中国地质大学(武汉)

机器学习第二次作业

姓 名：叶宇涛

专 业：计算机科学与技术

学 号：20191000595

指导老师：刘超

目录

[4 第四章作业 3](#_Toc99732203)

[4.1 任意选择4个UCI数据集，对基于信息增益划分选择(ID3)、基于基尼指数划分选择(CART)，基于对率回归划分选择的决策树算法(包括未剪枝、预剪枝、后剪枝三种)进行实验比较。 3](#_Toc99732204)

[4.1.1 编程题目理解 3](#_Toc99732205)

[4.1.2 决策树算法原理阐述 4](#_Toc99732206)

[4.1.3 算法设计思路 6](#_Toc99732207)

[4.1.4 实验流程、测试结果及分析 8](#_Toc99732208)

[4.1.5 代码结构，核心代码简要分析 17](#_Toc99732209)

[4.1.6 本次实验解决的主要问题，主要收获 19](#_Toc99732210)

[4.1.7 编码及内容撰写中的参考来源 19](#_Toc99732211)

[4.2 选择表4.2的西瓜数据集，采用”队列”数据结构，编程实现基于信息增益划分选择的非递归的深度优先搜索的未剪枝决策树算法，计算验证集精度。 20](#_Toc99732212)

[4.2.1 编程题目理解 20](#_Toc99732213)

[4.2.2 栈结构非递归建树算法原理阐述 20](#_Toc99732214)

[4.2.3 算法设计思路 21](#_Toc99732215)

[4.2.4 实验流程、测试结果及分析 23](#_Toc99732216)

[4.2.5 代码结构，核心代码简要分析 23](#_Toc99732217)

[4.2.6 本次实验解决的主要问题，主要收获 23](#_Toc99732218)

[4.2.7 编码及内容撰写中的参考来源 23](#_Toc99732219)

[4.3 选择表4.2的西瓜数据集，改写为基于信息增益划分选择的非递归的广度优先搜索的未剪枝决策树算法，计算验证集的精度。 23](#_Toc99732220)

[4.3.1 编程题目理解 23](#_Toc99732221)

[4.3.2 决策树算法原理阐述 24](#_Toc99732222)

[4.3.3 算法设计思路 24](#_Toc99732223)

[4.3.4 实验流程、测试结果及分析 24](#_Toc99732224)

[4.3.5 代码结构，核心代码简要分析 24](#_Toc99732225)

[4.3.6 本次实验解决的主要问题，主要收获 24](#_Toc99732226)

[4.3.7 编码及内容撰写中的参考来源 24](#_Toc99732227)

# 第四章作业

## 任意选择4个UCI数据集，对基于信息增益划分选择(ID3)、基于基尼指数划分选择(CART)，基于对率回归划分选择的决策树算法(包括未剪枝、预剪枝、后剪枝三种)进行实验比较。

### 编程题目理解

题目要求选择4个UCI数据集，分别进行基于信息增益、基尼指数、对率回归的决策树算法的实现，并且还要实现未剪枝、预剪枝、后剪枝三种不同的算法。对于不同的决策树，改变的是划分的依据，也就是信息增益、基尼指数这一类指标的不同。在这里选择了以下数据集：Iris、wine、breast\_cancer、diabetes。都是UCI中的分类数据集，并且没有少数据和异常数据的情况。其中，主要利用iris数据集进行测试。

### 决策树算法原理阐述

决策树是一种分类模型。其输入是带有标签的数据，输出是一颗决策树。其非叶节点代表的是逻辑判断；叶节点代表的是分类的子集。决策树算法原理是通过训练数据形成if-then的判断结构。从树的根节点到叶节点的每一条路径构成一个判断规则。我们需要选择合适的特征作为判断节点，可以快速的分类，减少决策树的深度。最理想的情况是，通过特征的选择把不同类别的数据集贴上对应类标签，树的叶子节点代表一个集合，集合中数据类别差异越小，其数据纯度越高。

**决策树划分伪代码如下：**

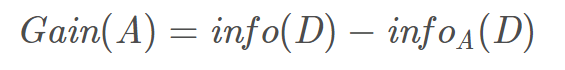


图表 4.1‑1 决策树算法伪代码

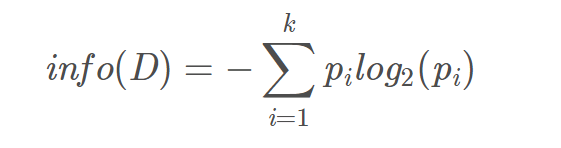
其中，根据数据特征函数的不同，可以划分不同的算法。比较经典的算法有ID3、C4.5、Cart算法。

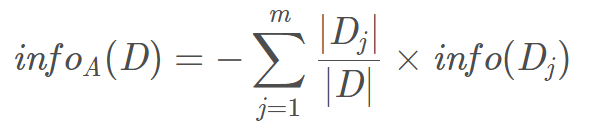
**ID3：**

ID3使用的数据特征函数是信息增益，给出公式如下：



其中，





计算所有类别的信息增益，选择信息增益最大的作为分类节点，这就是ID3算法的划分依据。

**CART：**

CART算法用基尼指数作为划分指标：

这代表每个类别中的两个样本不同的概率，也可以依据这个判断样本的纯度。

**基于对率回归：**

基于对率回归算法的指标就是利用对数几率回归的算法划分样本，对于一个树节点中的样本，根据全部特征可以进行对率转换为如下形式：

然后通过梯度下降算法求得对率回归的一组参数，通过这组参数，可以将该节点样本值分为两类。实现的决策树应如下图所示：



图表 4.1‑2 基于对率回归算法

**预剪枝：**

预剪枝需要预先设置一个验证集，在每次划分节点的时候，计算不划分时候的精确度，以及假如划分之后的精确度。如果不划分时候的精确度比划分的高，那么划分，否则，将当前节点设置为叶子节点，类别设置为当前数据集中样本数量最多的类别。

**后剪枝：**

对于后剪枝算法，同样需要设置验证集。在已经建好树的情况下，递归判断当前树节点的验证集精确度和划分之后的验证集精确度，来确定是否需要划分。如果不需要划分，将当前节点设置为叶子节点，类别设置为数据集中样本数量最多的类别。

### 算法设计思路

首先需要构建一棵决策树，因此，将按照书上面的流程进行建树。设置一个节点类，存放该节点的子树节点、特征类别等变量。再设置一个决策树类，初始化的时候初始化根节点、以及采用基尼指数或者信息增益的方法。并判断：

1. 如果输入的数据集中样本全部属于一类，判定为叶子节点，返回；
2. 如果输入的数据集中特征用完，判定为叶子节点返回；
3. 否则调用分割函数，找到最佳的特征名称和分割点；
4. 如果是离散值，根据最佳的特征名称进行划分，并且删除这个最佳的特征，对该特征下的特征值循环生成子树。
5. 如果是连续值，根据最佳的分割点划分D+和D-,对于D+部分递归生成子树，对于D-部分递归生成子树。
6. 打印该树。

**对于如何选择最佳信息增益，分别根据离散值以及连续值进行判断:**

1. 如果这个特征是离散值，那么，该特征求出的信息增益值，返回信息增益值。
2. 如果这个特征是连续值，那么，对该特征下的所有值求中点，求取完中点后，遍历每一个中点，划分为D+和D-，求取信息增益值。最后，选择最大的信息增益值，返回信息增益值和划分点。
3. 根据最大的信息增益值选择特征，如果是离散特征，需要删除这个特征，如果是连续特征，则不需要删除。

**预剪枝算法设计：**

1. 判断当前节点是否是叶子节点或者数据集为空，是则返回；
2. 计算当前节点中样本最多的类别，计算未划分时候的精确度；
3. 计算最佳的划分点，进行假设划分，计算划分后的验证集精确度，如果划分效果好于不划分，那么划分。
4. 设置子树节点，递归调用计算。

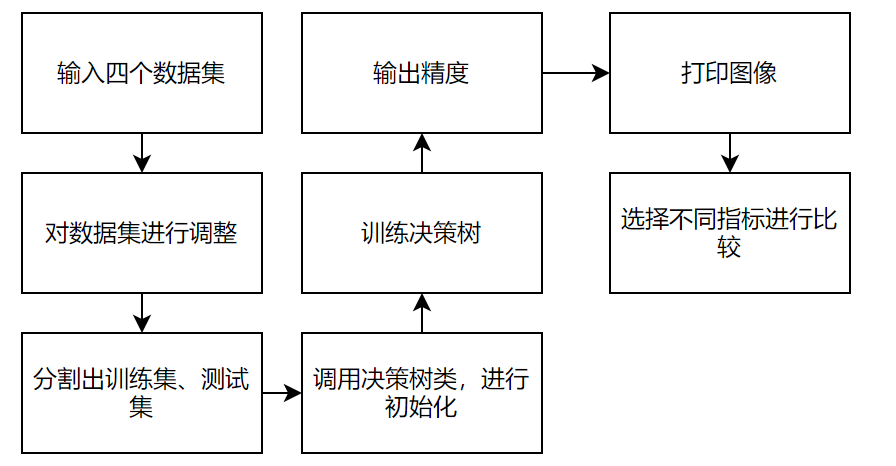
**后剪枝算法设计：**

1. 判断当前节点是否是叶子节点或者数据集为空，是则返回；
2. 计算当前节点中样本最多的类别，计算未划分时候的精确度；
3. 递归调用建树。
4. 如果递归到左右子树都为叶子节点，判断是否划分时候验证集的精确度好于不划分时候的精确度，是则设置当前节点为叶子节点，重新设置高度、类别设置为数据集中样本数量最多的类别，进行剪枝。
5. 返回。

**最后，利用matplotlib对该树进行打印。打印步骤如下：**

1. 根据叶子节点的个数，选择合适的x偏移量和y偏移量进行打印。
2. 输入节点，判定这个节点是不是叶子节点，如果是叶子节点，直接打印。
3. 如果不是叶子节点，递归调用该函数，增加x偏移量和y偏移量。并且打印线的名称。

