0, 0.0, 1.0, 23.0, 1.0, 1.0, 25.355]
  - Predicted – drugY
  - Actual – drugY
- [1.0, 0.0, 0.0, 47.0, 0.0, 1.0, 13.093]
  - Predicted – drugY
  - Actual – drugY
- [1.0, 0.0, 0.0, 47.0, 0.0, 1.0, 10.114]
  - Predicted – drugY

    - Actual – drugY
- [0.0, 1.0, 0.0, 28.0, 1.0, 1.0, 7.798]
    - Predicted – drugC
    - Actual – drugX
- [1.0, 0.0, 0.0, 61.0, 1.0, 1.0, 18.043]
    - Predicted – drugA
    - Actual – drugA
- [0.0, 1.0, 0.0, 22.0, 1.0, 1.0, 8.607]
    - Predicted – drugX
    - Actual – drugX
- [0.0, 1.0, 0.0, 49.0, 1.0, 1.0, 16.275]
    - Predicted – drugB
    - Actual – drugC
- [1.0, 0.0, 0.0, 41.0, 0.0, 1.0, 11.037]
    - Predicted – drugY
    - Actual – drugY
- [0.0, 1.0, 0.0, 60.0, 0.0, 1.0, 15.171]
    - Predicted – drugB
    - Actual – drugX
- [1.0, 0.0, 0.0, 43.0, 0.0, 0.0, 19.368]
    - Predicted – drugX
    - Actual - drugX