

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



**Проект по Системи основани на
знания
Преподавател: Проф. д-р Мария
Нишева-Павлова**

Проект: Оцеляване при Титаник

Изготвили:

Весела Илианова Стоянова 71949,

Велиана Георгиева Кирова 71929,

Севджан Басри Мехмед 71981

Дата: 13.01.2022г.

Съдържание

1. Задание – кратко описание на решаваната задача;
2. Описание (с блок-схема или псевдокод) на използвания/предложения алгоритъм за решаване на задачата;
3. Описание на данните – описание на структурата и произхода на използваните данни;
4. Описание на особеностите на създадения/използвания програмен код;
5. Примерни резултати от работата на създаденото приложение;
6. Оценка на ефективността на реализацията

1. Задание

Целта на проекта е да предскаже какъв тип хора е по-вероятно да оцелеят при потъването на кораба въз основа години, пол, клас и т.н. Например много би било вероятно да оцелее богат човек от първа класа, отколкото човек от трета класа.

Използваме статистически алгоритъм, който се базира на Евклидово разстояние (разстоянието между две точки). Считаме всеки ред от таблицата(данните) като отделен вектор. Съответно нововъведените данни образуват един вектор.

Сравняваме новия вектор с данните от таблицата и изчисляваме най-краткото разстояние (на кой от векторите е подобен най-много) и така определяме дали даден човек ще оцелее или не.

2. Описание с псевдокод.

```
ersonController.java × TitanicKNN.java × Main.java × titanic.csv × homeController.java
public class TitanicKNN {

    private final int k;
    private List<List<Double>> X;
    private List<Integer> y; //vector from X's classes

    public TitanicKNN(int k) { this.k = Math.max(k, 1); }
    //our rows
    public static Double euclideanDistance(List<Double> s1,
                                           List<Double> s2){

        int size = s1.size();
        double sum = 0;

        for (int i = 0; i < size; i++) {
            sum += Math.pow(s1.get(i) - s2.get(i), 2);
        }

        return Math.sqrt(sum);
    }

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

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

private Integer predictHelper(List<Double> x){
    //sort x depending on euclideanDistance
    List<Integer> nearestN = X.stream()
        .sorted(Comparator.comparingDouble(s -> euclideanDistance(s, x)))
        .limit(k) //Get k nearest samples
        .map(s -> y.get(X.indexOf(s))) // get the class of this vector
        .collect(Collectors.toList());

    //Majority vote
    List<Integer> distinct = nearestN.stream()
        .distinct()
        .sorted(Comparator.comparingInt(s -> Collections.frequency(nearestN, s)))
        .collect(Collectors.toList());

    return distinct.get(distinct.size() - 1);
}
```

```
ersonController.java × TitanicKNN.java × Main.java × titanic.csv × homeController.java
@FXML
private TextField txtAccuracy;

private boolean isShipClassValid(String shipClass){
    return shipClass.equals("1") || shipClass.equals("2")
        || shipClass.equals("3");
}

private boolean isNumberStringValid(String ageString){
    if (ageString == null) {
        return false;
    }
    try {
        double age = Double.parseDouble(ageString);
        if (age < 0){
            return false;
        }
    } catch (NumberFormatException exception) {
        return false;
    }
    return true;
}

ersonController.java × TitanicKNN.java × Main.java × titanic.csv × homeController.java
private static Map<String, Double> genderEncoder = new HashMap<>(){
    put("F", 1.0);
    put("M", 0.0);
};

private static void trainTestSplit(List< List<Double>> entities,
                                   List<Integer> y,
                                   List< List<Double>> X_train,
                                   List< List<Double>> X_test,
                                   List<Integer> y_train,
                                   List<Integer> y_test){

    int size = entities.size();
    //percent data for train
    int limit = (int) Math.round(size * 0.7);

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

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

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