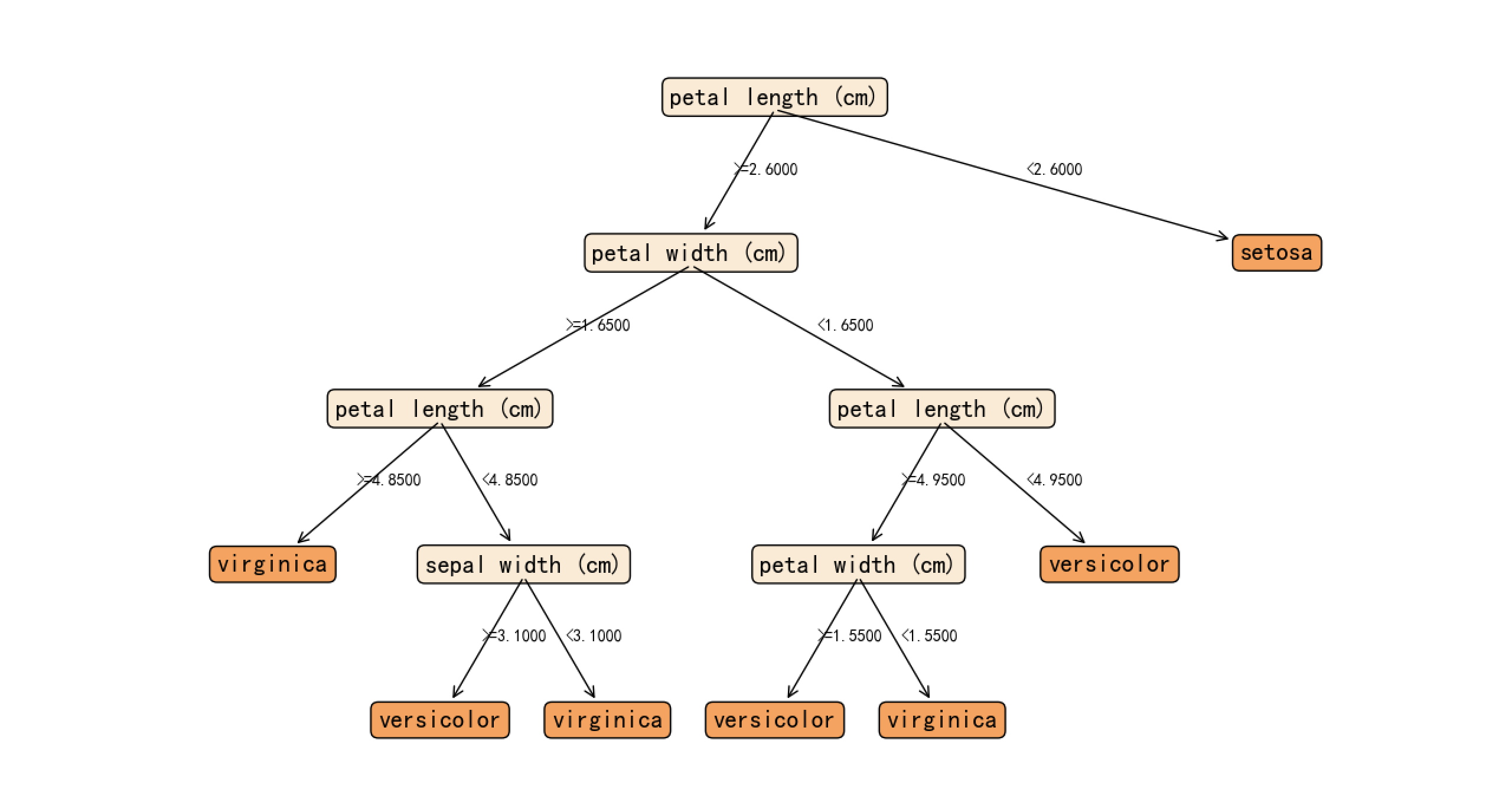
### 实验流程、测试结果及分析



图表 4.1‑3 决策树实验流程图

对于信息增益：

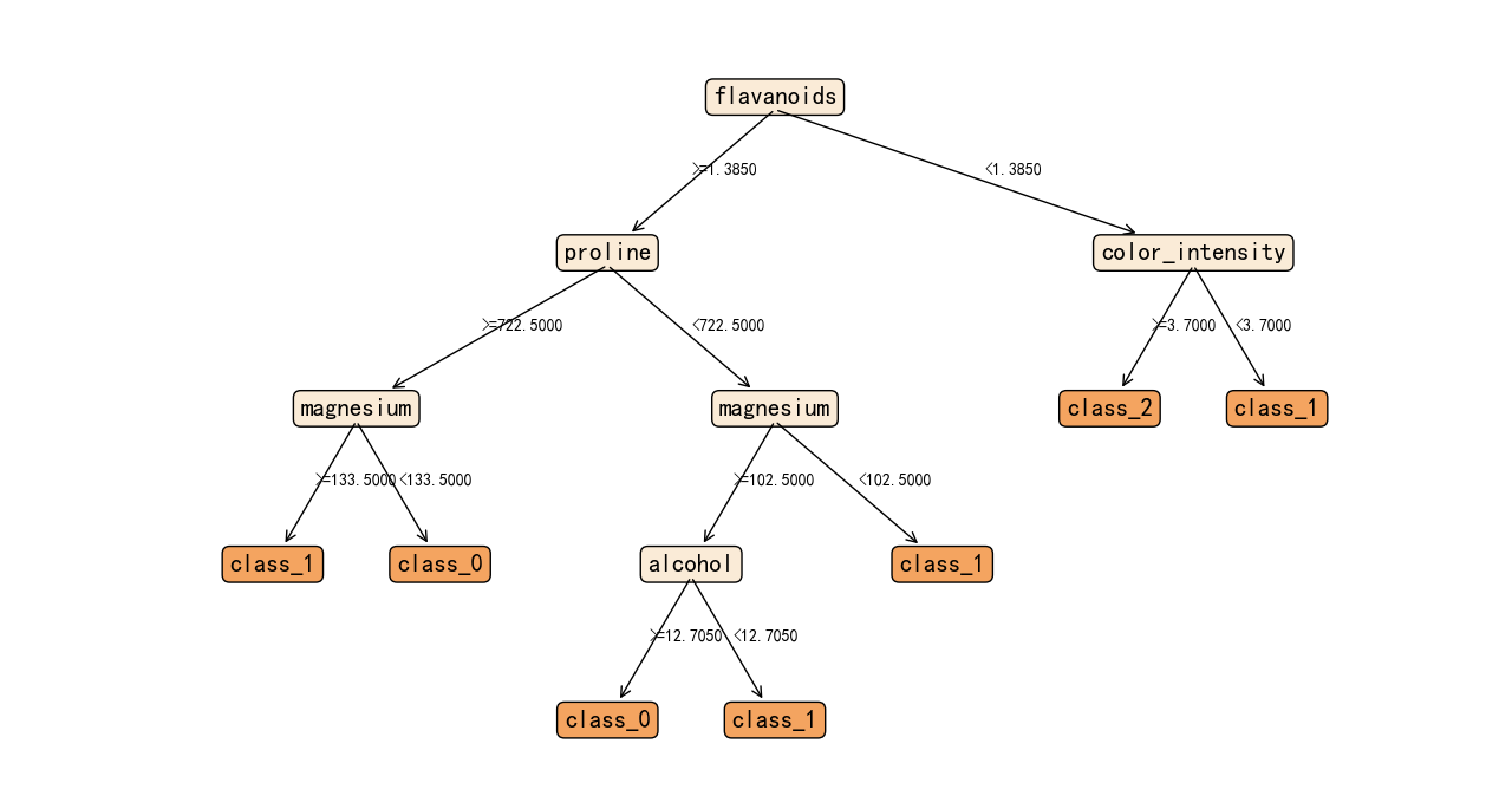
**未剪枝：**



图表 4.1‑4 信息增益IRIS数据集



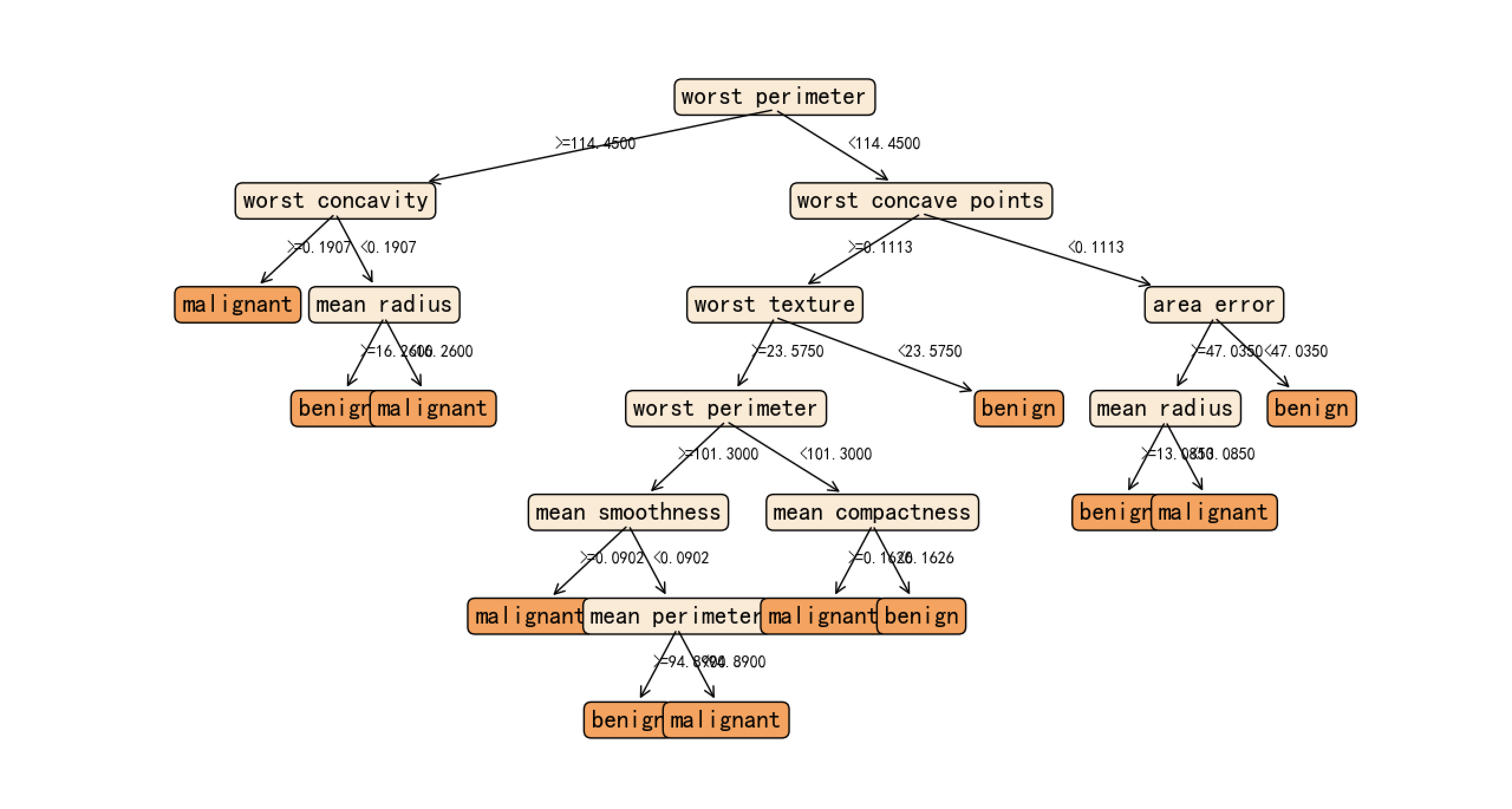
图表 4.1‑5 iris数据集正确率



图表 4.1‑6 wine数据集



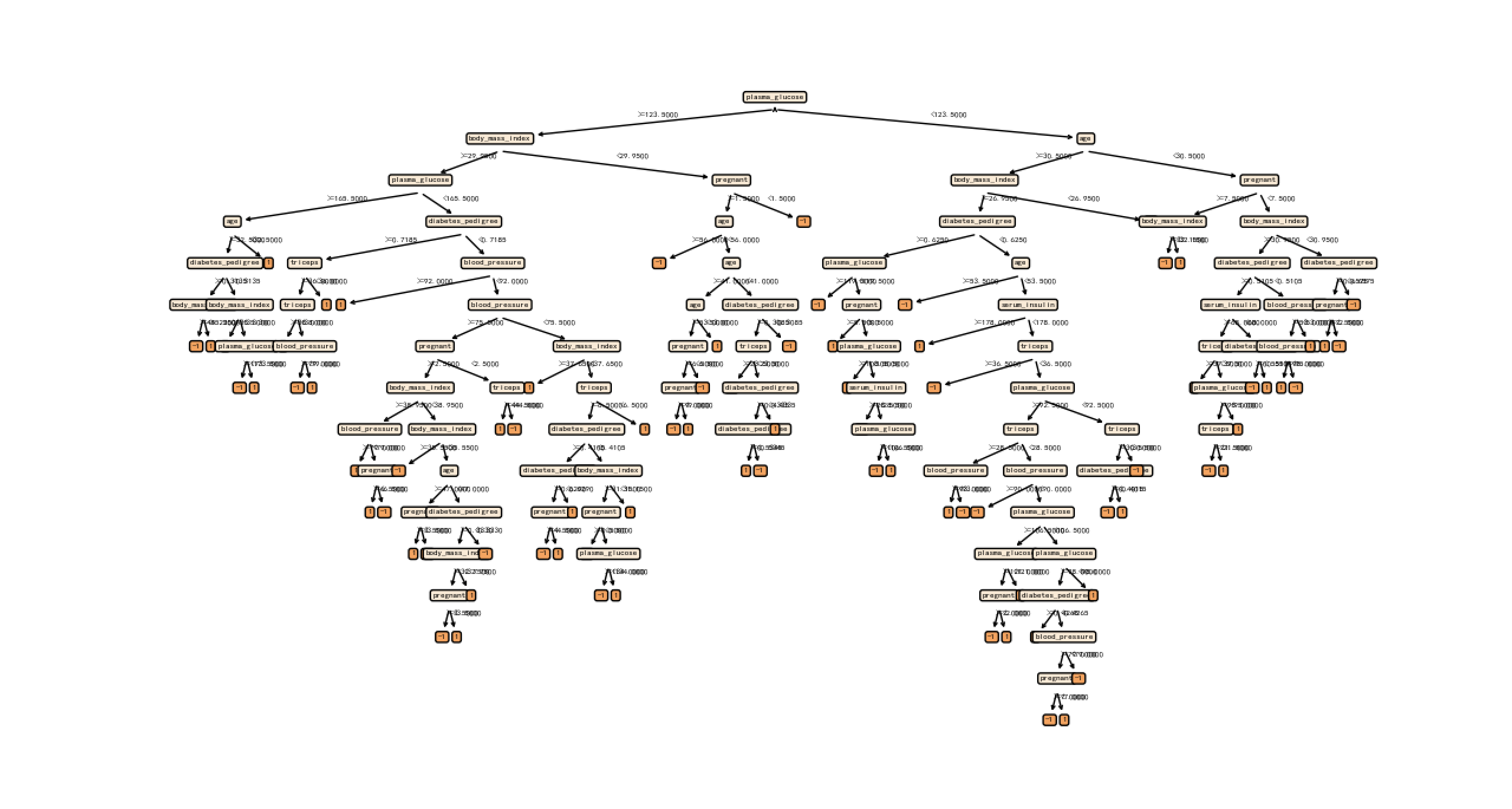
图表 4.1‑7 wine数据集正确率



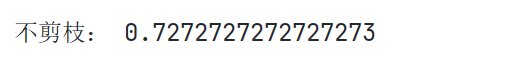
图表 4.1‑8 breast\_cancer数据集



图表 4.1‑9 breast\_cancer数据集正确率

