|  |
| --- |
| SJTU-SE |
| 测试计划 |
| 第六组 |

|  |
| --- |
| 陈诺李 珊 516030910175  王梦瑶 516030910177  陈 诺 516030910199  胡雨奇 516030910257 |

目录

[1. 测试目的 2](#_Toc8596589)

[2. 测试方法 2](#_Toc8596590)

[3. 测试环境 2](#_Toc8596591)

[4. 测试代码简述 3](#_Toc8596592)

[4.2.2 DecisionTree.java 3](#_Toc8596593)

[4.2.3 InfoGain.java 5](#_Toc8596594)

[4.2.3 TreeNode.java 10](#_Toc8596595)

# 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

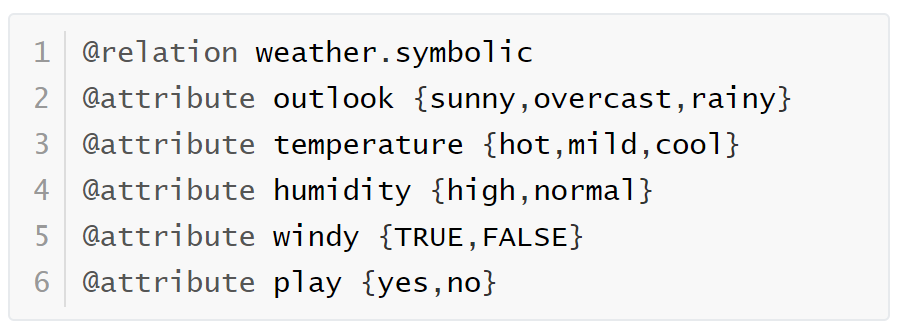