```

List<List<String>> X = new ArrayList<>();
List<String> columns = new ArrayList<>();

try (BufferedReader br = new BufferedReader(
    new FileReader( fileName: ".\\src\\titanic.csv"))) {
    String line;
    columns = Collections.singletonList(br.readLine());

    while ((line = br.readLine()) != null) {
        String[] values = line.split( regex: "\\," );
        X.add(Arrays.asList(values));
    }
} catch (IOException e) {
    e.printStackTrace();
}

List<Double> classes = X.stream()
    .map(s -> Double.parseDouble(s.get(0)))
    .collect(Collectors.toList());

List<Double> ages = X.stream()
    .map(s -> Double.parseDouble(s.get(1)))
    .collect(Collectors.toList());

List<Double> genders = X.stream()
    .map(s -> genderEncoder.get(s.get(2)))
    .collect(Collectors.toList());

List<Double> fares = X.stream()
    .map(s -> Double.parseDouble(s.get(3)))
    .collect(Collectors.toList());

List<Integer> survived = X.stream()
    .map(s -> Integer.parseInt(s.get(4)))
    .collect(Collectors.toList());

List<List<Double>> gatheredEntities = new ArrayList<>();

int size = classes.size();
for (int i = 0; i < size; i++) {
    List<Double> gatheredBackToEntities = new ArrayList<>();

    gatheredBackToEntities.add(classes.get(i));
    gatheredBackToEntities.add(ages.get(i));
    gatheredBackToEntities.add(genders.get(i));
    gatheredBackToEntities.add(fares.get(i));

    gatheredEntities.add(i, gatheredBackToEntities);
}

List<List<Double>> X_train = new ArrayList<>();
List<List<Double>> X_test = new ArrayList<>();

List<Integer> y_train = new ArrayList<>();
List<Integer> y_test = new ArrayList<>();

trainTestSplit(gatheredEntities, survived, X_train,
    X_test, y_train, y_test);

TitanicKNN model = new TitanicKNN( k: 1);
model.fit(X_train, y_train);

List<Integer> y_pred = model.predict(X_test);

int correct = 0;
size = y_pred.size();

for (int i = 0; i < size; i++) {
    if(y_pred.get(i).equals(y_test.get(i))) {
        ++correct;
    }
}

//Get the information about the person
List<List<Double>> X_test2 = new ArrayList<>();
List<Double> allData = new ArrayList<>();
allData.add(new PersonShipClass);
allData.add(new PersonAge);
allData.add(new PersonGender);
allData.add(new PersonFare);

X_test2.add( index: 0, allData);

