SJTU-SE

测试计划

第六组

李　珊 516030910175

王梦瑶 516030910177

陈　诺 516030910199

胡雨奇 516030910257

目录

1. 测试目的

1. 联系和掌握白盒测试的一般过程与步骤。
2. 掌握使用 Intellij 和 Junit 进行测试的方法。
3. 通过测试检验源代码的可靠性。

2. 测试方法

利用 Intellij 和 Junit 来进行白盒测试。

通过针对源代码编写测试样例，使得代码覆盖率高于 95%，并且尽量覆盖判断条件。

3. 测试环境

系统：Window10 64 位

IDE：Intell IDEA

库：Maven、Junit

4. 测试代码简述

我们选用的代码是决策树。

通过 `outlook`, `temperature`, `humidity`, `windy` 来决定是否 `play`。

```
@relation weather.symbolic
@attribute outlook {sunny,overcast,rainy}
@attribute temperature {hot,mild,cool}
@attribute humidity {high,normal}
@attribute windy {TRUE,FALSE}
@attribute play {yes,no}
```

源代码由三份文件组成：

```
DecisionTree.java
InfoGain.java
TreeNode.java
```

其中 `DecisionTree.java` 是顶层文件，`InfoGain.java` 和 `TreeNode.java` 作为底层类被 `DecisionTree.java` 调用。

4.2.2 DecisionTree.java

DecisionTree.java 共有 13 个方法

```
// 设置InfoGain
public void setInfoGain(InfoGain in);
// 训练函数
public void train(String data_path, String targetAttr);
// 构建决策树
public TreeNode buildDT(String fatherName, String fatherValue, ArrayList<Integer>
subset,LinkedList<Integer> selatt);
// 剪枝函数
public ArrayList<int[]> cutBranch(TreeNode node);
// 获得叶子结点的数目
public int[] get_leafNum(TreeNode node);
// 读取arff文件，给attribute、attributevalue、data赋值
public void read_trainARFF(File file);
// 打印Data
public void printData();
// 将决策树存储到xml文件中
public void write_DecisionTree(String filename);
// 写TreeNode节点
private void write_Node(BufferedOutputStream bos, TreeNode node, String block);
// 设置决策变量
public void setDec(int n);
public void setDec(String targetAttr);
// main()
public static void main(String[] args);
// 获得根节点
public TreeNode getRoot();
```

1. main()

DecisionTree.java 的顶层函数是 main()，其大致架构如下：

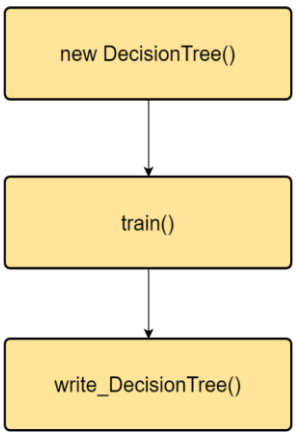

图 1. main() 架构

其中:

1. `write_DecisionTree()` 为将训练结果持久化进文件的函数，不涉及源代码内其他函数的调用。这里不用图像赘述。
2. `train()`为训练代码。

2. train()

Train()为训练代码。train()的大致架构如下：

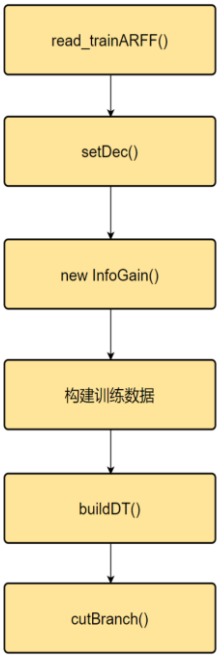

图 2. Train()架构

其中：

1. read_trainRAFF() 为读取 arff 文件，给 attribute、attributevalue、data 赋值， 此处同样不再赘述。
2. setDec() 为设置决策变量的函数，此处也不再赘述。
3. new InfoGain() 是为当前决策树新建了一个 InfoGain 类，这部分将在后面提到。
4. 构建训练数据部分为通过 read_trainRAFF 中读到的文件来构建已知的 attributevalue 和 data 等，同样没有更深入的函数调用。
5. buildDT()用来构建决策树。
6. curBranch() 是剪枝函数。

4.2.3 InfoGain.java

该类共有 8 个方法：

```java
// 构造函数
public InfoGain(ArrayList<String[]> trainData, int decatt);
// 计算信息熵
public double getEntropy(Map<String, Integer> attributeNum);
public double getEntropy(ArrayList<Integer> subset, int attributeIndex);
// 信息熵增益率相关
public int getGainRatioMax(ArrayList<Integer> subset, LinkedList<Integer> selatt);
// 判断分类是否唯一
public boolean isPure(Map<String,Integer> targetNum);
// 获得对应数据子集的对应特征的值-频率字典
public  Map<String,Integer> get_AttributeNum(ArrayList<Integer> subset, int attributeIndex );
// 获得数据在某一特征维度下的子集划分
public Map<String, ArrayList<Integer>> get_AttributeSubset(ArrayList<Integer> subset, int attributeIndex);
// 根据类-数目，判读分类结果
public String get_targetValue(Map<String,Integer> targetNum);
```

