SJTU-SE

测试报告

第 6 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. 测试前准备

4.1 开始之前对 Intellij 和 Junit 的相关设置

在 Edit Run Configuration 中新建一个 Junit configuration。

设置 Junit 测试类的运行 configuration，将 Code Coverage 设为 Tracing，并且勾选 Track per test coverage.

在 Package 中勾选需要测试覆盖率的类。

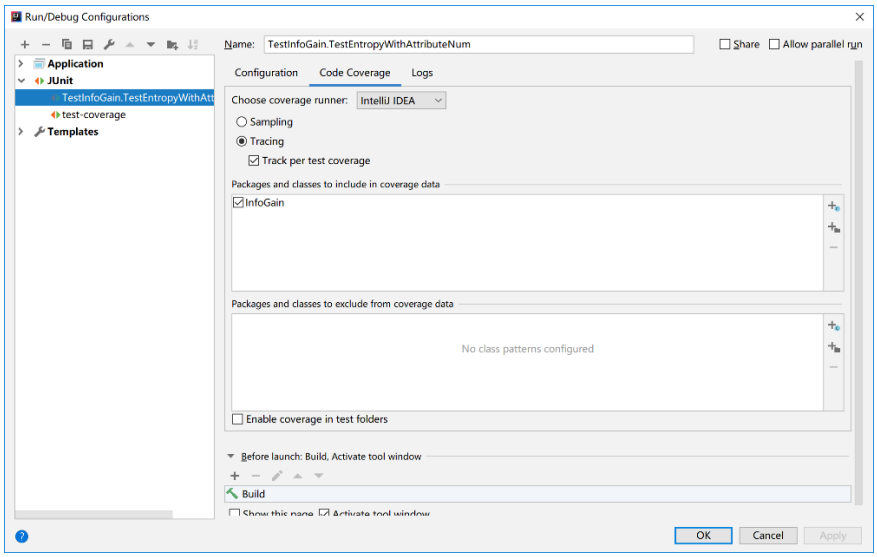

图 1. 新建 Run Configuration

在运行时选择 `Run with coverage` 就可以查看测试用例的覆盖率。

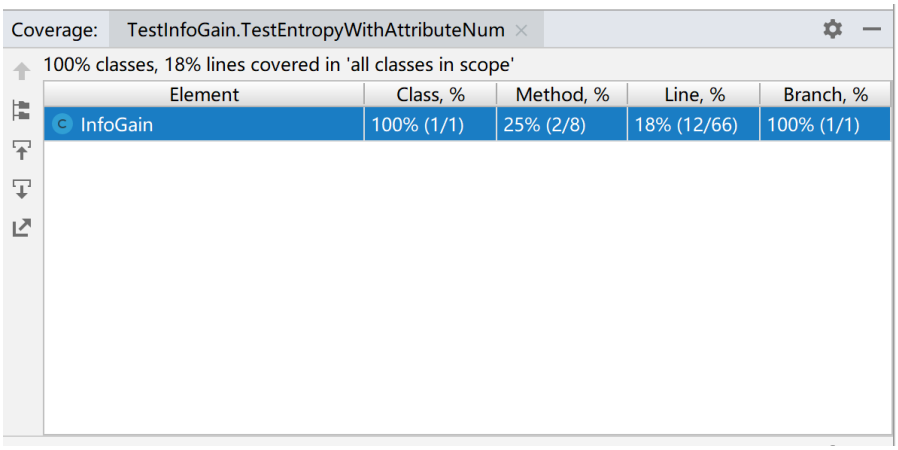

图 2. 查看 coverage

6. 测试内容

通过阅读代码，来为函数添加测试代码。通过设计测试样例使得测试覆盖率尽可能高。

6.1 测试结果

最终的覆盖率：

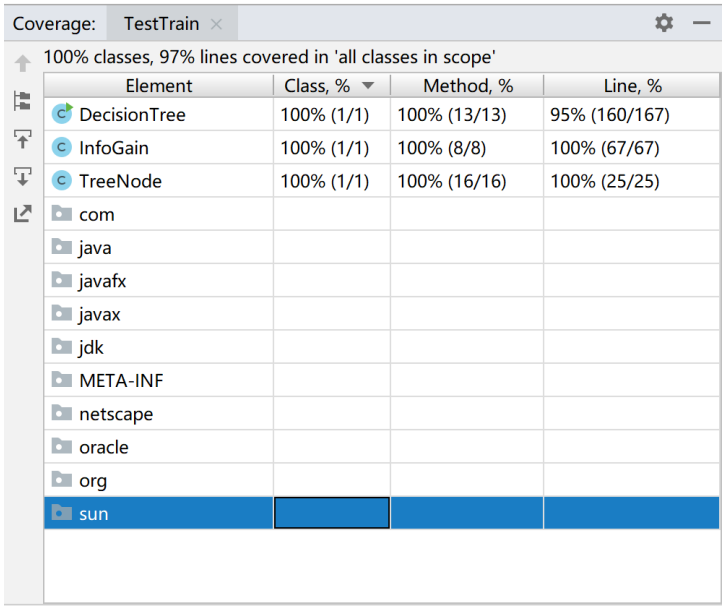

图 3. 总覆盖率

6.2 DecisionTree

源代码：

```
./src/main/java/Decision.java
```

测试代码：

```
./src/test/java/TestDecisionTree.java
./src/test/java/TestTrain.java
```

对源代码中的函数的分析如下：

6.2.1 printData

编码人员用于调试输出的辅助方法，逻辑是遍历打印 train data，没有分支，采用初始化后直接调用的方式测试正确性。

6.2.2 read_trainARFF

读取 arff 后缀的数据文件，并解析成为 DecisionTree 数据结构。

测试包含两个用例，

1. 正常的输入数据文件名。

   不抛出异常，同时完成数据结构的初始化

2. 不存在的文件

   抛出 FileNotExist 异常

6.2.3 write_DecisionTree

```
public void write_DecisionTree(String filename) {