IDE：Intell IDEA

库：Maven、Junit

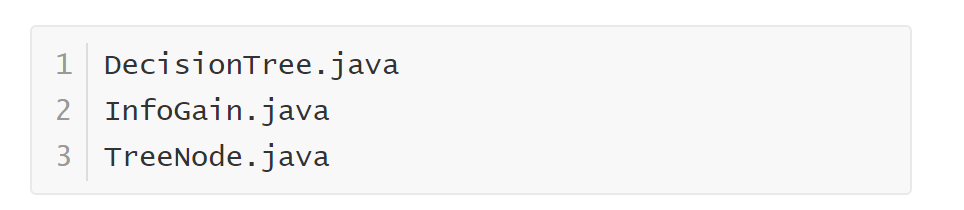
# 4. 测试代码简述

我们选用的代码是决策树。

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



源代码由三份文件组成：



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

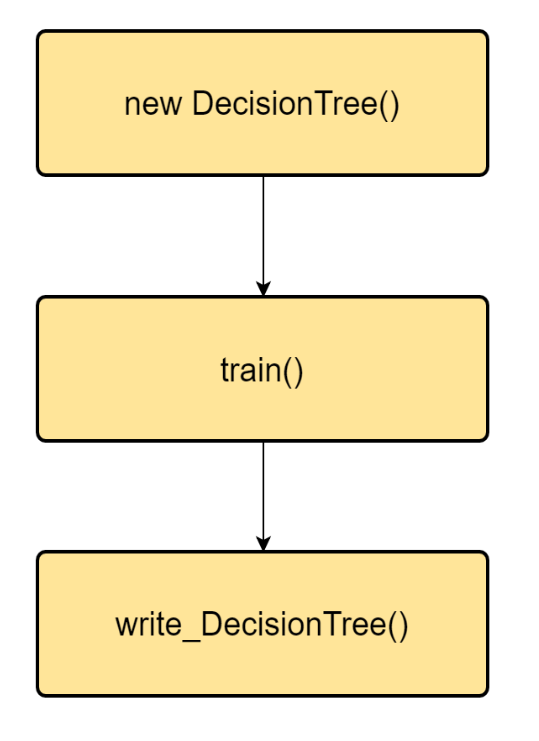
### 4.2.2 DecisionTree.java

DecisionTree.java共有13个方法



#### 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：

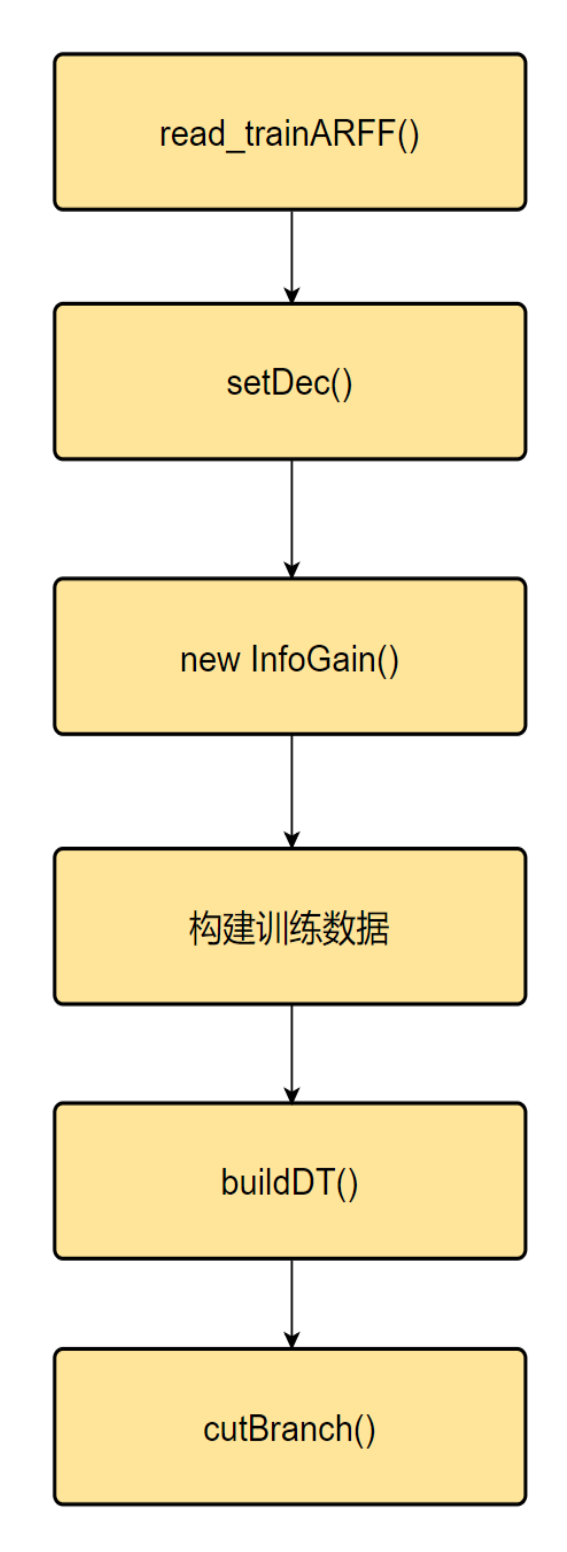


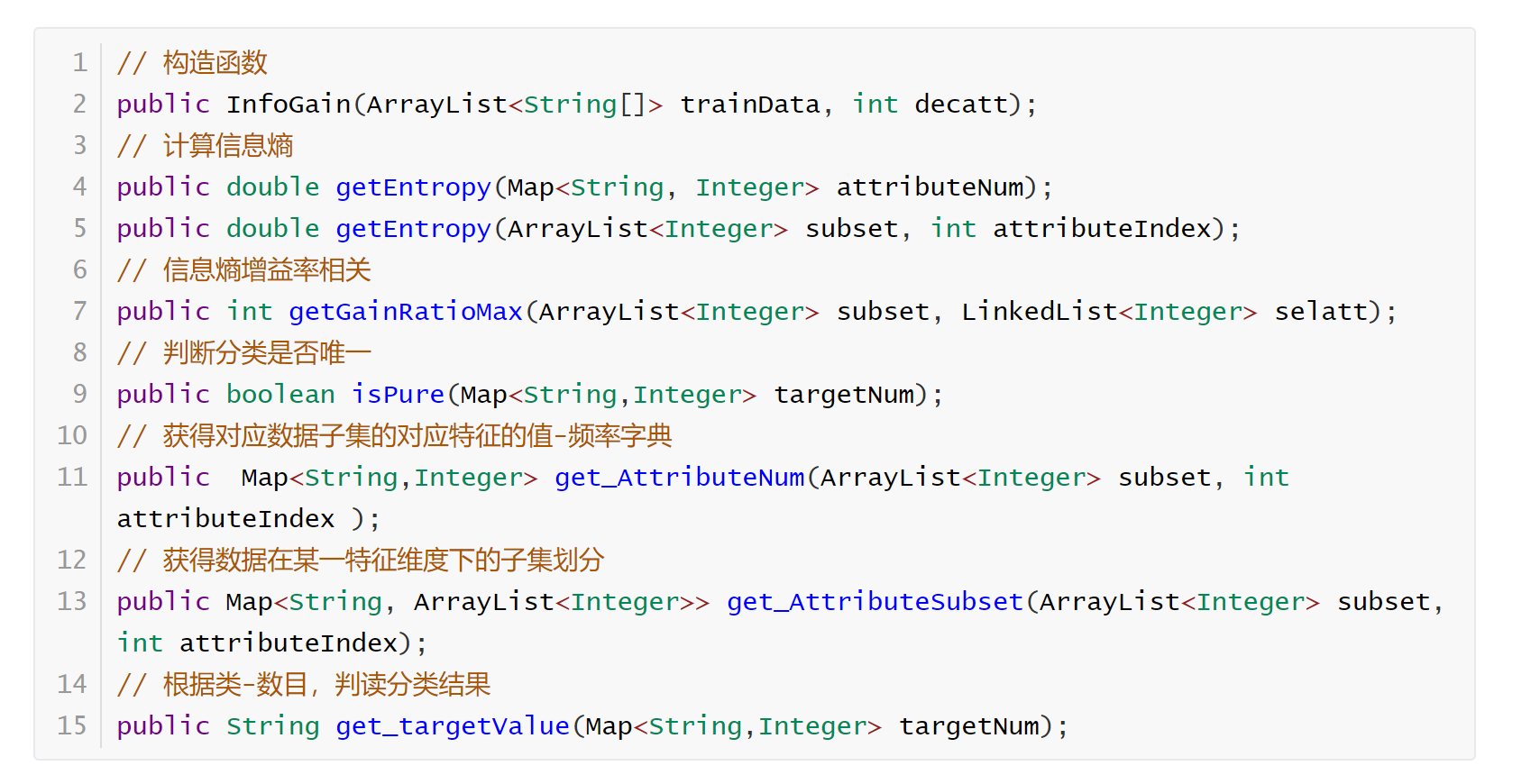
