决策树

package ai6;

import java.util.HashMap;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Set;

public class DicisionTree {//决策树

public static void main(String[] args) throws Exception {

String[] attrNames = new String[] { "AGE", "INCOME", "STUDENT",

"CREDIT\_RATING" };

// 读取样本集

Map<Object, List<Sample>> samples = readSamples(attrNames);

// 生成决策树

Object decisionTree = generateDecisionTree(samples, attrNames);

// 输出决策树

outputDecisionTree(decisionTree, 0, null);

}

/\*\*

\* 读取已分类的样本集，返回Map：分类 -> 属于该分类的样本的列表

\*/

static Map<Object, List<Sample>> readSamples(String[] attrNames) {

// 样本属性及其所属分类（数组中的最后一个元素为样本所属分类）

Object[][] rawData = new Object[][] {

{ "<30 ", "High ", "No ", "Fair ", "0" },

{ "<30 ", "High ", "No ", "Excellent", "0" },

{ "30-40", "High ", "No ", "Fair ", "1" },

{ ">40 ", "Medium", "No ", "Fair ", "1" },

{ ">40 ", "Low ", "Yes", "Fair ", "1" },

{ ">40 ", "Low ", "Yes", "Excellent", "0" },

{ "30-40", "Low ", "Yes", "Excellent", "1" },

{ "<30 ", "Medium", "No ", "Fair ", "0" },

{ "<30 ", "Low ", "Yes", "Fair ", "1" },

{ ">40 ", "Medium", "Yes", "Fair ", "1" },

{ "<30 ", "Medium", "Yes", "Excellent", "1" },

{ "30-40", "Medium", "No ", "Excellent", "1" },

{ "30-40", "High ", "Yes", "Fair ", "1" },

{ ">40 ", "Medium", "No ", "Excellent", "0" } };

// 读取样本属性及其所属分类，构造表示样本的Sample对象，并按分类划分样本集

Map<Object, List<Sample>> ret = new HashMap<Object, List<Sample>>();

for (Object[] row : rawData) {

Sample sample = new Sample();

int i = 0;

for (int n = row.length - 1; i < n; i++)

sample.setAttribute(attrNames[i], row[i]);

sample.setCategory(row[i]);

List<Sample> samples = ret.get(row[i]);

if (samples == null) {

samples = new LinkedList<Sample>();

ret.put(row[i], samples);

}

samples.add(sample);

}

return ret;

}

/\*\*

\* 构造决策树

\*/

static Object generateDecisionTree(

Map<Object, List<Sample>> categoryToSamples, String[] attrNames) {

// 如果只有一个样本，将该样本所属分类作为新样本的分类

if (categoryToSamples.size() == 1)

return categoryToSamples.keySet().iterator().next();

// 如果没有供决策的属性，则将样本集中具有最多样本的分类作为新样本的分类，即投票选举出分类

if (attrNames.length == 0) {

int max = 0;

Object maxCategory = null;

for (Entry<Object, List<Sample>> entry : categoryToSamples

.entrySet()) {

int cur = entry.getValue().size();

if (cur > max) {

max = cur;

maxCategory = entry.getKey();

}

}

return maxCategory;

}

// 选取测试属性

Object[] rst = chooseBestTestAttribute(categoryToSamples, attrNames);

// 决策树根结点，分支属性为选取的测试属性

Tree tree = new Tree(attrNames[(Integer) rst[0]]);

// 已用过的测试属性不应再次被选为测试属性

String[] subA = new String[attrNames.length - 1];

for (int i = 0, j = 0; i < attrNames.length; i++)

if (i != (Integer) rst[0])

subA[j++] = attrNames[i];

// 根据分支属性生成分支

@SuppressWarnings("unchecked")

Map<Object, Map<Object, List<Sample>>> splits =

/\* NEW LINE \*/(Map<Object, Map<Object, List<Sample>>>) rst[2];

for (Entry<Object, Map<Object, List<Sample>>> entry : splits.entrySet()) {

Object attrValue = entry.getKey();

Map<Object, List<Sample>> split = entry.getValue();

Object child = generateDecisionTree(split, subA);

tree.setChild(attrValue, child);

}

return tree;

}

/\*\*

\* 选取最优测试属性。最优是指如果根据选取的测试属性分支，则从各分支确定新样本

\* 的分类需要的信息量之和最小，这等价于确定新样本的测试属性获得的信息增益最大

\* 返回数组：选取的属性下标、信息量之和、Map(属性值->(分类->样本列表))