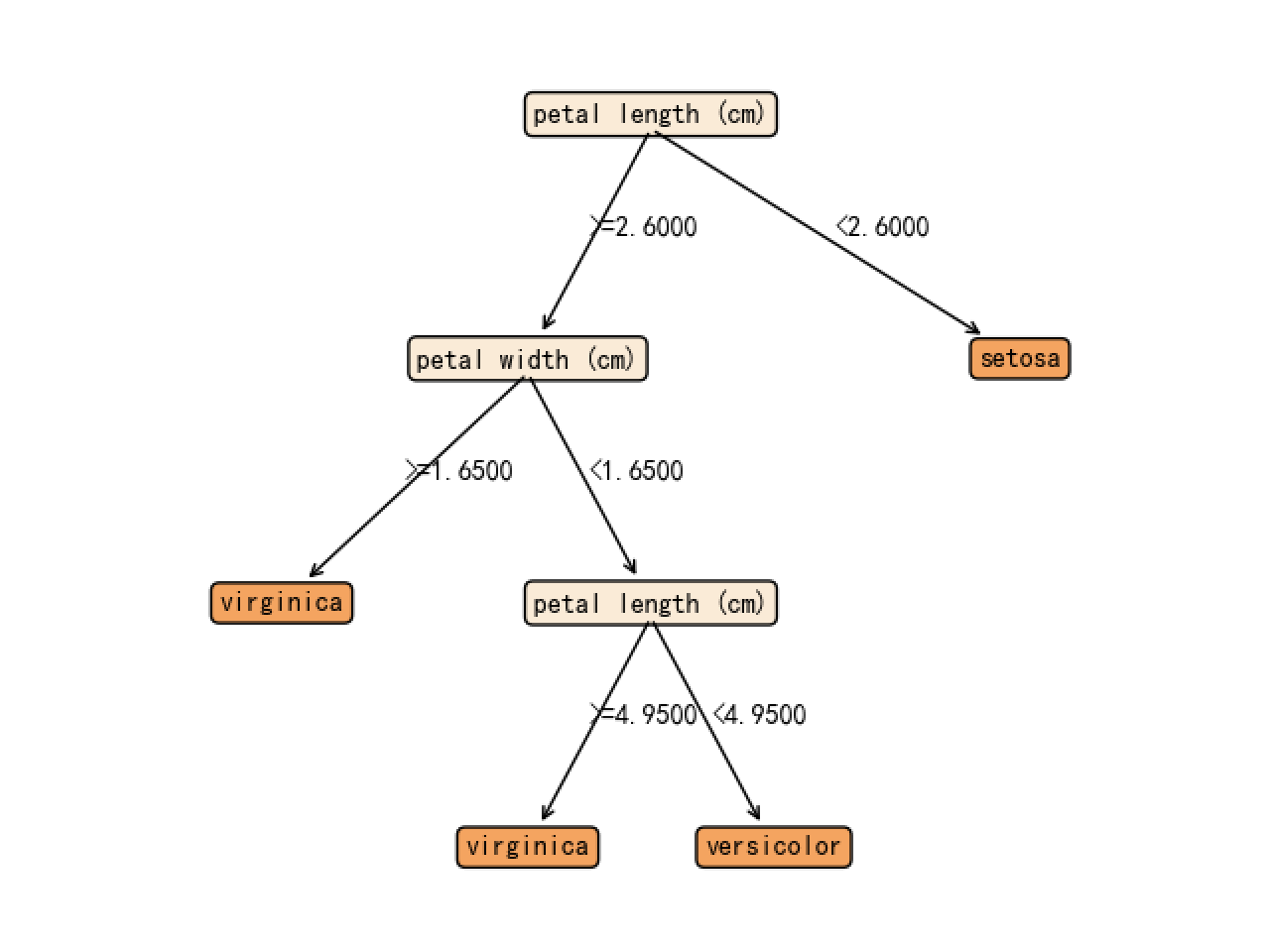


图表 4.1‑10 diabetes数据集



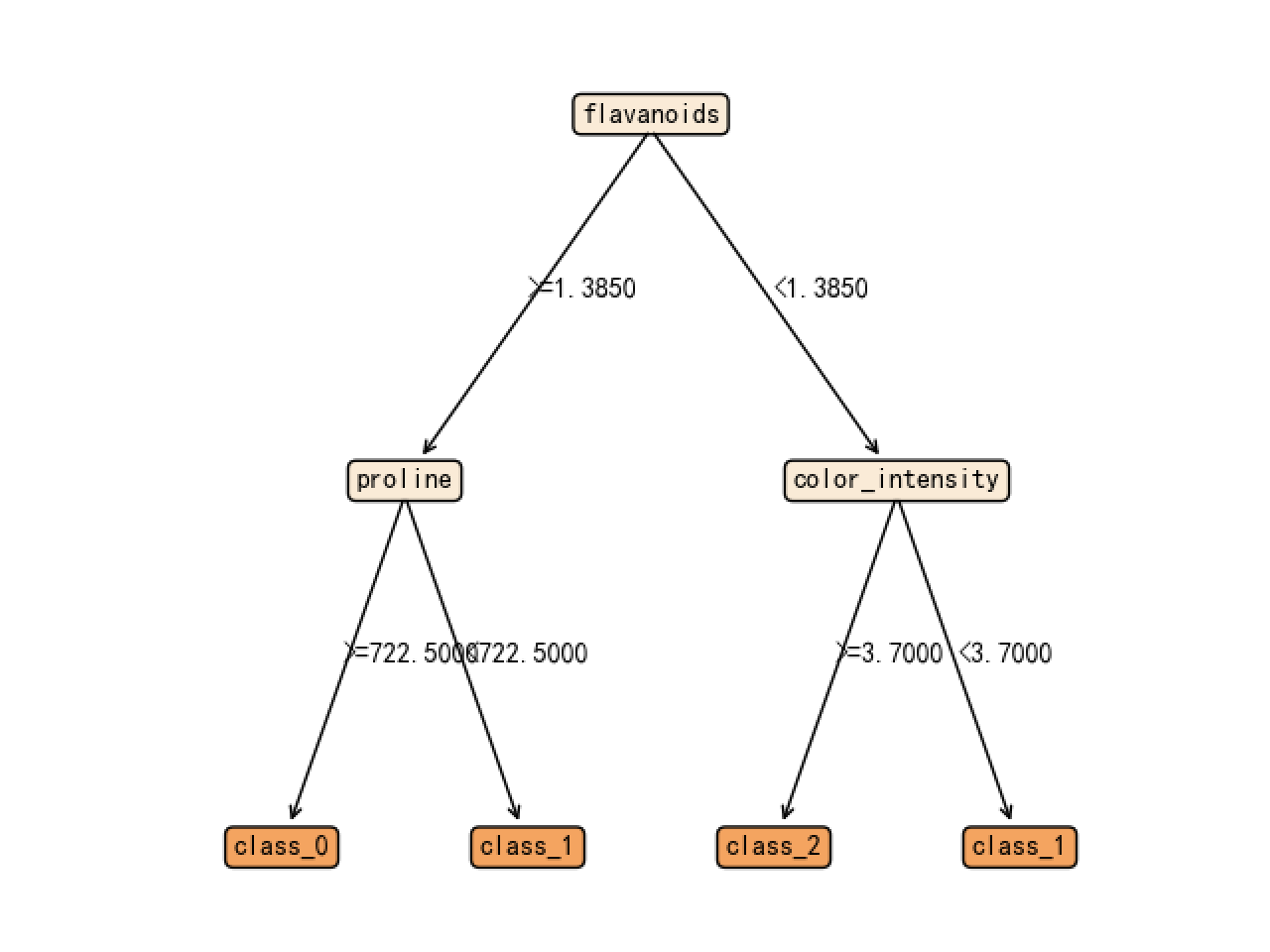
图表 4.1‑11 diabetes数据集正确率

预剪枝；



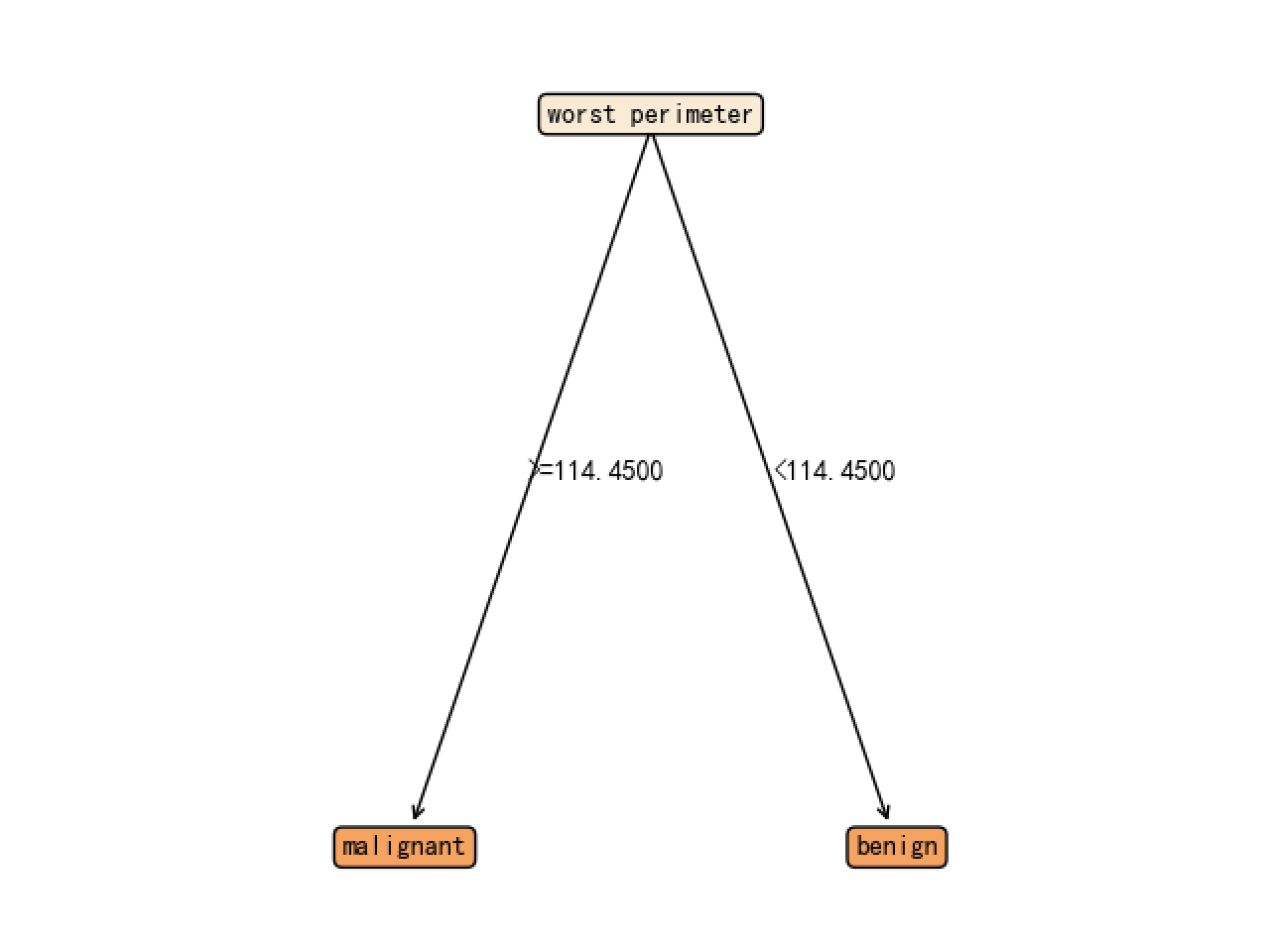


图表 4.1‑12 预剪枝的iris数据集基于信息增益决策树



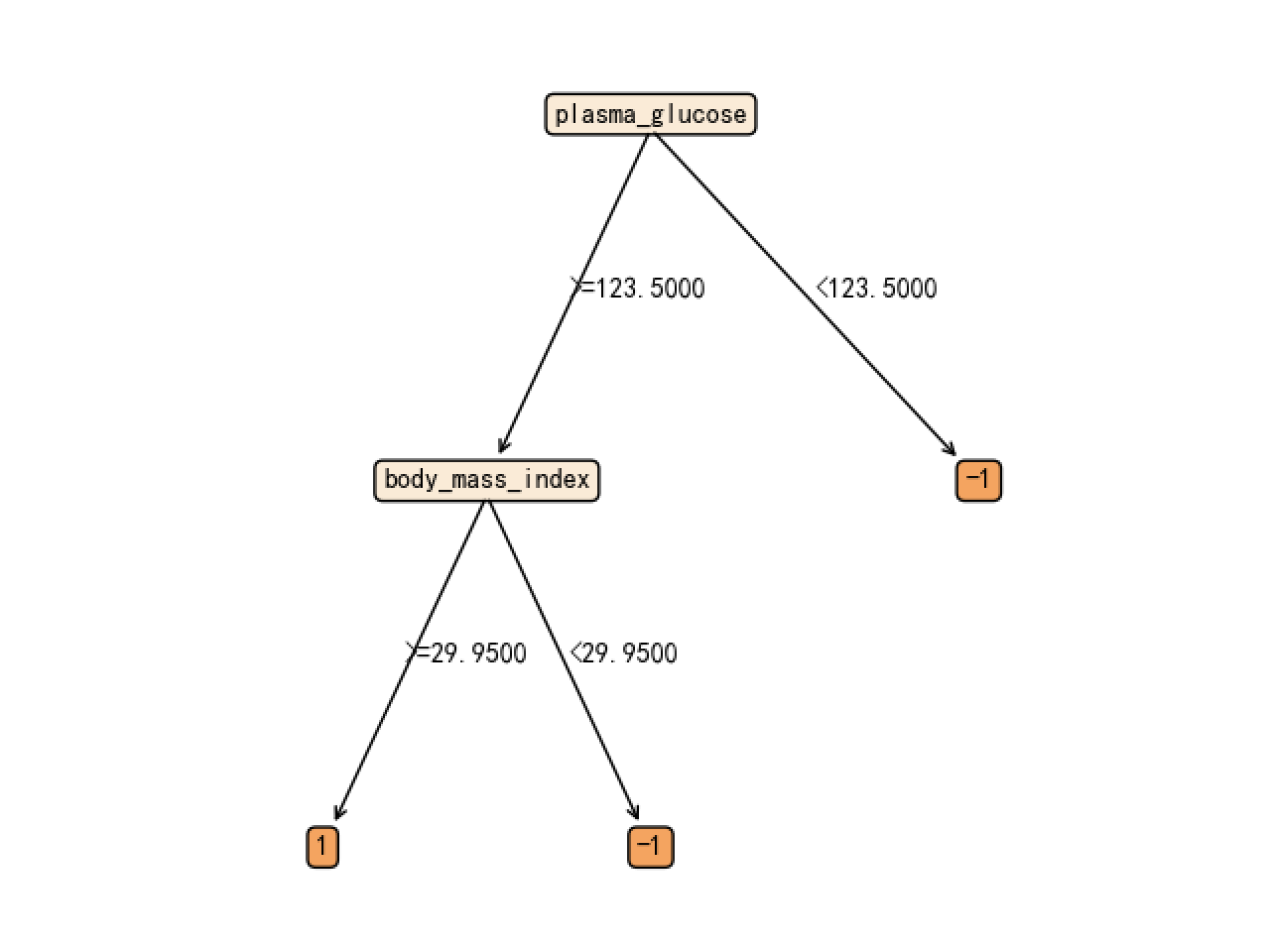


图表 4.1‑13 预剪枝的wine数据集基于信息增益决策树





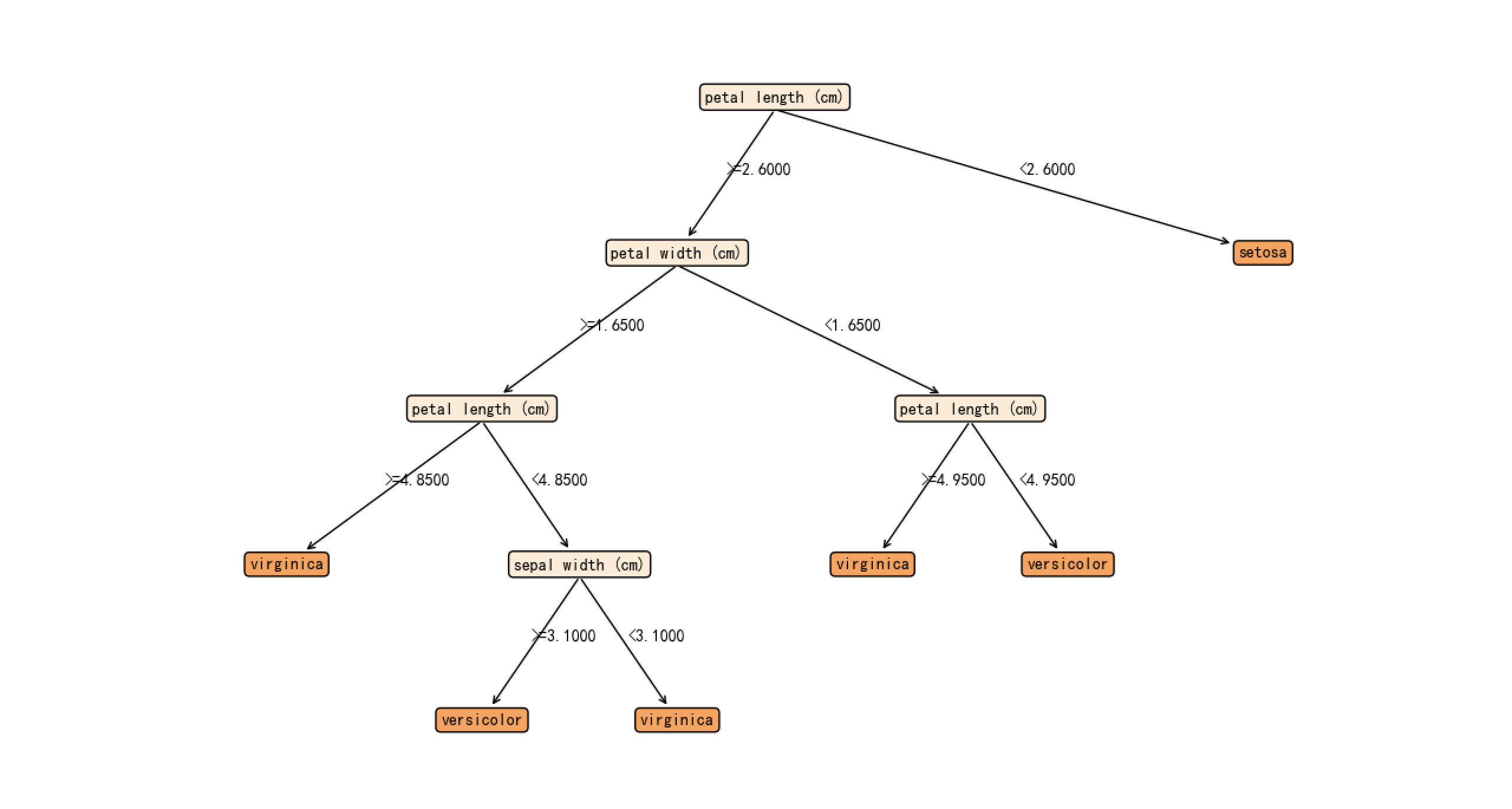
图表 4.1‑14 预剪枝的breast\_cancer数据集基于信息增益的决策树