图2. Train()架构

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：



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：



这段代码的程序流程图如图所示

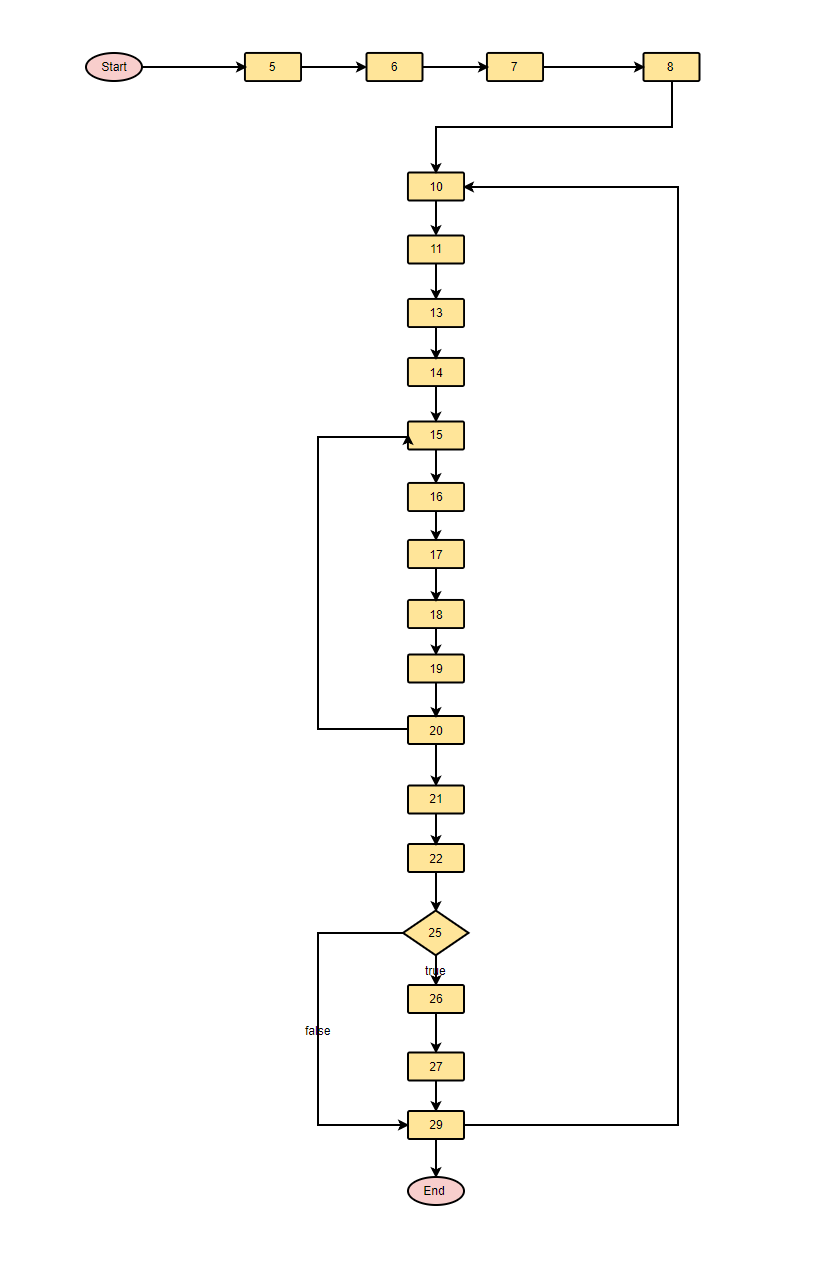


图3. getRatioMax()的程序图

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。

1. 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。

* 1. 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。
  2. 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。