以下选取 `getGainRatioMax()`来介绍白盒测试中的覆盖分类。

该函数的源代码如下：

```java
//信息熵增益率相关
public int getGainRatioMax(ArrayList<Integer> subset, LinkedList<Integer> selatt){
    //计算原信息熵

    Map<String, Integer> old_TargetNum = get_AttributeNum(subset, decatt);
    double oldEntropy = getEntropy(old_TargetNum);
    double maxGainRatio=0;
    int maxIndex=decatt;

    for(int attributeIndex: selatt){
        Map<String, ArrayList<Integer>> attributeSubset = get_AttributeSubset(subset, attributeIndex);

        int sum = 0;
        double newEntropy = 0;
        for(ArrayList<Integer> tempSubset: attributeSubset.values()){
            int num = tempSubset.size();
            sum += num;
            double tempEntropy = getEntropy(tempSubset,decatt);
            newEntropy += num * tempEntropy;
        }
        newEntropy /= sum;
        double tempGainRatio = (oldEntropy - newEntropy)/getEntropy(subset, attributeIndex);  //计算信息增益率

        //如果信息增益率为负，应该停止分支，此处避免麻烦没有做进一步讨论。
        if(tempGainRatio > maxGainRatio){
            maxGainRatio = tempGainRatio;
            maxIndex = attributeIndex;
        }
    }
    return  maxIndex;
}
```

这段代码的程序流程图如图所示

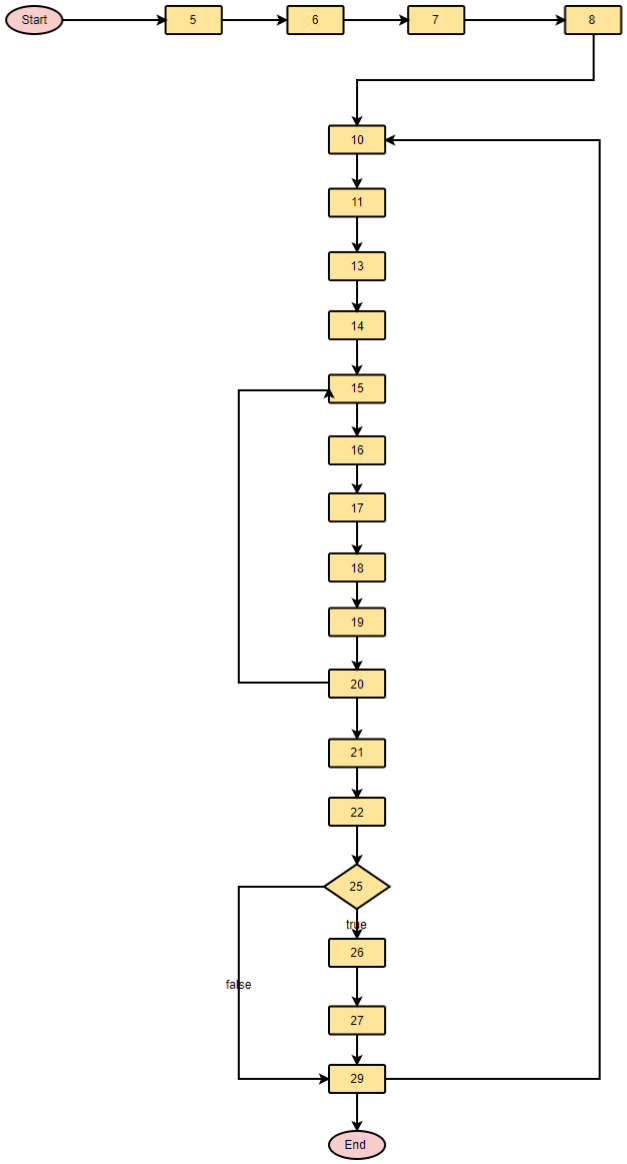

图 3. getRatioMax()的程序图

对于测试的代码来说，语句覆盖率必须要大于 95%，然后依据设计的测试用例严格程度可以分为 6 种：

1. 语句覆盖测试用例设计：

   设计合适的测试用例使得所有的语句都被覆盖。

2. 判定覆盖测试用例设计：

设计合适的输入使得判断语句（本函数之中只有 line 25 是判断语句）的每个取值分支都至少经历一次。 也即使得 `tempGainRatio > maxGainRatio` 和 `tempGainRatio <= maxGainRatio` 都至少经历一次。

a) 优点：较语句覆盖可以有更强的覆盖，简单易理解，只需要关注单个判定，无需细分即可得到测试用例。

b) 缺点：大部分的判定条件都是多个组合起来的（or/and/case），若仅判断整个条件的结果，会遗漏部分数据类型测试路径。

3. 路径覆盖测试用例设计：

保证每条可能执行到的路径至少执行一次。

4. 条件覆盖测试用例设计：

选择足够的测试用例，使得运行这些测试用例时，判定中每个条件的所有可能结果至少出现一次，但未必能覆盖全部分支。

条件覆盖要检查每个符合谓词的子表达式值为真和假两种情况，要独立衡量每个子表达式的结果，以确保每个子表达式的值为真和假两种情况都被测试到。

5. 判定条件覆盖测试用例设计：

判定-条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有可能判断结果至少执行，即要求各个判断的所有可能的条件取值组合至少执行一次。

6. 条件组合覆盖设计用例设计：

使所有判定中各条件判断结果的所有组合至少出现一次，满足这种覆盖标准成为条件组合覆盖。这是算是覆盖最全的了。

4.2.3 TreeNode.java

这个类相对简单，函数大多数只是在设置、获得类的属性。

```
public TreeNode();
public String getNodeType() ;
public void setNodeType(String nodeType);
public String getAttributeName() ;
public void setAttributeName(String attributeName);
public String getAttributeValue() ;
public void setAttributeValue(String attributeValue);
public ArrayList<TreeNode> getChildTreeNode() ;
public void setChildTreeNode(ArrayList<TreeNode> childTreeNode) ;
public TreeNode getFatherTreeNode() ;
public void setFatherTreeNode(TreeNode fatherTreeNode);
public Map<String, Integer> getTargetNum();
public void setTargetNum(Map<String, Integer> targetNum);
public String getTargetValue();
public void setTargetValue(String targetValue);
@Override
public String toString();
```

这里不做过多解释。