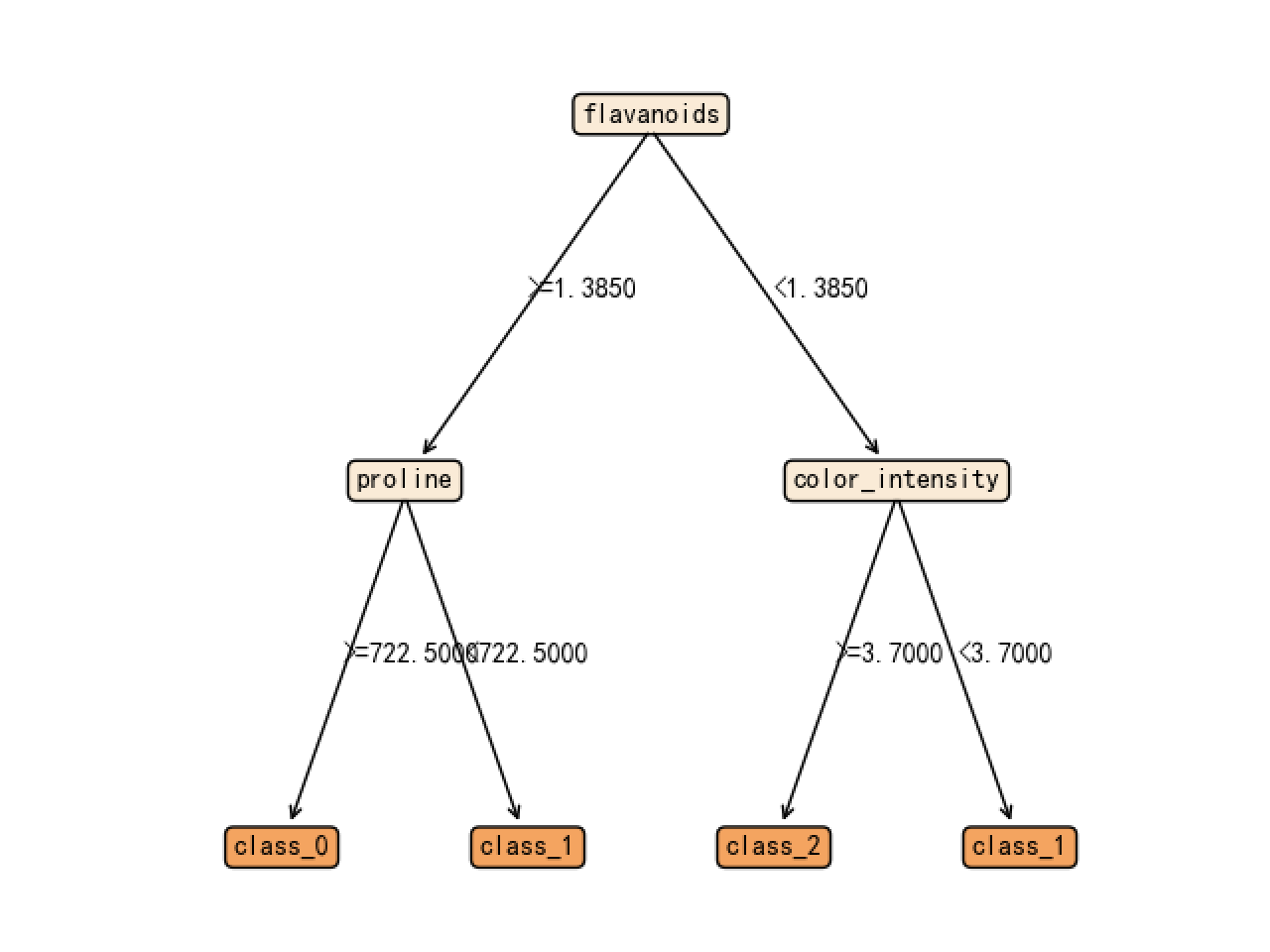
图表 4.1‑15 预剪枝的diabetes数据集的基于信息增益的决策树

**后剪枝：**



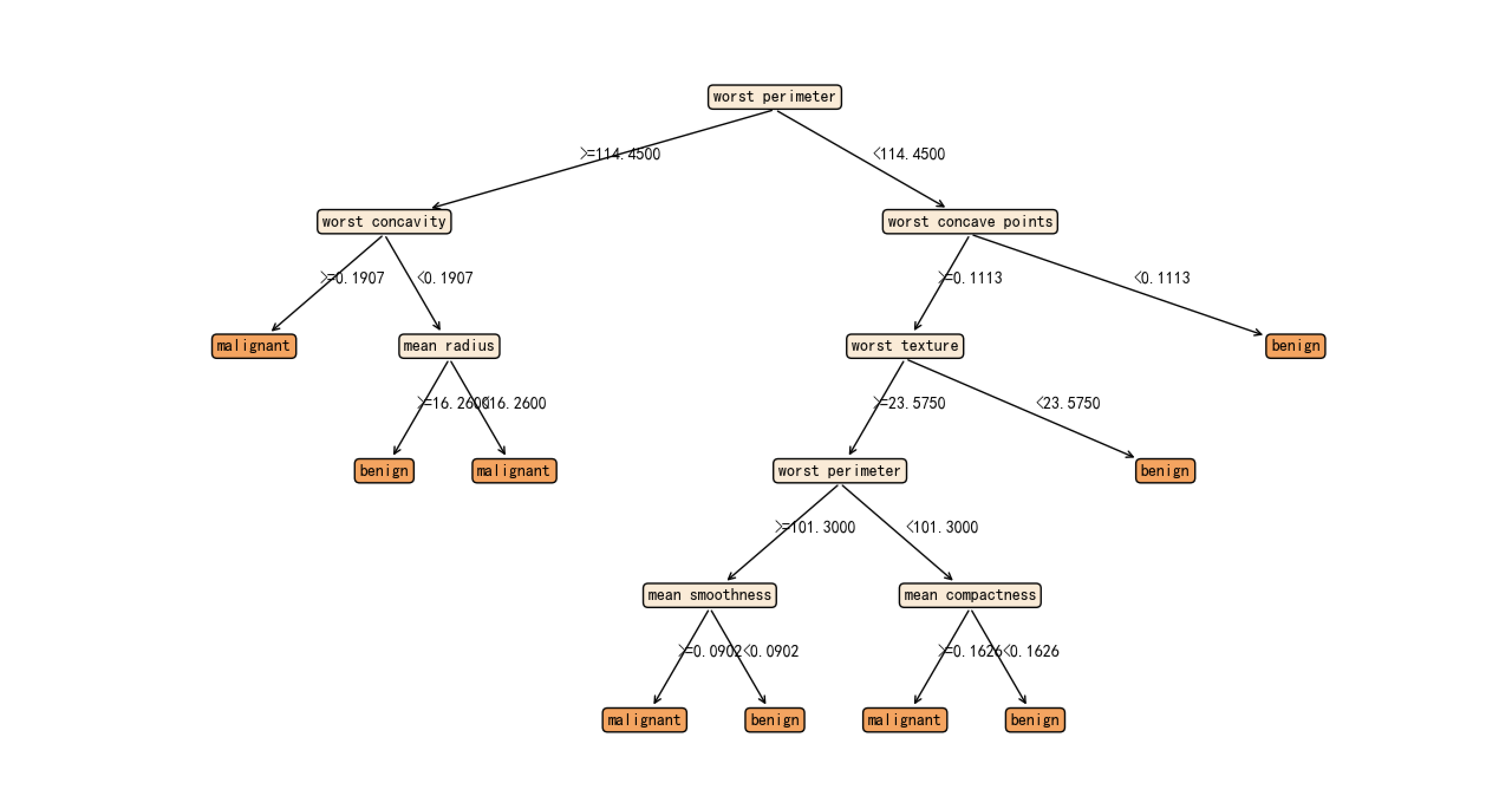


图表 4.1‑16 后剪枝的iris数据集基于信息增益的决策树



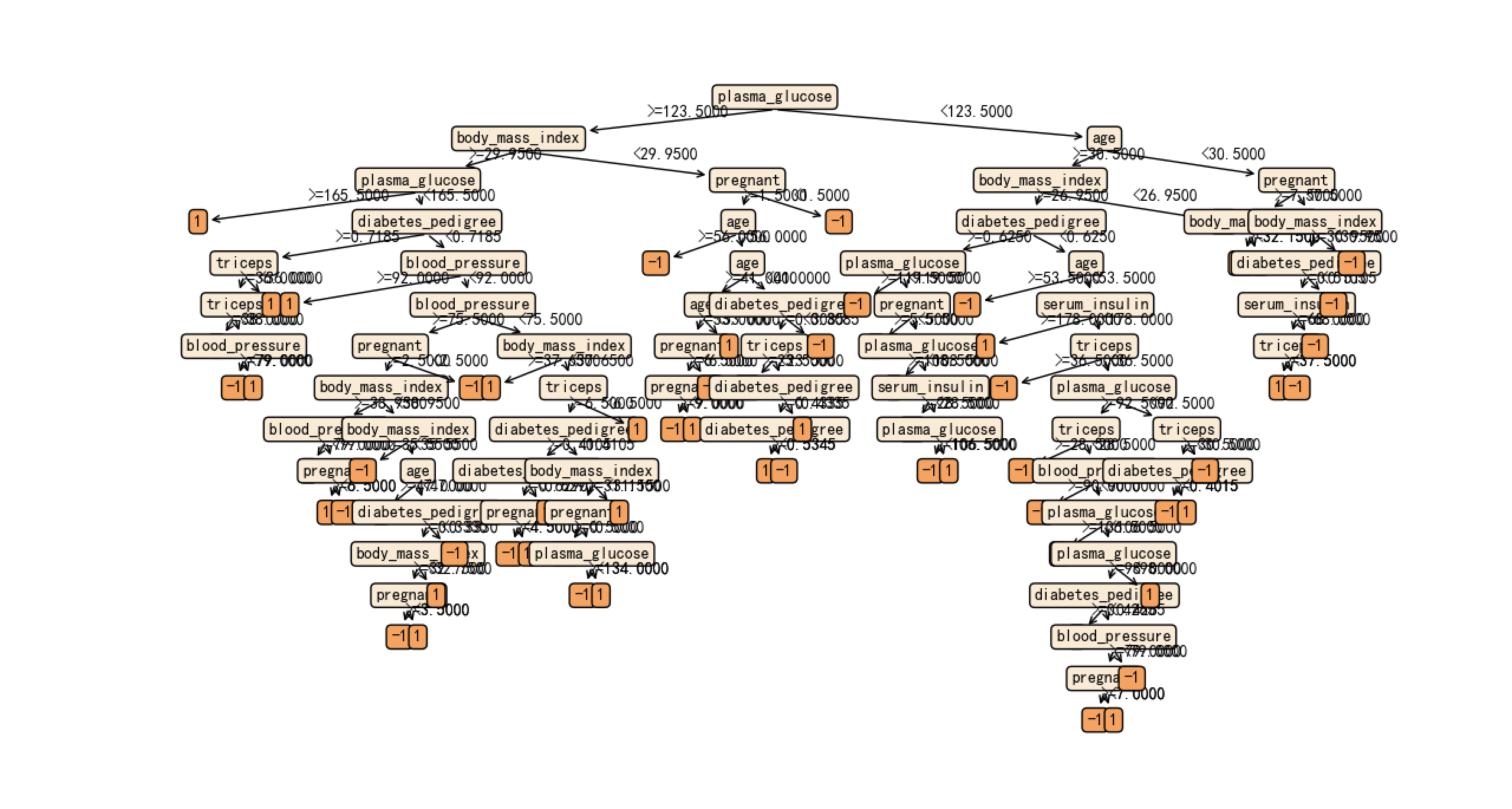


图表 4.1‑17 后剪枝的wine数据集基于信息增益的决策树





图表 4.1‑18 后剪枝的breast\_cancer数据集基于信息增益的决策树

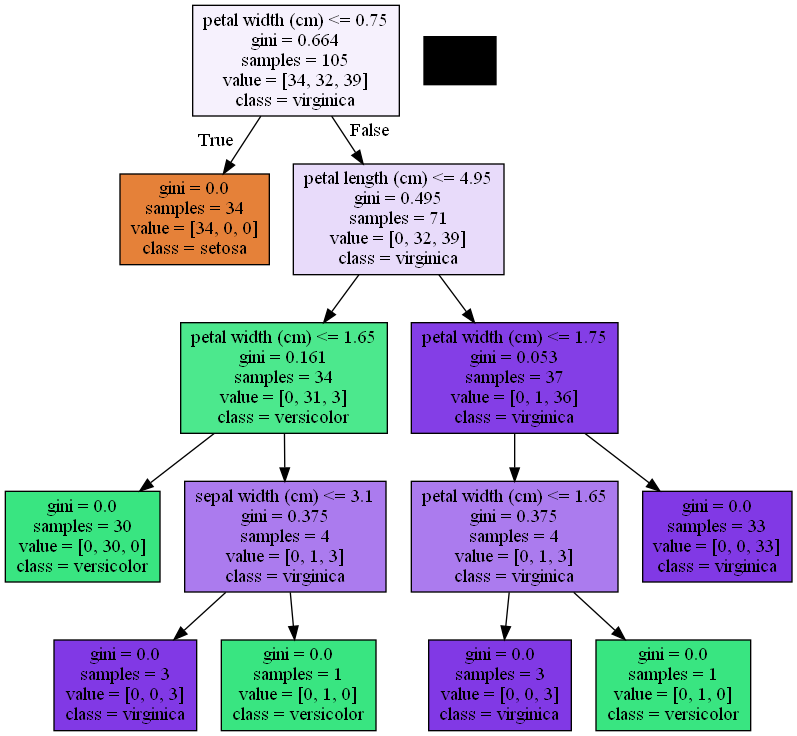




图表 4.1‑19 后剪枝的diabetes数据集基于信息增益的决策树

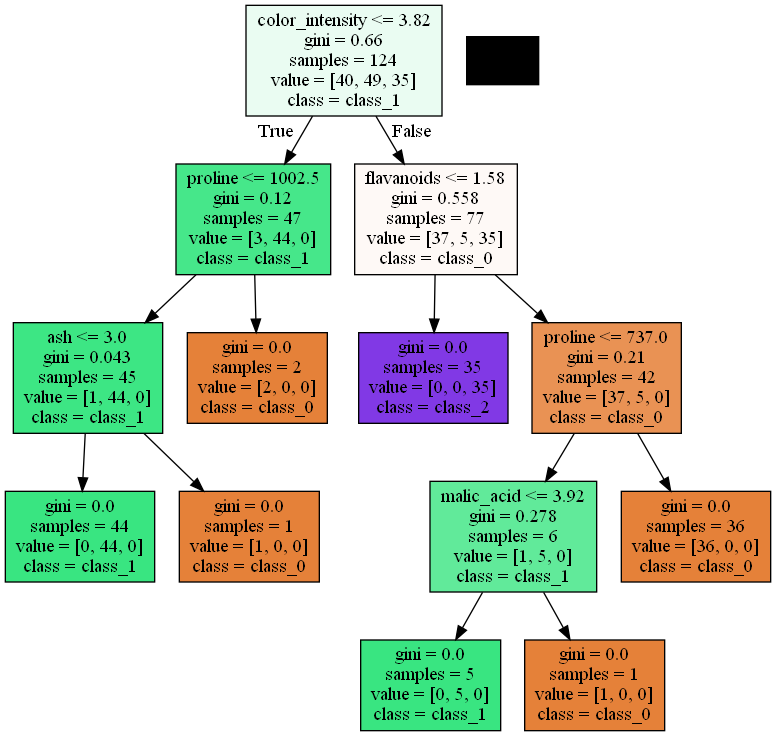
对于基尼指数：

未剪枝：



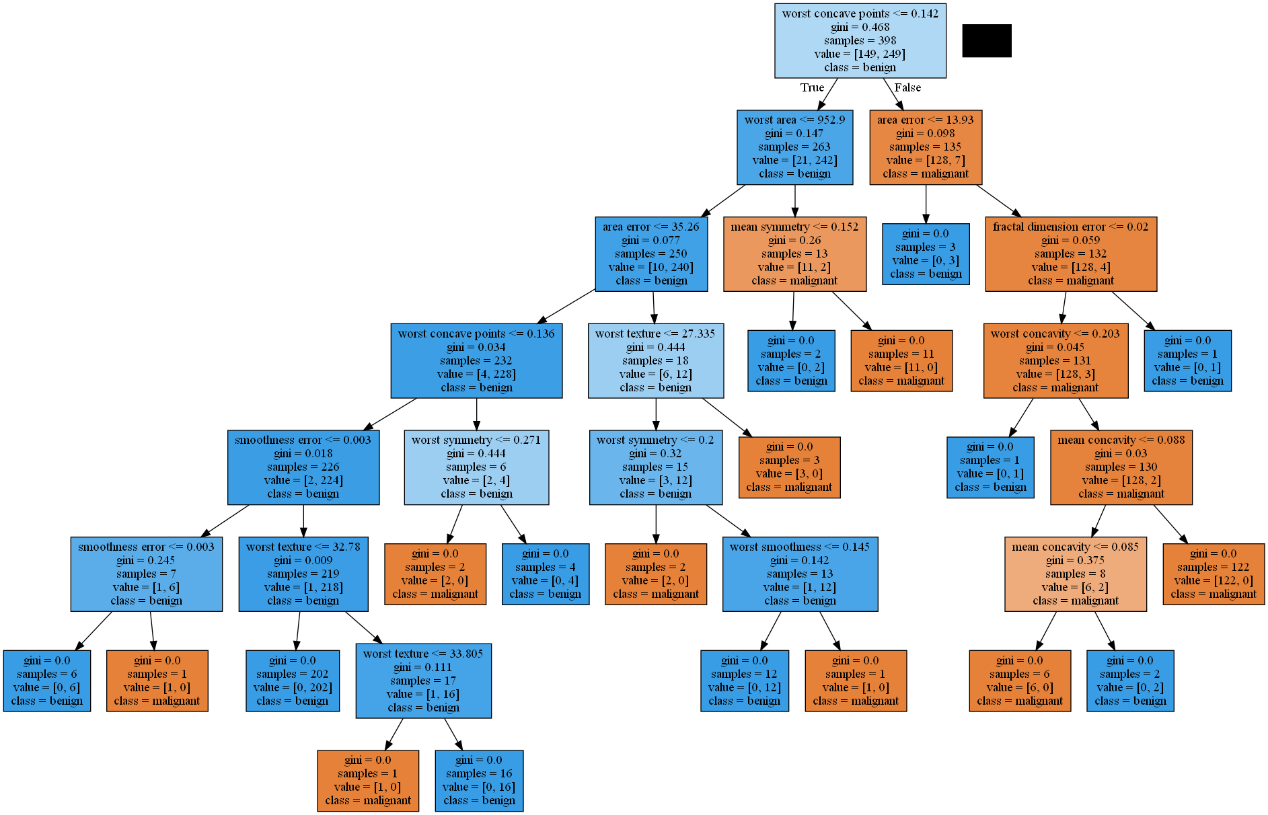


图表 4.1‑20 iris数据集基于基尼指数的决策树



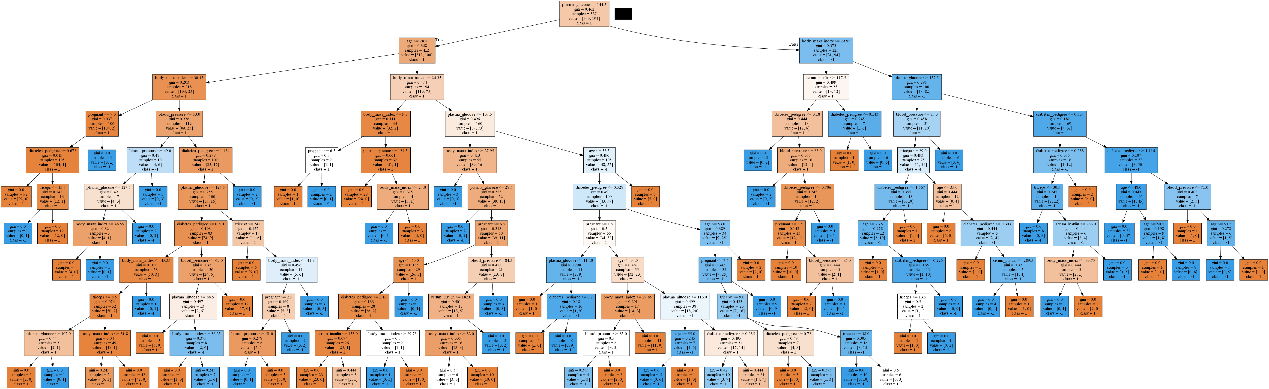


图表 4.1‑21wine数据集基于基尼指数的决策树





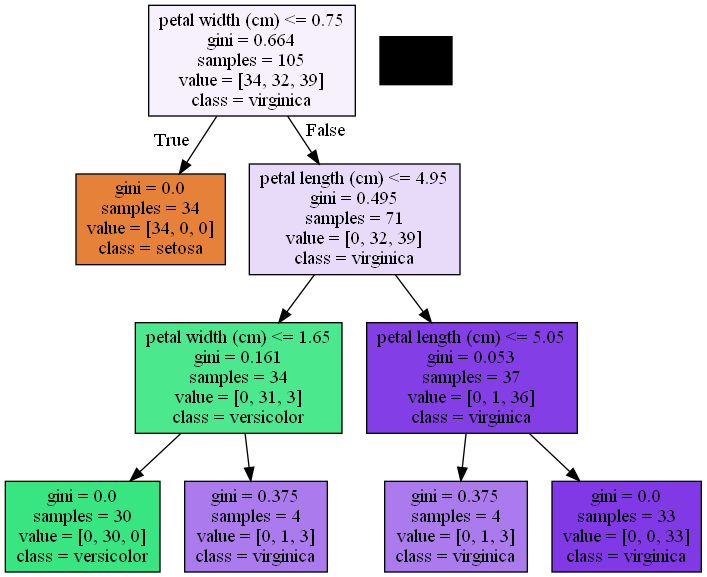
图表 4.1‑22 breast\_cancer数据集基于基尼指数的决策树





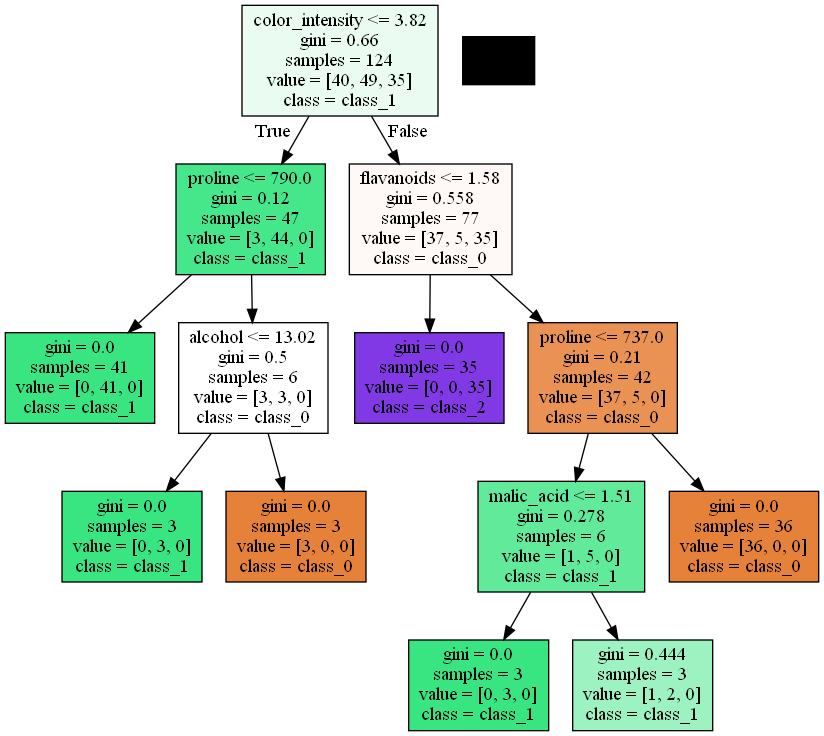
图表 4.1‑23 breast\_cancer数据集基于基尼指数的决策树

预剪枝：



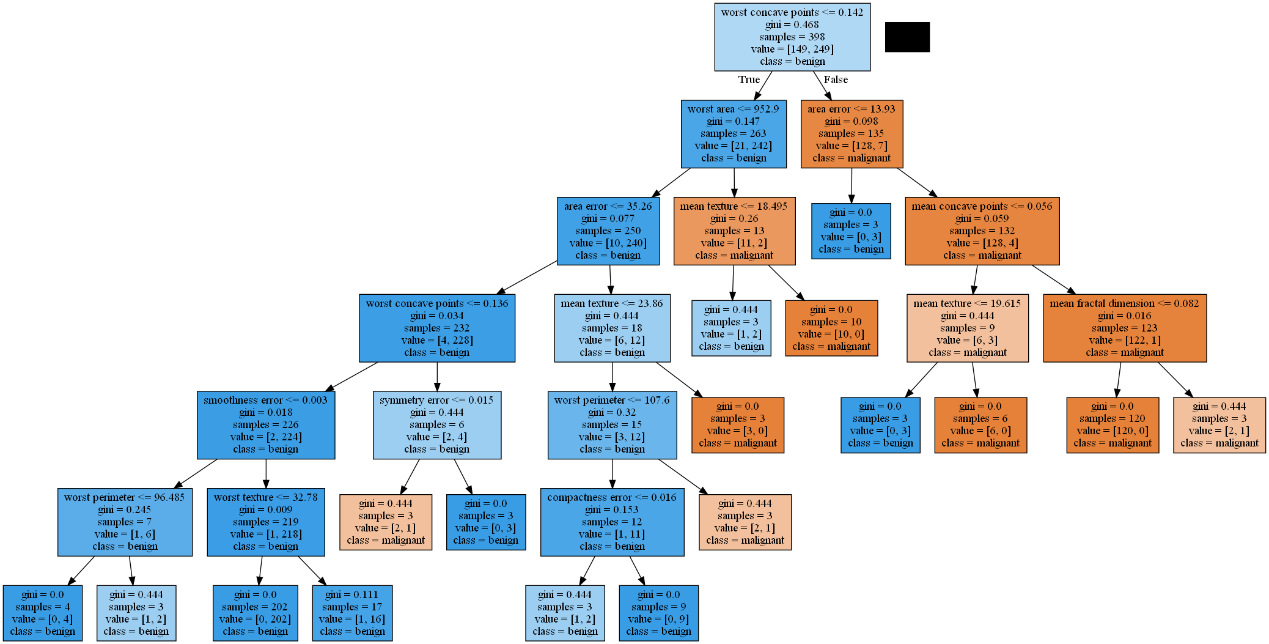


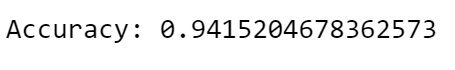
图表 4.1‑24 预剪枝后的iris数据集基于基尼指数的决策树



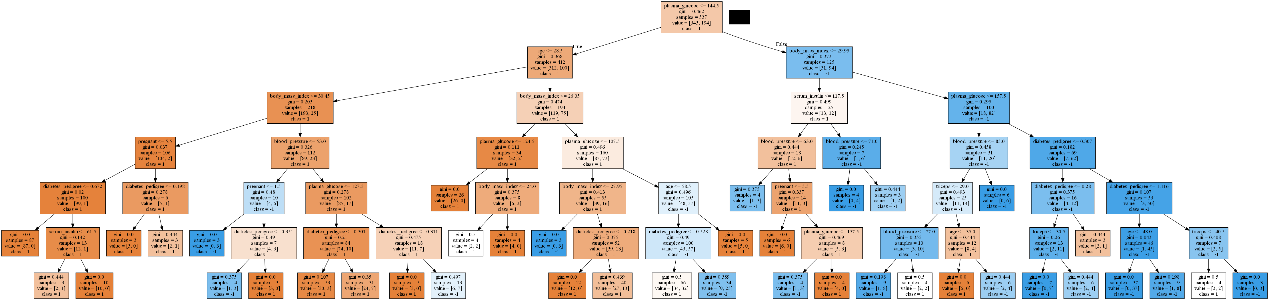


图表 4.1‑25 预剪枝后的wine数据集基于基尼指数的决策树





图表 4.1‑26预剪枝后的bresat\_cancer数据集基于基尼指数的决策树





图表 4.1‑27 预剪枝后的diabetes数据集基于基尼指数的决策树

可以画出如下表格:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 正确率 | 信息增益 | 信息增益(预剪枝) | 信息增益(后剪枝) | 基尼指数(预剪枝) | 基尼指数(未剪枝) |
| Iris | 96.6% | 100% | 100% | 97.9% | 97.8% |
| wine | 97.2% | 94.4% | 94.4% | 96.2% | 94.4% |
| breast\_cancer | 91.2% | 86.8% | 91.2% | 94.2% | 91.2% |
| diabetes | 72.7% | 72.7% | 72.7% | 74.9% | 74.0% |

表格 实验结果

对比可以发现，iris数据集因为数据量比较小的原因，拟合的效果比较好，最后的正确率也比较高。而breast\_cancer和diabetes数据集涉及到的数据量和特征比较多，因此，训练要消耗比较长的时间，并且，数据量的增大也导致决策树过拟合，导致树的规模过大，测试的效果也就不理想。比如diabetes数据集，在未剪枝的情况下树的规模达到11层，并且很多节点包含的样本太少，出现明显的过拟合情况。如果限制节点的大小和树的深度，就可以提高准确率，训练速度也会高很多。

预剪枝对于决策树通常会导致欠拟合，从diabetes数据集可以看出，未剪枝和后剪枝时候的决策树是十分庞大的，而预剪枝只有两个节点，说明欠拟合程度比较严重，泛化能力可能不是很好。而后剪枝效果比较好，适合大多数情况下使用。

### 代码结构，核心代码简要分析

重要代码注释如下：

1. **def** geerate\_tree(self, X\_train, y\_train):
2. my\_tree = Node()#创建根节点
3. my\_tree.leaf\_num = 0#初始化叶子节点数量