    try {
        File file = new File(filename);
        if (!file.exists())
            file.createNewFile();
        FileOutputStream fs = new FileOutputStream(filename);
        BufferedOutputStream bos = new BufferedOutputStream(fs);
        write_Node(bos, root, "");
        bos.flush();
        bos.close();
        fs.close();
    }catch (IOException e){
        e.printStackTrace();
    }
}
```

对应的函数图为

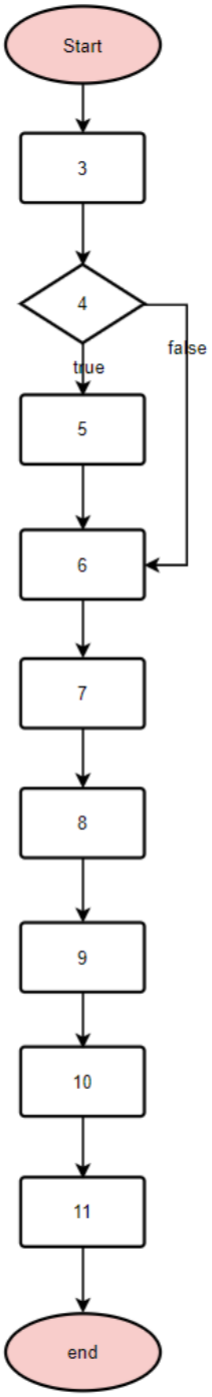

图 4. write_DecisionTree 函数图

如上图所示，遍历所有分支需要使 4 节点为 true 和 false 各一次，设计两个测试用例

1. output file 文件存在的情况

   应当覆盖文件内容

2. output file 文件不存在的情况

   应当创建文件后，写入文件内容

6.2.4 write_Node

将节点数据按序保存到 xml 文件中，通过输入预先准备的 arff 数据并核对输出文件内容测试。

6.2.5 setDec

设置成员变量的值，通过调用一次测试。程序不抛出权限异常则说明该方法正确。

6.2.6 get_leafNum

获得叶子结点里的数目，返回一个二元数组，第一个元素是最小值，第二个元素是总和。如果结点只有一个元素，最小值设为 0，最大值是总和，如果没有元素，最小值设置为最大的 int 值。

通过测试 TreeNode 的 targetNum 的 size 为 0,1 和大于 1 的三种情况，都通过则说明该方法正确。

设计三个测试用例：

1. size = 0 返回{Integer.MAX_VALUE,0}
2. size = 1 返回{0,sum}
3. size > 2 返回{min,sum}