txtYesOrNo.setText(String.valueOf(y_pred.get(0)));
txtAccuracy.setText(String.valueOf((
    (double) correct / (double) size)));
}

```

3. Описание на данните.

Използвани са реални данни, базирани на хората, които са загинали и оцелели при истинската катастрофа на Титаник.

Ресурс:

https://www.kaggle.com/c/titanic/data?select=gender_submission.csv

4. Описание на особеностите на създадения програмен код.

В проекта се използва принципът за разделянето на наличните данни на две, а именно train data и test data. Train data ще бъдат данните, от които моделът ще се учи, а test data – данните, по които ще съдим колко добре се справя моделът с непозната за него информация.

KNN класа ще използваме за репрезентация на k-Nearest-Neighbors моделът. Той съдържа полетата:

- k – броят на най-близките съседи
- List<List<Double>> X – лист от вектори, съдържащи информация (съответно вектори, съдържащи стойностите за class, age, sex, fare)
- List<String> y – лист от резултатите (survived)

Функцията fit се използва за трениране на модела, т.е. за да му подадем данните, от които ще се учи.

Функцията **euclideanDistance**, която изчислява разстоянието между два вектора.

След като тренирането е завършено, вече може да използваме функцията predict, за да правим прогнози за резултата (дали човекът ще оцелее или не). За реализиране на функцията predict се използва помощна функция predictHelper, която сортира елементите от всеки вектор на X според Евклидовото разстояние, взема резултатите на най-добрите k от тях и им прилага Majority vote, което им придава някаква стойност.

AddPersonController класът съдържа останалата логика за изпълнението на задачата.

В него взимаме данните, които потребителят е въвел като първо правим проверка дали данните са коректни и предсказваме дали човекът е оцелял или не, процентът на точност на предсказаният резултат.

За да се изчисли този процент първо се прочитат данните от дадения файл titanic.csv. След това те се преразпределят в List елементи

по **Pclass, age, sex, fare, survived**. Тъй като колоната **sex** съдържа String стойности, създаваме за нея специален енкодинг за преобразуването им в число (**sex** (F = 1, M = 0)).

След това информацията се разделя на **train** и **test** с помощта на метода **trainTestSplit**. **Train** подаваме на **fit** метода на нашия модел, а **test** – на неговия **predict** метод.

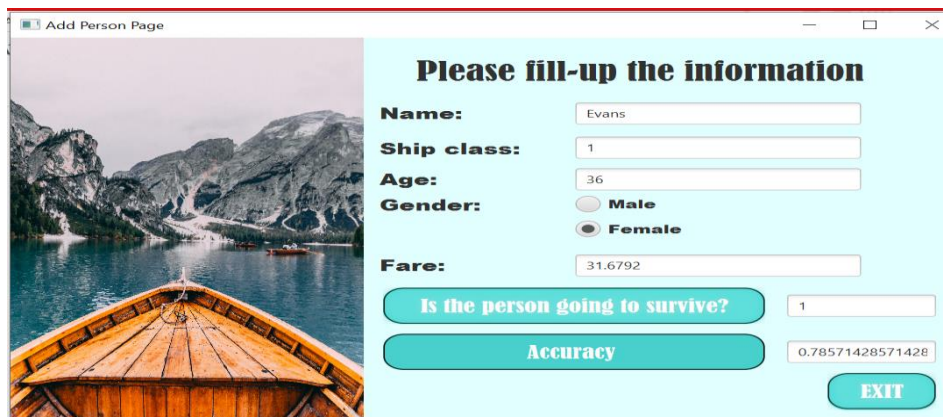
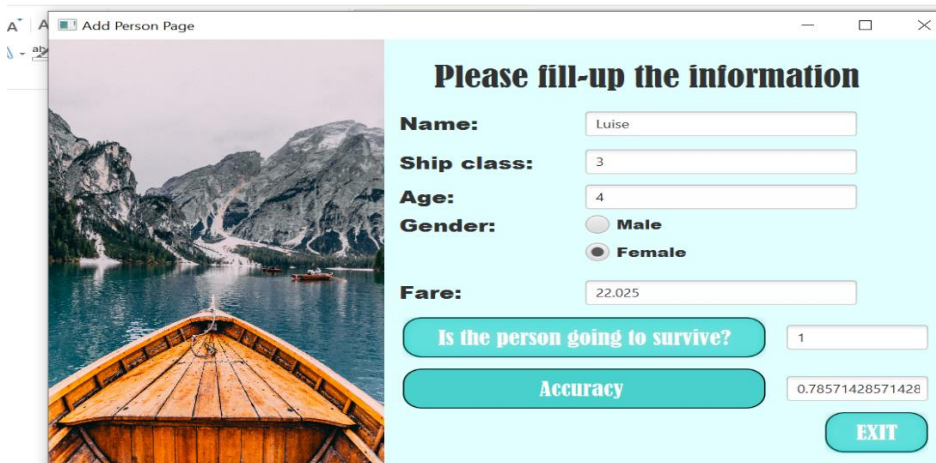
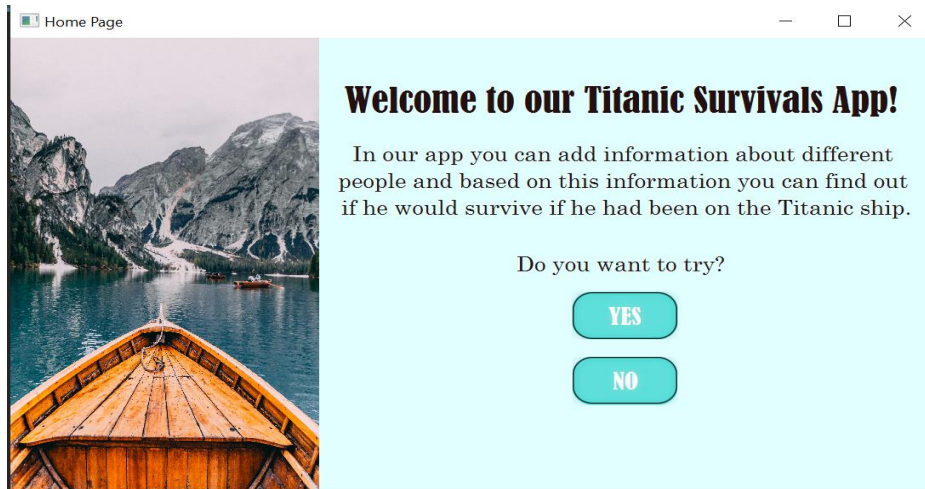
Накрая съпоставяме резултатите, които моделът предсказва, с реалните резултати и изчисляваме коректността на получения резултат.

