Донецкий Национальный Технический Университет

Лабораторная работа № 2

«Автоматизированное приобретение знаний из баз данных»

Выполнил:

ст. группы ИПЗ -13

Лысенко А. С.

Проверил:

асс. каф ПМИ

Павловский Е.В.

Покровск 2017

Название: Автоматизированное приобретение знаний из баз данных

Цель работы: освоить метод индуцирования знаний из баз данных и получить навыки разработки алгоритма приобретения знаний по технологии Data Mining.

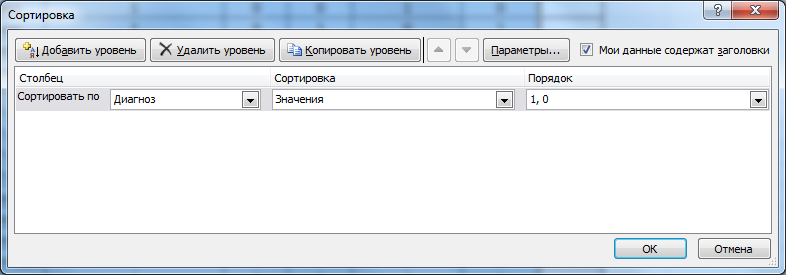
Исходная база данных, из которой извлекаются знания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Диагноз | повышенная температура | насморк | кашель | мышечная боль | упадок сил |
| ОРЗ | 1 | 1 | 1 | 1 | 0 |
| ОРЗ | 0 | 1 | 0 | 1 | 0 |
| ОРЗ | 0 | 1 | 1 | 0 | 0 |
| ОРЗ | 1 | 0 | 0 | 1 | 0 |
| ОРЗ | 0 | 1 | 1 | 1 | 0 |
| Грипп | 1 | 1 | 1 | 0 | 1 |
| Грипп | 1 | 0 | 0 | 1 | 0 |
| Грипп | 1 | 0 | 1 | 0 | 1 |
| Грипп | 1 | 1 | 0 | 0 | 1 |
| Грипп | 1 | 0 | 1 | 1 | 1 |
| Простуда | 0 | 0 | 0 | 1 | 0 |
| Простуда | 1 | 1 | 0 | 0 | 1 |
| Простуда | 0 | 1 | 0 | 1 | 0 |
| Простуда | 0 | 1 | 0 | 1 | 1 |
| Простуда | 1 | 1 | 0 | 0 | 0 |

Искомый атрибут «Диагноз» будем называть атрибутом класса.

Для построения дерева решений нужно взять один из атрибутов таблицы в качестве основного (корневого) атрибута. Пусть это будет «Диагноз».

Преобразуем исходную таблицу к следующему виду (сортируем по графе Диагноз)



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Диагноз | повышенная температура | насморк | кашель | мышечная боль | упадок сил |
| Грипп | 1 | 1 | 1 | 0 | 1 |
| Грипп | 1 | 0 | 0 | 1 | 0 |
| Грипп | 1 | 0 | 1 | 0 | 1 |
| Грипп | 1 | 1 | 0 | 0 | 1 |
| Грипп | 1 | 0 | 1 | 1 | 1 |
| ОРЗ | 1 | 1 | 1 | 1 | 0 |
| ОРЗ | 0 | 1 | 0 | 1 | 0 |
| ОРЗ | 0 | 1 | 1 | 0 | 0 |
| ОРЗ | 1 | 0 | 0 | 1 | 0 |
| ОРЗ | 0 | 1 | 1 | 1 | 0 |
| Простуда | 0 | 0 | 0 | 1 | 0 |
| Простуда | 1 | 1 | 0 | 0 | 1 |
| Простуда | 0 | 1 | 0 | 1 | 0 |
| Простуда | 0 | 1 | 0 | 1 | 1 |
| Простуда | 1 | 1 | 0 | 0 | 0 |

Поскольку теперь для атрибута класса наше дерево решений выводит однозначный ответ, то дерево решений построено.