`cutBranch`

有三种情况：

1. 是本身是叶子结点，则不需要剪枝，直接返回
2. 非叶子结点，符合剪枝条件，剪枝
3. 非叶子结点，不符合剪枝条件，不用剪枝

设计三个测试用例：

1. 本身是叶子结点：

node 和剪枝之前不变，返回 size 为 1 的{min,sum}数组 ArrayList

2. 非叶子结点，符合剪枝条件

   node 把子节点的数组的元素清零，属性设为 leafNode,返回 size 为 1 的{min,sum}数组 ArrayList

3. 非叶子结点，不符合剪枝条件

   node 和剪枝之前不变，返回 size 为叶子节点个数的{min,sum}数组 ArrayList

如果三个用例的返回结果和 node 的属性都符合期望值，则说明测试通过。

6.2.7 buildDT

构建决策树，设置节点名称和节点值，数据行子集，数据列子集，再根据 infogain 的信息，构建决策树，返回根节点。

6.2.8 train

子方法的正确性在上述测试中已经覆盖，train 函数的逻辑是按序调用上述方法实现程序逻辑。

测试由已有的数据构建得到正确的决策树，如果得到的决策树和期望的树结构一致，各节点的各个属性的值都一致，则说明测试通过。

6.2.9 main

子方法的正确性在上述测试中已经覆盖，main 函数的逻辑是按序调用上述方法实现程序逻辑。

在测试中使用预先准备的 arff 数据调用 main 函数，能正常运行到程序结束并输出结果则说明测试通过。

6.3 InfoGain

源代码:
`./src/main/java/InfoGain.java`
测试代码:
`./src/test/java/TestInfoGain.java`

6.3.1 getEntropy

这个函数用于计算信息熵，由有无 subset 分为两个函数。

针对两个函数分别编写测试样例，使得测试代码能够覆盖到两个函数的所有行。

没有 subset 的函数的主要逻辑为：

遍历 map 好的 attribute，计算他们的熵总和，并得到最终的信息熵，函数中包含一次循环，没有条件判断语句。

在测试代码中初始化一个 InfoGain 对象后，初始化含有两个键值对的 map 作为 atrributes 传入函数中进行测试，期望的信息熵为 1，使用 assertEquals 来判断输出是否正确。

对于有 subset 的函数，其主要逻辑为：

首先通过 get_AttributeNum()函数来获得映射关系，然后在调用上一个函数来计算信息熵。

在测试代码中初始化一个 InfoGain 对象后，构造对应的 subset，使得期望的信息熵为 0, 1, 1.5，并分别使用 assertEquals 来判断输出是否正确。

两个测试均通过。

6.3.2 getRatioMax

这个函数用于计算信息熵增益率，当信息熵增益率为负，就可以停止该分支以减少无谓的计算量。

同样，在测试函数中新建一个 InfoGain，并初始化属性。通过构造输入函数的参数来获得不同的期望输出。

测试代码中，通过初始化了两个不同的 InfoGain 并赋予不同的初始化属性，来使得对于同样的信息输入产生不同的信息熵增益率。使用 assertEquals 来判断输出是否符合期望。

测试通过。

6.3.3 isPure

这个函数用来判断分类是否唯一，函数本身比较短。

通过构造关系映射并传入函数即可检验。当关系映射中只有一对键值对的时候，分类是唯一的，函数应该返回 true，当有多个键值对的时候，分类不唯一，应该返回 false。

使用 assertTrue 和 assertFalse 来判断输出是否符合期望。

测试通过。

6.3.4 get_AttributeNum

这个函数用来获得对应数据子集的对应特征值的值-频字典。

函数的逻辑为遍历传入的数据子集，并为每个数据构造值-频键值对，存入 map 中。

通过构造不同的数据子集，并分别使用这些子集来调用本函数，测试获得的值-频字典即可判断函数正确性。

测试中构造了 3 个不同的数据子集，并分别调用本函数，同时构造了三个期望输出的 map 对象来检查实际输出。

使用 assertEquals 来判断，测试通过。

6.3.5 get_targetValue

这个函数用来根据类-数目，判读分类结果。

函数遍历了传入的 map 对象，得到了其中数目最大的类的名字。

在测试函数中，构造了三个不同的 map 对象传入函数中，使用 assertEquals 来判断输出是否符合期望。

测试通过。

6.4 TreeNode

源代码:
`./src/main/java/TreeNode.java`
测试代码:
`./src/test/java/TestTreeNode.java`

该部分大多数函数均只有一行（设置属性或读取属性），这里不再为所有函数一一说明逻辑。

测试代码大多数都是构造 TreeNode 并设置相应的属性，然后进行测试。

测试通过。

7. 实验结果

通过对决策树的白盒测试，我们认为程序能够基本满足需求。

通过白盒测试，我们对代码的逻辑流程有了更加透彻深入的理解，同时也掌握了使用 Intellij 和 Junit 进行白盒测试的基础方法。

8. 参考

8.1 决策树算法

https://www.cnblogs.com/hermione1985/p/6750209.html
https://blog.csdn.net/jiaoyangwm/article/details/79525237

8.2 白盒测试报告

https://wenku.baidu.com/view/4c8c7573f242336c1eb95ea1.html
https://online.visual-paradigm.com/cn/