Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Ижевский государственный технический университет имени М.Т. Калашникова»

Кафедра «Программное обеспечение»

Отчет

по лабораторной работе №5 по теме

«Построение дерева решений по алгоритму ID3»

по дисциплине

«Математические основы искусственного интеллекта»

|  |  |
| --- | --- |
| Выполнил:  студенты гр. Б06-191-1 | Мусин Р.И.  Чапаева А.С. |
| Принял: | Коробейников А.В. |

Ижевск 2018

СОДЕРЖАНИЕ ЗАДАНИЯ

В рамках дисциплины искусственного интеллекта познакомиться с принципами построения дерева принятия решений на основе алгоритма ID3.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Алгоритм ID3 — один из алгоритмов для построения дерева принятия решений.

Суть алгоритма:

1. Взять все неиспользованные признаки и посчитать их энтропию относительно тестовых образцов
2. Выбрать признак, для которого энтропия минимальна (а информационная выгода соответственно максимальна)
3. Сделать узел дерева, содержащий этот признак

ID3:

1. Если все примеры положительны, то возвратить узел с меткой «+».
2. Если все примеры отрицательны, то возвратить узел с меткой «-».
3. Если множество признаков пустое, то возвратить узел с меткой, которая больше других встречается в значениях целевого признака в примерах.
4. Иначе:

- A — признак, который лучше всего классифицирует примеры (с максимальной информационной выгодой).

- Создать корень дерева решения; признаком в корне будет являться A.

- Для каждого возможного значения A (v\_i):

- Добавить новую ветвь дерева ниже корня с узлом со значением A=v\_i

- Выделить подмножество Examples(v\_i) примеров, у которых A=v\_i.

- Если подмножество примеров пусто, то ниже этой новой ветви добавить узел с меткой, которая больше других встречается в значениях целевого признака в примерах.

- Иначе, ниже этой новой ветви добавить поддерево, вызывая рекурсивно ID3(Examples(v\_i), Целевой признак, Признаки)

5. Возвратить корень.

ВХОДНЫЕ ДАННЫЕ

Наша программа работает на примере задачи, рассматривающей принятие решений по оформлению кредита на основе кредитной истории, доходов, и наличию поручителей.

|  |  |  |  |
| --- | --- | --- | --- |
| **Кредитная история** | **Риск** | **Поручительство** | **Доход** |
| Плохая | Высокий | Нет | 10 |
| Неизвестна | Высокий | Нет | 30 |
| Неизвестна | Низкий | Нет | 30 |
| Неизвестна | Низкий | Нет | 10 |
| Неизвестна | Низкий | Нет | 50 |
| Неизвестна | Высокий | Адекватное | 50 |
| Плохая | Низкий | Нет | 10 |
| Плохая | Низкий | Адекватное | 50 |
| Хорошая | Низкий | Нет | 50 |
| Хорошая | Высокий | Адекватное | 50 |
| Хорошая | Высокий | Нет | 10 |
| Хорошая | Высокий | Нет | 30 |
| Хорошая | Высокий | Нет | 30 |
| Плохая | Высокий | Нет | 50 |

ЛИСТИНГ ПРОГРАММЫ

**package** main.java;  
  
**import** java.io.IOException;  
**import** java.nio.file.Paths;  
**import** java.util.\*;  
**import** java.util.function.Function;  
**import** java.util.stream.Collectors;  
  
**class** Algorithm {  
 **private static final int *ENTITY\_COUNT*** = 14;  
 **private static final** List<String> ***PROPERTIES*** = Arrays.*asList*(  
 **"Кредитная история"**,  
 **"Долг"**,  
 **"Поручительство"**,  
 **"Доход"** );  
  
 Algorithm() {  
 **try** {  
 Scanner in;  
 List<Entity> entities = **new** ArrayList<>(***ENTITY\_COUNT***);  
 in = **new** Scanner(Paths.*get*(**"examples.txt"**));  
 **for** (**int** i = 0; i < ***ENTITY\_COUNT***; i++)  
 entities.add(Entity.*read*(in));  
 in = **new** Scanner(Paths.*get*(**"answers.txt"**));  
 **for** (**int** i = 0; i < ***ENTITY\_COUNT***; i++)  
 entities.get(i).**answer** = in.next();  
  