Порождаем правила:

1. Если повышенная температура = да, насморк = да, кашель = да, мышечная боль = нет и упадок сил = да, то Грипп
2. Если повышенная температура = да, насморк = нет, кашель = нет, мышечная боль = да и упадок сил = нет, то Грипп
3. Если повышенная температура = да, насморк = нет, кашель = да, мышечная боль = нет и упадок сил = да, то Грипп
4. Если повышенная температура = да, насморк = да, кашель = нет, мышечная боль = нет и упадок сил = да, то Грипп
5. Если повышенная температура = да, насморк = нет, кашель = да, мышечная боль = да и упадок сил = да, то Грипп
6. Если повышенная температура = да, насморк = да, кашель = да, мышечная боль = да и упадок сил = нет, то ОРЗ
7. Если повышенная температура = нет, насморк = да, кашель = нет, мышечная боль = да и упадок сил = нет, то ОРЗ
8. Если повышенная температура = нет, насморк = да, кашель = да, мышечная боль = нет и упадок сил = нет, то ОРЗ
9. Если повышенная температура = да, насморк = нет, кашель = нет, мышечная боль = да и упадок сил = нет, то ОРЗ
10. Если повышенная температура = нет, насморк = да, кашель = да, мышечная боль = да и упадок сил = нет, то ОРЗ
11. Если повышенная температура = нет, насморк = нет, кашель = нет, мышечная боль = да и упадок сил = нет, то Простуда
12. Если повышенная температура = да, насморк = да, кашель = нет, мышечная боль = нет и упадок сил = да, то Простуда
13. Если повышенная температура = нет, насморк = да, кашель = нет, мышечная боль = да и упадок сил = нет, то Простуда
14. Если повышенная температура = нет, насморк = да, кашель = нет, мышечная боль = да и упадок сил = да, то Простуда
15. Если повышенная температура = да, насморк = да, кашель = нет, мышечная боль = нет и упадок сил = нет, то Простуда

**Общее описание алгоритма C4.5**

**Обозначения**

*T* — множество примеров (таблица или подтаблица данных); (15)

*m* — количество условных атрибутов (столбцов таблицы) (5)

|*T* | — мощность множества примеров (количество строк в таблице или подтаблице данных); (15)

*C***1** , *C***2** , …, *C****k*** — значения, принимаемые атрибутом класса; (5)

X — текущий условный атрибут, по которому мы хотим провести разбиение

***A*1** , *A***2** , …, *A****n*** — значения, принимаемые текущим условным атрибутом;

Например, вероятность того, что будет Грипп, ОРЗ или Простуда, составляет

*P* = 5 / 15 = 0,33

Рассмотрим поведение алгоритма C4.5.

1. Рассчитаем Gain(*X*) для всех условных атрибутов исходной таблицы.

Info(*T*) = -(0,33·log2(0.33) + 0,33·log2(0.33)) = -(-0,53 – 0,53) = 1,06

Расчёт критерия разбиения для атрибута «**Повышенная температура (ПТ)**»

Info(*T*1) = -(5/5 · log2(5/5)) =-(1\* 0) = 0.

Info(*T*2) = -(2/5· log2(2/5)) =-(0.4 \* (-1.32)) = 0,21.

Info(*T*3) = -(2/5· log2(2/5)) =-(0.4 \* (-1.32)) = 0,21.

InfoПТ(*T*) = 5/15 · 0 + 2/15 · 0,21 + 2/15 · 0,21= 0+ 0,0273 +0,0273= 0,055;

**Gain(ПТ) = 1,06 – 0,055 = 1,005.**

Расчёт критерия разбиения для атрибута «**Насморк**»

Info(*T*1) = -(2/5 · log2(2/5)) =-(0,4\* (-1,32)) = 0,53.

Info(*T*2) = -(4/5· log2(4/5)) =-(0.8 \* (-0.32)) = 0,26.

Info(*T*3) = -(4/5· log2(4/5)) =-(0.8 \* (-0.32)) = 0,26.

InfoНАСМОРК(*T*) = 2/15 · 0,53 + 4/15 · 0,26 + 4/15 · 0,26= 0,069+ 0,068+0,068 = 0,204;

**Gain(Насморк) = 1,06 – 0,204= 0,856.**

Расчёт критерия разбиения для атрибута «**Кашель**»

Info(*T*1) = -(3/5 · log2(3/5)) =-(0,6\* (-0,74)) = 0,44.

Info(*T*2) = -(3/5· log2(3/5)) =-(0.6 \* (-0.74)) = 0,44.

Info(*T*3) = -(0/5· log2(0/5)) =-(0 \* 0)= 0.