5. Примерни резултати от работата на създаденото приложение и оценка на ефективността на реализацията.


Първо имаме начална страница с кратко описание и ако искаме да въведем данни и да видим вероятността дали човекът ще оцелее кликаме на **YES**, ако не на **NO**, след което изпълнението на програмата се прекратява.

Данни за случайни пътници в Титаник:

Name	Pclass	Age	Sex	Fare	Survive
Kink-Heilmann, Miss. Luise Gretchen	3	4	female	22.025	1
Evans, Miss. Edith Corse	1	36	female	31.6792	1
Carlsson, Mr. Carl Robert	3	24	male	7.8542	0
Lurette, Miss. Elise	1	58	female	146.5208	1
Van der hoef, Mr. Wyckoff	1	61	Male	33.5	0
Buckley, Miss. Katherine	3	18.5	female	7.2833	1



Add Person Page



Please fill-up the information

Name:

Ship class:

Age:

Gender: ☒ Male ☐ Female


Fare:

Is the person going to survive?

Accuracy

EXIT

Add Person Page



Please fill-up the information

Name:

Ship class:

Age:

Gender: ☐ Male ☒ Female


Fare:

Is the person going to survive?

Accuracy

EXIT

Add Person Page



Please fill-up the information

Name:

Ship class:

Age:

Gender: ☒ Male ☐ Female

Fare:

Is the person going to survive?

Accuracy

EXIT



Please fill-up the information

Name:

Ship class:

Age:

Gender: ☐ Male ☒ Female

Fare:

Is the person going to survive?

Accuracy

EXIT

6. Инструкции за компилиране.

За компилиране на програмата ще ви е нужно да имате инсталиран и настроен: [JDK](#), [JavaFX Windows SDK](#), [Scene Builder](#)

За настройване може да е полезен следния линк:

<https://www.jetbrains.com/help/idea/javafx.html#troubleshoot>

След което можете да стартирате програмата.

7. Препоръки за възможни подобрения

- Да можем да добавяме нови записи в csv файла през конзолата/графичния интерфейс
- Да добавим повече записи в csv файла, за да работи алгоритъмът по-точно
- kNN алгоритъма е често използван и сравнително прост, но има много недостатъци.

Недостатъци на k-NN:

1. Ефективността на k-NN е ниска и всички данни трябва да се изчисляват за всяка класификация, което отнема много време, когато данните са големи.

2. Когато размерът на извадката е много небалансиран, точността на прогнозиране на k-NN е ниска.

3. Заема много памет, тъй като трябва да получи всички съхранени данни, за да изчисли.

4. Стойността на k не е лесно да бъде избрана бързо и оптималната ситуация трябва да се получи чрез сравнение. Но съществува алгоритъм, който подобрява ефективността и класификационната точност на k-NN с помощта на метода за предварителна обработка на данните - това е K-means алгоритъма, който може да използваме, ако искаме да подобрим нашето приложение.