4. **if** y\_train.nunique() == 1:  # 如果只有一个类别，则为叶子节点
5. my\_tree.is\_leaf = True#叶子节点
6. my\_tree.leaf\_class = y\_train.values[0]#叶子节点的类别
7. my\_tree.high = 0#叶子节点的高度
8. my\_tree.leaf\_num += 1#叶子节点数量加1
9. **return** my\_tree#返回叶子节点
10. **if** X\_train.empty:  # 如果特征用完了，数据为空，则为叶子节点
11. my\_tree.is\_leaf = True
12. my\_tree.leaf\_class = y\_train.value\_counts().idxmax()  # 返回样本最多的类别
13. my\_tree.high = 0
14. my\_tree.leaf\_num += 1
15. **return** my\_tree
16. best\_feature\_name, best\_impurity = self.choose\_best\_feature\_to\_split(X\_train, y\_train)  # 选择最佳特征
17. my\_tree.feature\_name = best\_feature\_name#最佳特征名称
18. my\_tree.impurity = best\_impurity[0]  # 最佳分割值的名称
19. my\_tree.feature\_index = self.columns.index(best\_feature\_name)#最佳特征索引
20. feature\_values = X\_train.loc[:, best\_feature\_name]#最佳特征值
21. **if** len(best\_impurity) == 1:  # 如果是离散值
22. my\_tree.is\_continuous = False#离散值
23. unique\_values = feature\_values.unique()#最佳特征值的唯一值
24. sub\_X\_train = X\_train.drop(best\_feature\_name, axis=1)  # 删除最佳特征
25. max\_high = -1#初始化最大高度
26. **for** value **in** unique\_values:  # 对每一个特征值
27. my\_tree.subtree[value] = self.geerate\_tree(sub\_X\_train[feature\_values == value],
28. y\_train[feature\_values == value])#递归生成子树
29. **if** my\_tree.subtree[value].high > max\_high:#如果子树的高度大于最大高度
30. max\_high = my\_tree.subtree[value].high  # 取最大的高度为子树高度
31. my\_tree.leaf\_num += my\_tree.subtree[value].leaf\_num  # 添加子树的叶子数量
32. my\_tree.high = max\_high + 1
33. **else**:  # 如果是连续值
34. my\_tree.is\_continuous = True
35. my\_tree.split\_value = best\_impurity[1]  # 最佳分割点
36. up\_part = '>={:.4f}'.format(my\_tree.split\_value)#大于等于最佳分割点的特征值
37. down\_part = '<{:.4f}'.format(my\_tree.split\_value)#小于最佳分割点的特征值
38. my\_tree.subtree[up\_part] = self.geerate\_tree(X\_train[feature\_values >= my\_tree.split\_value],
39. y\_train[feature\_values >= my\_tree.split\_value])#递归生成大于等于最佳分割点的子树