 Vertex root = **new** Vertex();  
 buildTree(entities, root);  
 root.print(0);  
 } **catch** (IOException e) {  
 **throw new** RuntimeException(e);  
 }  
 }  
  
 **private void** buildTree(List<Entity> activeEntities, Vertex vertex) {  
 **if** (groupByAnswer(activeEntities).size() == 1) {  
 vertex.**name** = activeEntities.get(0).**answer**;  
 **return**;  
 }  
  
 String maxInformativityProperty = ***PROPERTIES***.stream()  
 .filter(p -> activeEntities.stream().map(e -> e.**properties**.get(p)).distinct().count() > 1)  
 .max(Comparator.*comparingDouble*(p -> calculatePropertyInformativity(activeEntities, p)))  
 .orElse(**null**);  
  
 vertex.**name** = maxInformativityProperty;  
  
 groupByPropertyValue(activeEntities, maxInformativityProperty).forEach((value, entities) -> {  
 Vertex nextVertex = **new** Vertex();  
 vertex.**adj**.add(**new** Edge(value, nextVertex));  
 buildTree(entities, nextVertex);  
 });  
 }  
  
 **private** Map<String, List<Entity>> groupByPropertyValue(List<Entity> entities, String propertyName) {  
 **return** entities.stream()  
 .collect(Collectors.*groupingBy*(e -> e.**properties**.get(propertyName)));  
 }  
  
 **private** Map<String, List<Entity>> groupByAnswer(List<Entity> entities) {  
 **return** entities.stream()  
 .collect(Collectors.*groupingBy*(e -> e.**answer**));  
 }  
  
 **private double** log2(**double** x) {  
 **return** Math.*log*(x) / Math.*log*(2);  
 }  
  
 **private double** calculateAnswersInformativity(List<Entity> entities) {  
 **return** groupByAnswer(entities).values().stream()  
 .mapToDouble(curEntities -> (**double**) curEntities.size() / entities.size())  
 .map(x -> x \* log2(x))  
 .sum();  
 }  
  
 **private double** calculatePropertyInformativity(List<Entity> entities, String propertyName) {  
 **return** groupByPropertyValue(entities, propertyName).values().stream()  
 .mapToDouble(cur -> calculateAnswersInformativity(cur) \* cur.size() / entities.size())  
 .sum();  
 }  
  
 **static class** Entity {  
 String **answer**;  
 Map<String, String> **properties**;  
  
 Entity(Map<String, String> properties) {  
 **this**.**properties** = properties;  
 }  
  
 **static** Entity read(Scanner in) {  
 **return new** Entity(***PROPERTIES***.stream()  
 .collect(Collectors.*toMap*(Function.*identity*(), p -> in.next())));  
 }  
 }  
  
 **class** Vertex {  
 String **name**;  
 List<Edge> **adj** = **new** ArrayList<>();  
  
 **void** print(**int** depth) {  
 **for** (**int** i = 0; i < depth; i++) System.***out***.print(**" "**);  
 System.***out***.println(**name**.toUpperCase());  
 **for** (Edge e : **adj**) {  
 **for** (**int** i = 0; i < depth; i++) System.***out***.print(**" "**);  
 System.***out***.println(**" "** + e.**value**);  
 e.**to**.print(depth + 1);  
 }  
 }  
 }  
  
 **class** Edge {  
 String **value**;  
 Vertex **to**;  
  
 Edge(String value, Vertex to) {  
 **this**.**value** = value;  
 **this**.**to** = to;  
 }  
 }  
}

ПРИМЕР РАБОТЫ

ДОХОД

50

КРЕДИТНАЯ ИСТОРИЯ

Хорошая

LOW

Плохая

MIDDLE

Неизвестна

LOW

30

КРЕДИТНАЯ ИСТОРИЯ

Хорошая

MIDDLE

Плохая

HIGH

Неизвестна

ДОЛГ

Низкий

MIDDLE

Высокий

HIGH

10

HIGHa

ВЫВОД

В ходе работы были изучены принципы построения дерева принятия решений на основе алгоритма ID3. На языке высокого уровня была написана программа, исполняющая данный алгоритм.