\*/

static Object[] chooseBestTestAttribute(

Map<Object, List<Sample>> categoryToSamples, String[] attrNames) {

int minIndex = -1; // 最优属性下标

double minValue = Double.MAX\_VALUE; // 最小信息量

Map<Object, Map<Object, List<Sample>>> minSplits = null; // 最优分支方案

// 对每一个属性，计算将其作为测试属性的情况下在各分支确定新样本的分类需要的信息量之和，选取最小为最优

for (int attrIndex = 0; attrIndex < attrNames.length; attrIndex++) {

int allCount = 0; // 统计样本总数的计数器

// 按当前属性构建Map：属性值->(分类->样本列表)

Map<Object, Map<Object, List<Sample>>> curSplits =

/\* NEW LINE \*/new HashMap<Object, Map<Object, List<Sample>>>();

for (Entry<Object, List<Sample>> entry : categoryToSamples

.entrySet()) {

Object category = entry.getKey();

List<Sample> samples = entry.getValue();

for (Sample sample : samples) {

Object attrValue = sample

.getAttribute(attrNames[attrIndex]);

Map<Object, List<Sample>> split = curSplits.get(attrValue);

if (split == null) {

split = new HashMap<Object, List<Sample>>();

curSplits.put(attrValue, split);

}

List<Sample> splitSamples = split.get(category);

if (splitSamples == null) {

splitSamples = new LinkedList<Sample>();

split.put(category, splitSamples);

}

splitSamples.add(sample);

}

allCount += samples.size();

}

// 计算将当前属性作为测试属性的情况下在各分支确定新样本的分类需要的信息量之和

double curValue = 0.0; // 计数器：累加各分支

for (Map<Object, List<Sample>> splits : curSplits.values()) {

double perSplitCount = 0;

for (List<Sample> list : splits.values())

perSplitCount += list.size(); // 累计当前分支样本数

double perSplitValue = 0.0; // 计数器：当前分支

for (List<Sample> list : splits.values()) {

double p = list.size() / perSplitCount;

perSplitValue -= p \* (Math.log(p) / Math.log(2));

}

curValue += (perSplitCount / allCount) \* perSplitValue;

}

// 选取最小为最优

if (minValue > curValue) {

minIndex = attrIndex;

minValue = curValue;

minSplits = curSplits;

}

}

return new Object[] { minIndex, minValue, minSplits };

}

/\*\*

\* 将决策树输出到标准输出

\*/

static void outputDecisionTree(Object obj, int level, Object from) {

for (int i = 0; i < level; i++)

System.out.print("|-----");

if (from != null)

System.out.printf("(%s):", from);

if (obj instanceof Tree) {

Tree tree = (Tree) obj;

String attrName = tree.getAttribute();

System.out.printf("[%s = ?]\n", attrName);

for (Object attrValue : tree.getAttributeValues()) {

Object child = tree.getChild(attrValue);

outputDecisionTree(child, level + 1, attrName + " = "

+ attrValue);

}

} else {

System.out.printf("[CATEGORY = %s]\n", obj);

}

}

/\*\*

\* 样本，包含多个属性和一个指明样本所属分类的分类值

\*/

static class Sample {

private Map<String, Object> attributes = new HashMap<String, Object>();

private Object category;

public Object getAttribute(String name) {

return attributes.get(name);

}

public void setAttribute(String name, Object value) {

attributes.put(name, value);

}

public Object getCategory() {

return category;

}

public void setCategory(Object category) {

this.category = category;

}

public String toString() {

return attributes.toString();

}

}

/\*\*

\* 决策树（非叶结点），决策树中的每个非叶结点都引导了一棵决策树

\* 每个非叶结点包含一个分支属性和多个分支，分支属性的每个值对应一个分支，该分支引导了一棵子决策树

\*/

static class Tree {

private String attribute;

private Map<Object, Object> children = new HashMap<Object, Object>();

public Tree(String attribute) {

this.attribute = attribute;

}

public String getAttribute() {

return attribute;

}

public Object getChild(Object attrValue) {

return children.get(attrValue);

}

public void setChild(Object attrValue, Object child) {

children.put(attrValue, child);

}

public Set<Object> getAttributeValues() {

return children.keySet();

}

}

}

结果：