InfoКАШЕЛЬ(*T*) = 3/15 · 0,44 + 3/15 · 0,44 + 0/15 · 0 = 0,088+ 0,088+0 = 0,176;

**Gain(Кашель) = 1,06 – 0,176= 0,884.**

Расчёт критерия разбиения для атрибута «**Мышечная боль (МБ)**»

Info(*T*1) = -(2/5 · log2(2/5)) =-(0,4\* (-1,32)) = 0,53.

Info(*T*2) = -(4/5· log2(4/5)) =-(0.8 \* (-0.32)) = 0,26.

Info(*T*3) = -(3/5· log2(3/5)) =-(0.6 \* (-0.74)) = 0,44.

InfoМБ(*T*) = 2/15 · 0,53 + 4/15 · 0,26 + 3/15 · 0,44 = 0,069+ 0,068+0,088 = 0,225;

**Gain(МБ) = 1,06 – 0,225= 0,835.**

Расчёт критерия разбиения для атрибута «**Упадок сил (УС)**»

Info(*T*1) = -(4/5· log2(4/5)) =-(0.8 \* (-0.32)) = 0,26.

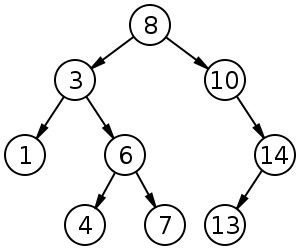
Info(*T*2) = -(0/5· log2(0/5)) =-(0 \* 0)= 0.

Info(*T*3) = -(2/5 · log2(2/5)) =-(0,4\* (-1,32)) = 0,53.

InfoУС(*T*) = 4/15 · 0,26 + 0/15 · 0 + 2/15 · 0,53 = 0,068+ 0+0,069= 0,137;

**Gain(УС) = 1,06 – 0,137= 0,923.**

Критерий разбиения Gain(*Xi*) максимально для условного атрибута «**Повышение температуры (ПТ)**»



SimpleTree.java

**package** tree.set;  
  
**import** java.util.Iterator;  
**import** java.util.LinkedList;  
**import** java.util.List;  
  
**public class** SimpleTree<E> **implements** Tree<E> {  
 **private static** Object *Object*;  
 **private int size** = 0;  
 **private** Leaf<E> **root**;  
 **private** List<E> **list**;  
  
 **public** SimpleTree() {  
 **list** = **new** LinkedList<E>();  
 **root** = **new** Leaf<>(**null**);  
 }  
  
 @Override  
 **public boolean** add(E e) {  
 **if** (**size** == 0)  
 **return** initRootLeaf(e);  
  
 Leaf<E> newNode = **new** Leaf<>(e);  
 Leaf<E> lastNode = findLastLeaf(**root**, newNode);  
  
 **if** (lastNode == **null**)  
 **return false**;  
  
 **size**++;  
 newNode.**parent** = lastNode;  
 **if** (lastNode.compareTo((E) newNode) < 0) {  
 lastNode.**right** = newNode;  
 **return true**;  
 } **else** {  
 lastNode.**left** = newNode;  
 **return true**;  
 }  
  
 }  
  
 *//ищу последний лист дерева* **private** Leaf<E> findLastLeaf(**final** Leaf<E> oldLeaf,  
 **final** Leaf<E> newLeaf) {  
 Leaf<E> lastLeaf = oldLeaf;  
 **int** compare = oldLeaf.compareTo((E) newLeaf);  
  
 **if** (compare < 0 && oldLeaf.**right** != **null**) {  
 lastLeaf = findLastLeaf(oldLeaf.**right**, newLeaf);  
 **return** lastLeaf;  
 }  
 **if** (compare > 0 && oldLeaf.**left** != **null**) {  
 lastLeaf = findLastLeaf(oldLeaf.**left**, newLeaf);  
 **return** lastLeaf;  
 }  
 **if** (compare == 0)  
 **return null**;  
  
 **return** lastLeaf;  
 }  
  
 **private boolean** initRootLeaf(**final** E e) {  
 **root**.**element** = e;  
 **size**++;  
 **return true**;  
 }  
  
 @Override  
 **public** List<E> get() {  
 **for** (E e : **this**)  
 **list**.add(e);  
 **return list**;  
 }  
  
 @Override  
 **public int** size() {  
 **return size**;  
 }  
  
 @Override  
 **public** Leaf find(E e) {  
  
 Leaf<E> eLeaf = **new** Leaf<>(e);  
 **return** binarySearch(**root**, eLeaf);  
 }  
  