1. 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。

1. 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。

1. 判定条件覆盖测试用例设计：

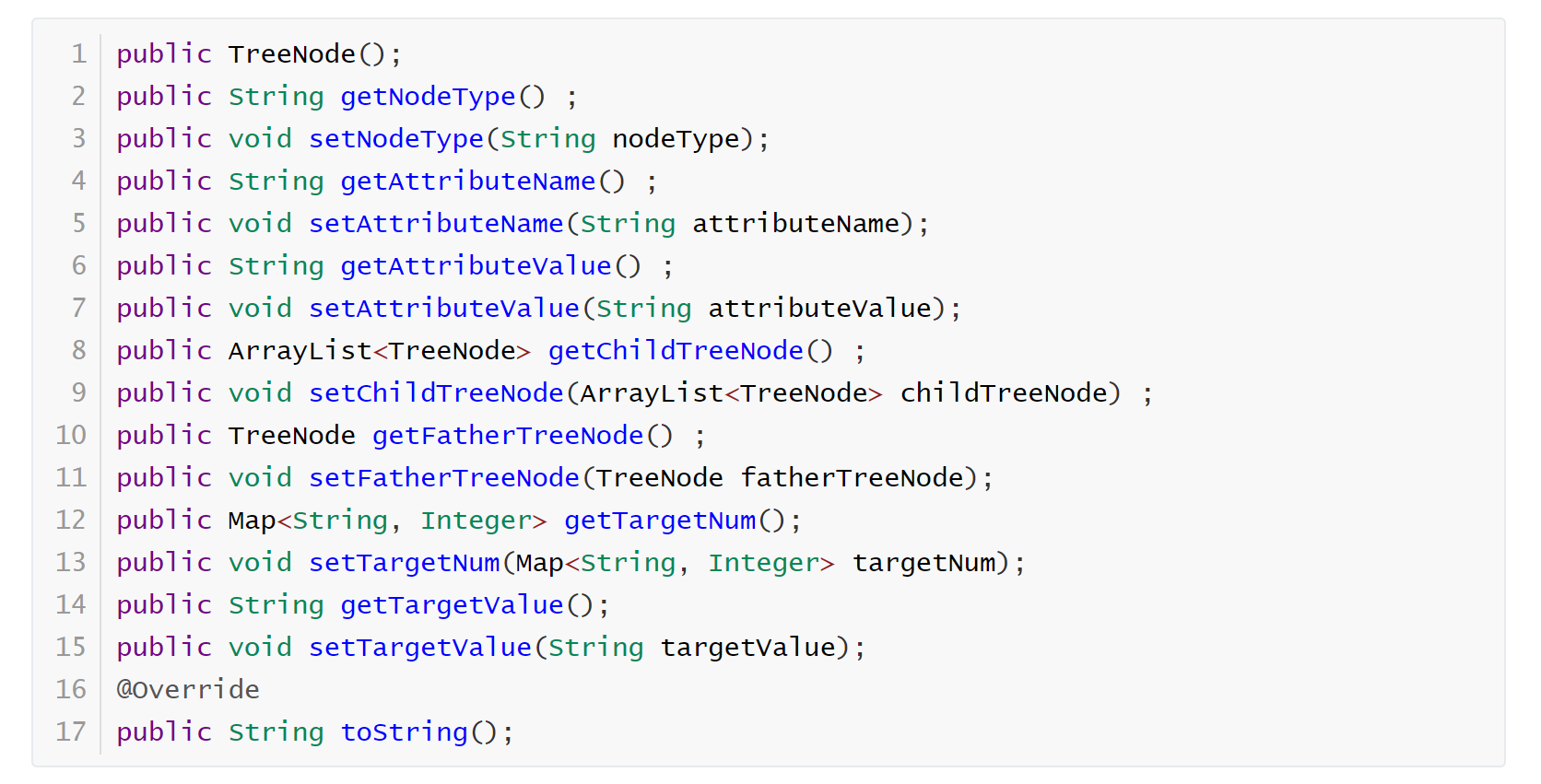
判定-条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有可能判断结果至少执行，即要求各个判断的所有可能的条件取值组合至少执行一次。

1. 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。



这里不做过多解释。