40. my\_tree.subtree[down\_part] = self.geerate\_tree(X\_train[feature\_values < my\_tree.split\_value],
41. y\_train[feature\_values < my\_tree.split\_value])
43. my\_tree.leaf\_num += my\_tree.subtree[up\_part].leaf\_num + my\_tree.subtree[down\_part].leaf\_num#添加子树的叶子数量
44. my\_tree.high = max(my\_tree.subtree[up\_part].high, my\_tree.subtree[down\_part].high) + 1#计算子树高度
45. **return** my\_tree

如何确定最佳分割点：

1. **def** choose\_best\_feature\_to\_split\_infogain(self, X\_train, y\_train):
2. feature\_names = X\_train.columns#特征名称
3. best\_feature\_name = None#最佳特征名称
4. best\_info\_gain = [float('-inf')]#最佳信息增益
5. entD = self.entropy(y\_train)  # 计算数据集的熵
6. **for** feature\_name **in** feature\_names:#对每一个特征
7. is\_continuous = True#设定为连续值
8. info\_gain = self.info\_gain(X\_train[feature\_name], y\_train, entD, is\_continuous)  # 对每个特征计算信息增益
9. **if** info\_gain[0] > best\_info\_gain[0]:#如果信息增益大于最佳信息增益
10. best\_info\_gain = info\_gain#更新最佳信息增益
11. best\_feature\_name = feature\_name#更新最佳特征名称
12. **return** best\_feature\_name, best\_info\_gain#返回最佳特征名称和最佳信息增益

### 本次实验解决的主要问题，主要收获

1. 每次实验主要依靠西瓜书中的公式，实现了一个决策树，并对4个UCI数据集进行了测试。从底层类开始一步步实现了决策树类，节点类，特征的选择以及递归调用建立子树。了解了决策树这个经典机器学习算法的原理，提高了自身的编程能力。
2. 遇到的主要问题集中在对于连续特征的处理，如何选取最佳划分点成为了关键。经过资料的查阅，明白了划分点依赖于该连续特征下的所有特征值的中点，遍历求出信息增益值最大的点，根据这点来划分连续特征。
3. 了解了过拟合问题对于算法的影响。在diabetes数据集中，过拟合的状况导致决策树规模过于庞大，因此，需要进行预剪枝或者后剪枝。
4. 由于时间的关系，没有自己实现对率回归的代码。但是对率回归本质上和信息增益是一样的，只不过少了求最大值的过程，而是直接通过对率回归求出最佳划分点，根据最佳划分点直接划分节点。

### 编码及内容撰写中的参考来源

1. 机器学习 周志华P73-P92
2. <https://blog.csdn.net/quinn1994/article/details/80083933>西瓜书学习（一）—决策树（上）
3. <https://blog.csdn.net/w417950004/article/details/77600913>【西瓜书笔记二】决策树

## 选择表4.2的西瓜数据集，采用”队列”数据结构，编程实现基于信息增益划分选择的非递归的深度优先搜索的未剪枝决策树算法，计算验证集精度。

### 编程题目理解

题目要求利用队列的数据结构，将原本的递归建树改造为非递归的深度搜索。由于是深度搜索，所以应该采用的是栈这一先进后出的数据结构。并且，利用书中的西瓜数据集对树进行测试。由于是采用信息增益的方式，可以对4.1的代码稍加修改，将递归改造成栈建树，就可以完成。