 **private** Leaf binarySearch(Leaf<E> leaf, Leaf<E> eLeaf) {  
 **int** compare = leaf.compareTo((E) eLeaf);  
  
 **if** (compare < 0 && leaf.**right** != **null**)  
 **return** binarySearch(leaf.**right**, eLeaf);  
  
 **if** (compare > 0 && leaf.**left** != **null**)  
 **return** binarySearch(leaf.**left**, eLeaf);  
 **if** (compare == 0)  
 **return** leaf;  
  
 **return null**;  
 }  
  
 @Override  
 **public** Iterator<E> iterator() {  
 **return new** Iterator<E>() {  
 **int count** = 0;  
 Iterator<Leaf<E>> **iterator** = **new** TreeIterator<>(**root**);  
  
 @Override  
 **public boolean** hasNext() {  
 **return iterator**.hasNext();  
 }  
  
 @Override  
 **public** E next() {  
 **count**++;  
 **return iterator**.next().**element**;  
 }  
 };  
 }  
  
 **class** Leaf<E> **implements** Comparable<E> {  
 **private** Leaf<E> **parent**;  
 **private** Leaf<E> **right**;  
 **private** Leaf<E> **left**;  
 **private** E **element**;  
  
 **private** Leaf(E element) {  
 **this**.**element** = element;  
 }  
  
 **public** E getElement() {  
 **return element**;  
 }  
  
 @Override  
 **public int** compareTo(E o) {  
 Leaf<E> node = (Leaf<E>) o;  
 **return this**.hashCode() - node.hashCode();  
 }  
  
 @Override  
 **public int** hashCode() {  
 **int** hash = 31;  
 hash = hash \* 17 + **element**.hashCode();  
 **return** hash;  
 }  
 }  
  
 **private class** TreeIterator<E> **implements** Iterator<Leaf<E>> {  
 **private** Leaf<E> **next**;  
  
 **private** TreeIterator(Leaf<E> root) {  
 **next** = root;  
 goToLeftmost();  
 }  
  
 **private void** goToLeftmost() {  
 **while** (**next**.**left** != **null**)  
 **next** = **next**.**left**;  
 }  
  
 @Override  
 **public boolean** hasNext() {  
 **return next** != **null** && **next**.**element** != **null**;  
 }  
  
 @Override  
 **public** Leaf<E> next() {  
 Leaf<E> r = **next**;  
 **if** (**next**.**right** != **null**)  
 **return** goRight(r);  
  
 **return** goUp(r);  
 }  
  
 **private** Leaf<E> goRight(Leaf<E> r) {  
 **next** = **next**.**right**;  
 **while** (**next**.**left** != **null**)  
 **next** = **next**.**left**;  
  
 **return** r;  
 }  
  
 **private** Leaf<E> goUp(Leaf<E> r) {  
 **while** (**true**) {  
 **if** (**next**.**parent** == **null**) {  
 **next** = **null**;  
 **return** r;  
 }  
 **if** (**next**.**parent**.**left** == **next**) {  
 **next** = **next**.**parent**;  
 **return** r;  
 }  
 **next** = **next**.**parent**;  
 }  
 }  
 }  
  
 **public static void** main(String[] args) {  
 SimpleTree<Object> tree = **new** SimpleTree<>();  
 tree.add(8);  
 tree.add(3);  
 tree.add(10);  
 tree.add(1);  
 tree.add(6);  
 tree.add(14);  
 tree.add(4);  
 tree.add(7);  
 tree.add(13);  
  
 System.***out***.println(**"Количество атрибуов: "** + tree.size());  
  
  
  
 System.***out***.println(**"Список атрибутов:"** + tree.get());  
  
 **int** index = (**int**) tree.find(7).**parent**.**element**;  
 System.***out***.println(**"Родитель атрибута 7:"** + tree.find(7).**parent**.**element**);  
 Object element1 = tree.find(index).**parent**.**element**;  
 System.***out***.println(**"Родитель атрибута "** + index + **":"** + element1);  
 Object element2 = tree.find(element1).**parent**.**element**;  
 System.***out***.println(**"Родитель атрибута "** + element1 + **":"** +element2);  
 }  
}

Tree.java

**package** tree.set;  
  
**import** java.util.List;  
  
**public interface** Tree<E> **extends** Iterable<E>{  
 **boolean** add(E e);  
 List<E> get();  
 **int** size();  
 SimpleTree.Leaf find(E e);  
}