### 栈结构非递归建树算法原理阐述

由于是要求栈结构建树，因此，需要一个list作为栈数据结构，而栈是一种先进后出的结构。因此，可以用来模拟树的深度遍历。用栈建树，每次取栈中最后一个元素作为当前节点，判断当前节点的状态，如果当前节点的高度已经超过最大高度，则跳过该节点。否则取该节点的最佳划分特征进行划分。划分后，再根据最佳划分特征，修改节点的数据集和标签集，遍历当前节点的最佳特征的所有特征值，将这些特征值作为节点判断。

因此，和递归算法类似，都是修改当前节点的训练集、标签集，找出最佳的划分特征进行划分出来的子节点加入栈，每次弹出最后一个元素作为当前节点继续划分，直到满足终止条件。

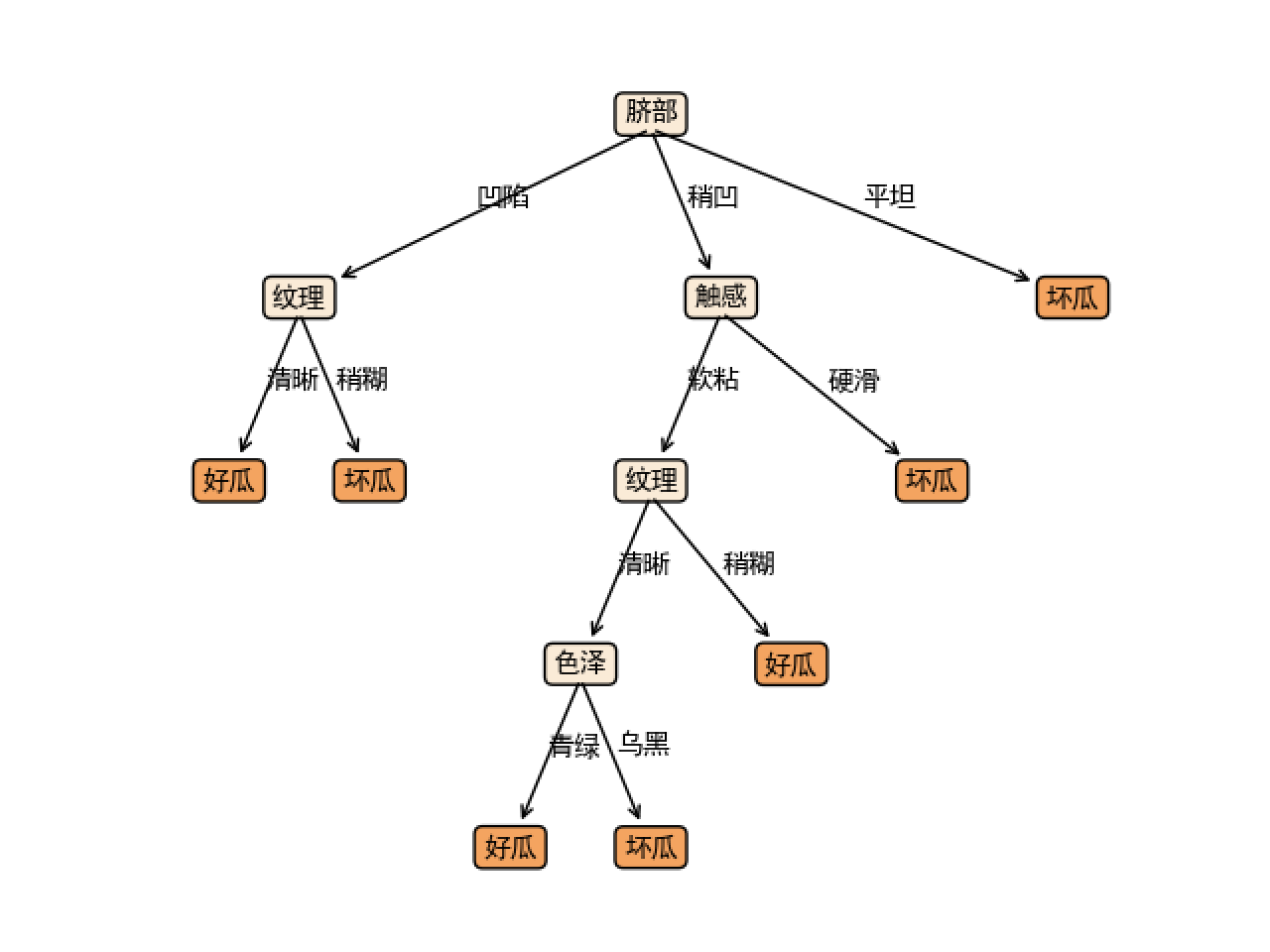
### 算法设计思路

非递归栈算法的设计思路如下：

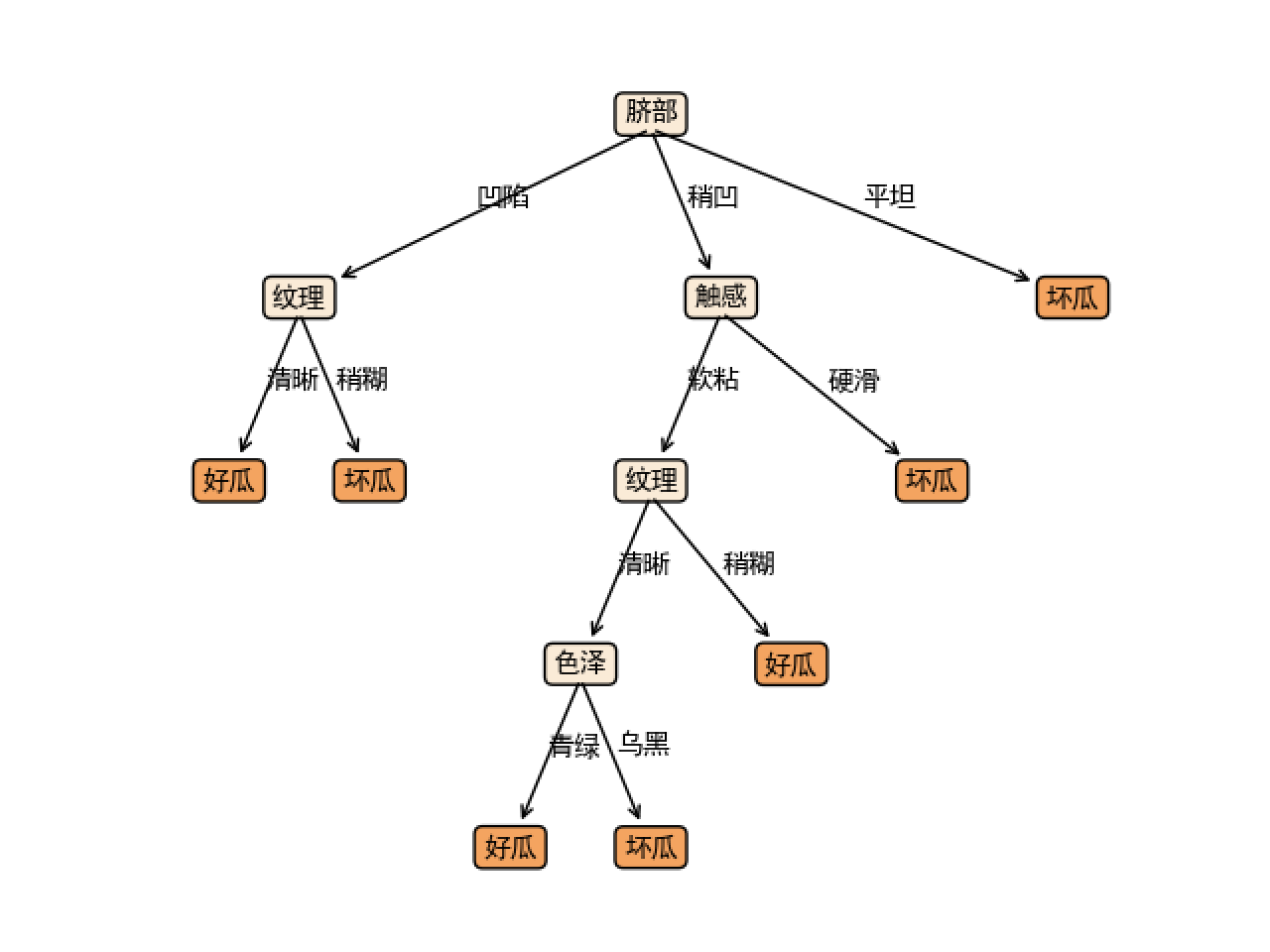
1. 设计一个栈结构node\_queue;
2. 如果栈结构非空，进行如下循环；
3. 取出最后一个元素作为curnode，如果curnode的高度大于限定的最大高度，则continue；
4. 否则，取出curnode的最佳特征的唯一值，对这些值进行如下循环；
5. 创建子节点nextnode，该节点的数据集为curnode的训练集对应的最佳特征的特征值的子集。标签集为curnode的标签集对应的最佳特征的特征值的子集。
6. 如果nextnode的数据集、标签集、属性集为空，则将该节点标记为叶子节点，设置nextnode的标签为nextnode数据集中最多的样本的标签。
7. 如果不为空，根据nextnode的数据集选出最佳特征，赋值对应的属性，将curnode的subtree添加nextnode，node\_queue添加nextnode，返回4.
8. 4循环结束后，退出循环。
9. 计算出当前决策树的各个节点包含的叶子数量，高度，方便接下来绘图。
10. 画出决策树，计算验证集正确率。

### 实验流程、测试结果及分析

为了显示出深度遍历，将在每次入栈、出栈的时候输出对应的节点标签名，进行检查。并且，最后的验证集将会输入到决策树中，跟原本的验证集的标签集进行对比，计算精度。

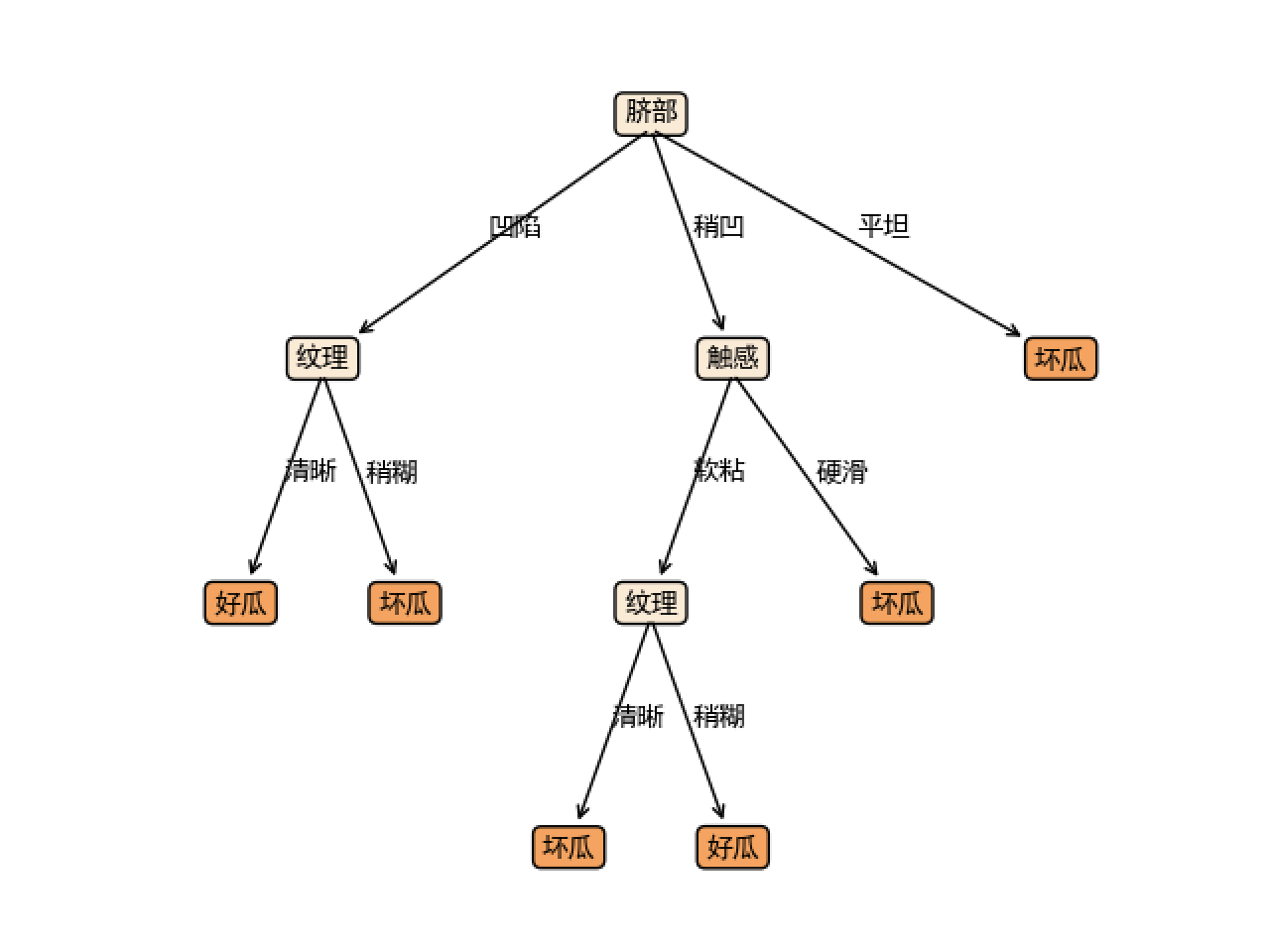


图表 4.2‑1 利用递归生成的基于信息增益的未剪枝决策树

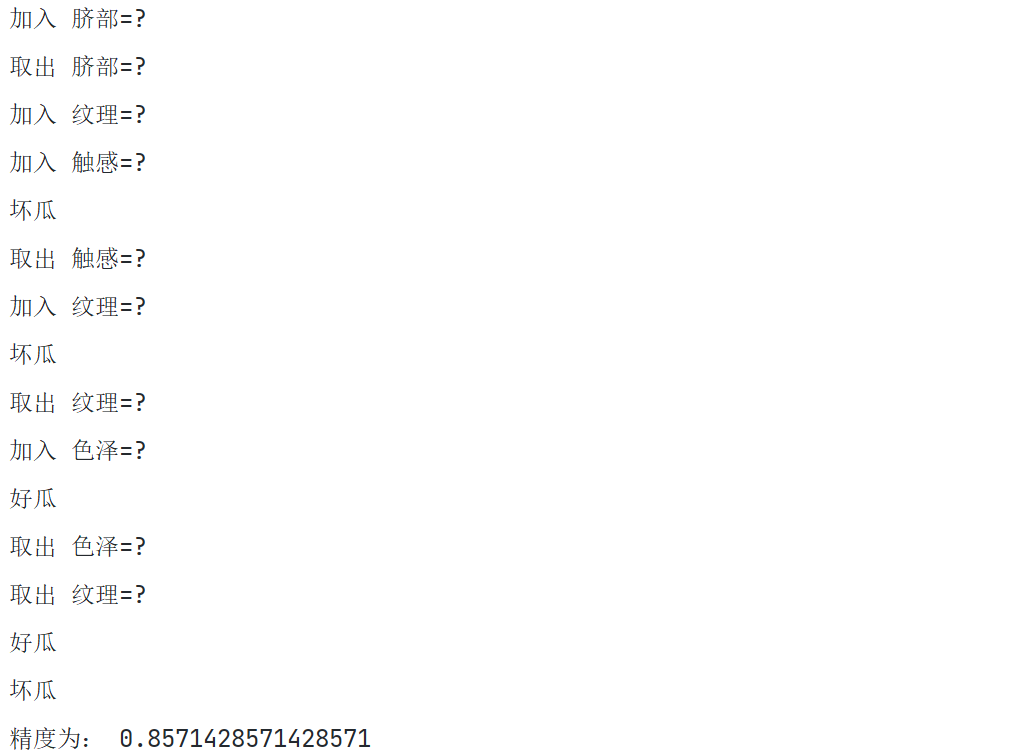


图表 4.2‑2 未限制高度的栈结构决策树

可以看出，未限制高度时当前决策树和遍历生成的决策树结构一致。



图表 4.2‑3 限制最大高度为2的决策树



图表 4.2‑4 遍历节点顺序以及精度

从图表4.2-4可以看出，利用栈结构生成的深度遍历决策树确实是深度遍历，且在限制最大高度的情况下，精度为85.7%。

### 代码结构，核心代码简要分析

核心代码如下，为深度遍历循环：



图表 ‑5 核心代码

### 本次实验解决的主要问题，主要收获

这次实验主要解决了基于栈结构的非递归决策树的建立问题。遇到的问题主要在于节点的数据集赋值需要特别注意为原来数据集的子集，并且需要特别注意节点的高度和叶子节点的高度问题，不然会报错空数值。主要收获在于更加了解了决策树算法的原理，可以自己实现底层的决策树代码，对信息增益公式了解更加深刻。对于栈结构的深度搜索算法的编写更加熟练，可以轻松完成编程任务。

发现的主要问题在于第一个节点的最佳特征的选择上面，存在两个特征的信息增益值是相同的，如果选了第一个特征，那么画出来的图像和书上面会很不同，并且精度会降低很多。如果选择第二个特征，那么精度会提升至八十多。主要原因在于数据集太小，西瓜数据集只包含十七个数据，不足以支撑整个决策树，欠拟合。

### 编码及内容撰写中的参考来源

[1]<https://blog.csdn.net/qq_37691909/article/details/85235472>西瓜书课后题——第四章（决策树）

[2] 机器学习 周志华P73-P92

## 选择表4.2的西瓜数据集，改写为基于信息增益划分选择的非递归的广度优先搜索的未剪枝决策树算法，计算验证集的精度。

### 编程题目理解

题目要求利用队列结构实现非递归的广度优先搜索的未剪枝决策树算法。和4.2题目相似，只需将选择节点的顺序从先进后出改为先进先出，即可完成对应结构的实现。

### 队列的广度优先算法原理阐述

由于是要求队列结构建树，因此，需要一个list作为队列数据结构，而栈是一种先进先出的结构。因此，可以用来模拟树的广度遍历。用队列建树，每次取队列中最后一个元素作为当前节点，判断当前节点的状态，如果当前节点的高度已经超过最大高度，则跳过该节点。否则取该节点的最佳划分特征进行划分。划分后，再根据最佳划分特征，修改节点的数据集和标签集，遍历当前节点的最佳特征的所有特征值，将这些特征值作为节点判断。

因此，和递归算法类似，都是修改当前节点的训练集、标签集，找出最佳的划分特征进行划分出来的子节点加入队列，每次弹出第一个元素作为当前节点继续划分，直到满足终止条件。

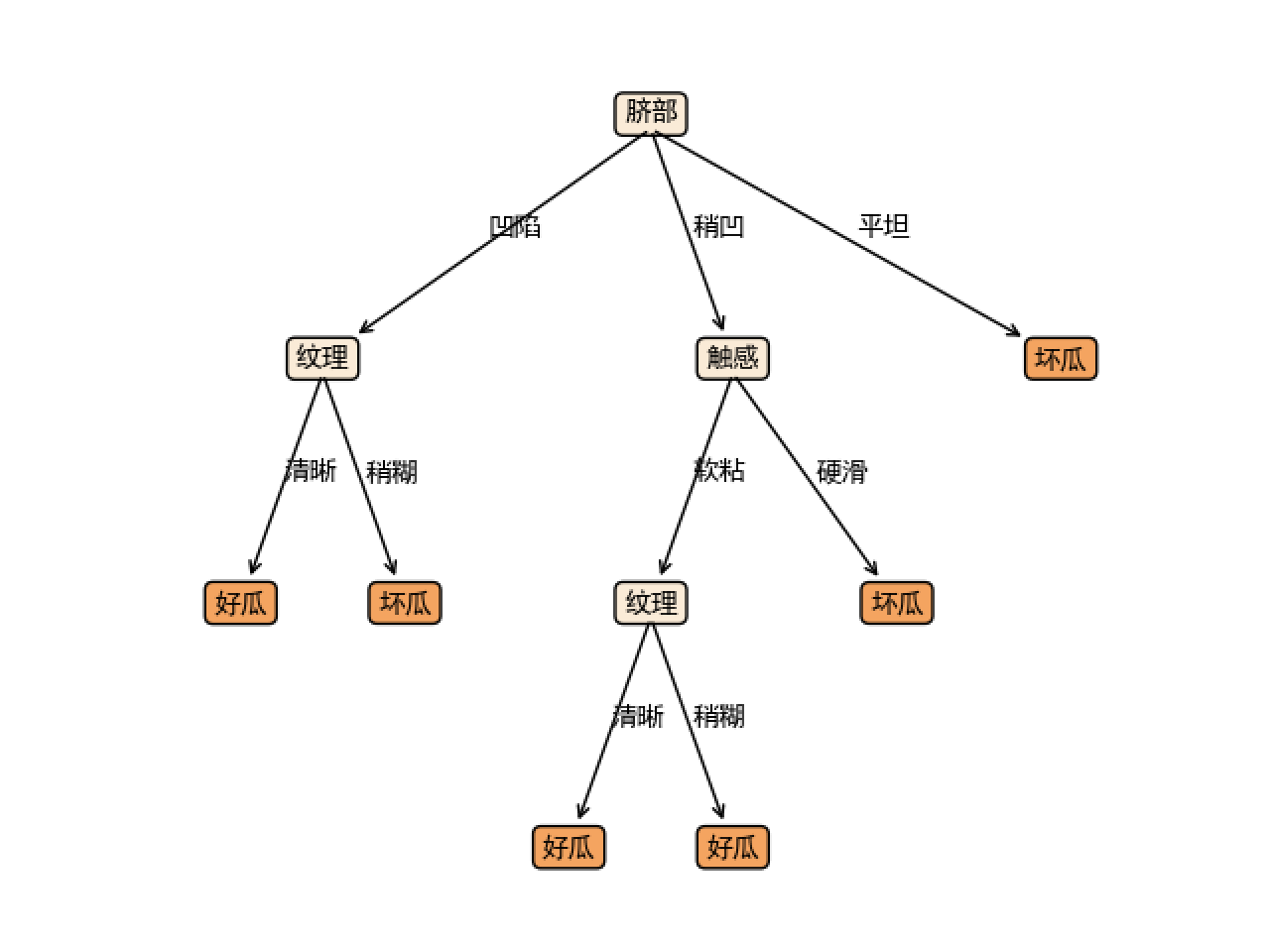
### 算法设计思路

非递归队列广度优先算法的设计思路如下：

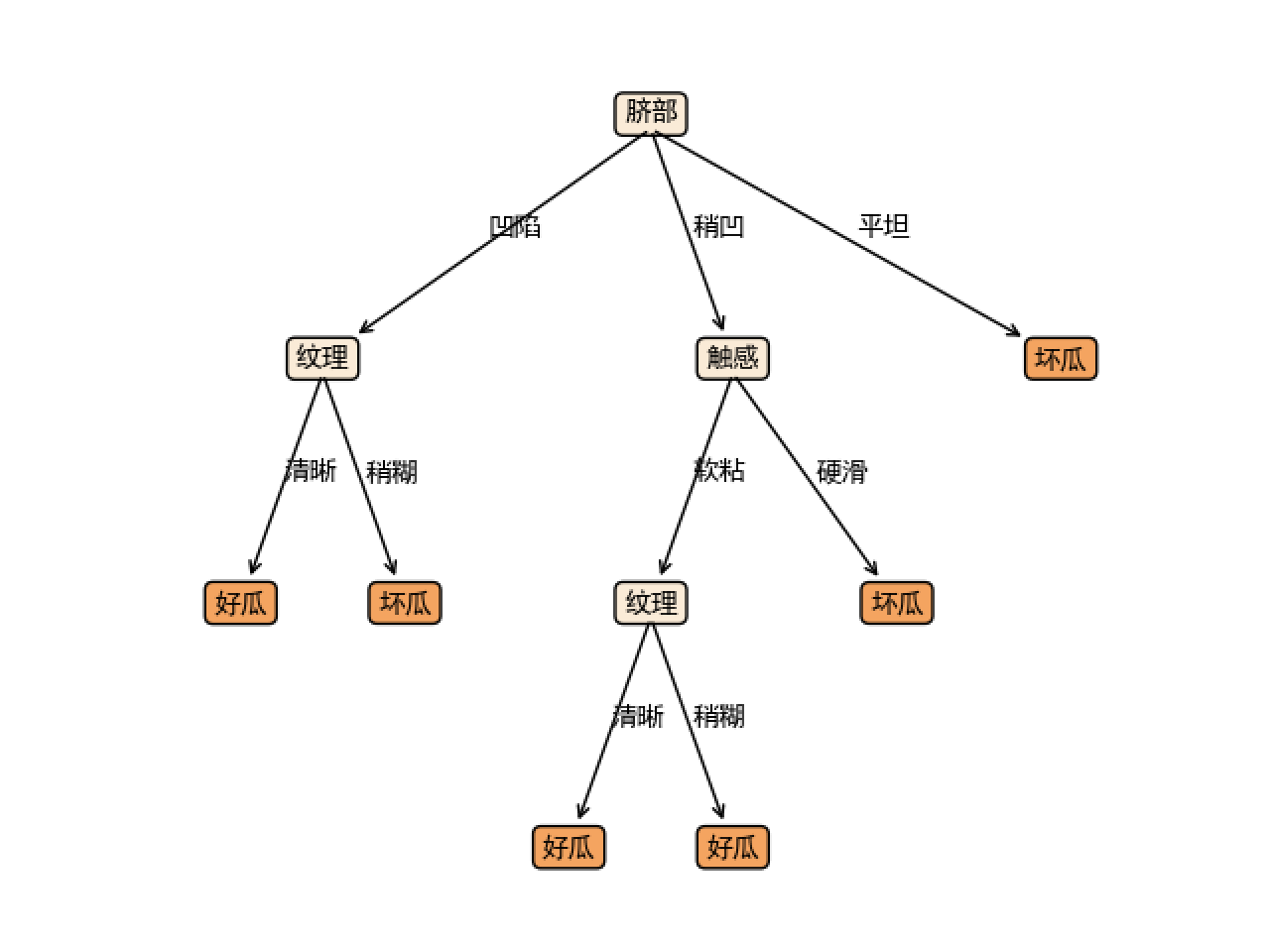
1. 设计一个队列结构node\_queue;
2. 如果栈结构非空，进行如下循环；
3. 取出最后一个元素作为curnode，如果curnode的高度大于限定的最大高度，则continue；
4. 否则，取出curnode的最佳特征的唯一值，对这些值进行如下循环；
5. 创建子节点nextnode，该节点的数据集为curnode的训练集对应的最佳特征的特征值的子集。标签集为curnode的标签集对应的最佳特征的特征值的子集。
6. 如果nextnode的数据集、标签集、属性集为空，则将该节点标记为叶子节点，设置nextnode的标签为nextnode数据集中最多的样本的标签。
7. 如果不为空，根据nextnode的数据集选出最佳特征，赋值对应的属性，将curnode的subtree添加nextnode，node\_queue添加nextnode，返回4.
8. 4循环结束后，退出循环。
9. 计算出当前决策树的各个节点包含的叶子数量，高度，方便接下来绘图。
10. 画出决策树，计算验证集正确率。

### 实验流程、测试结果及分析

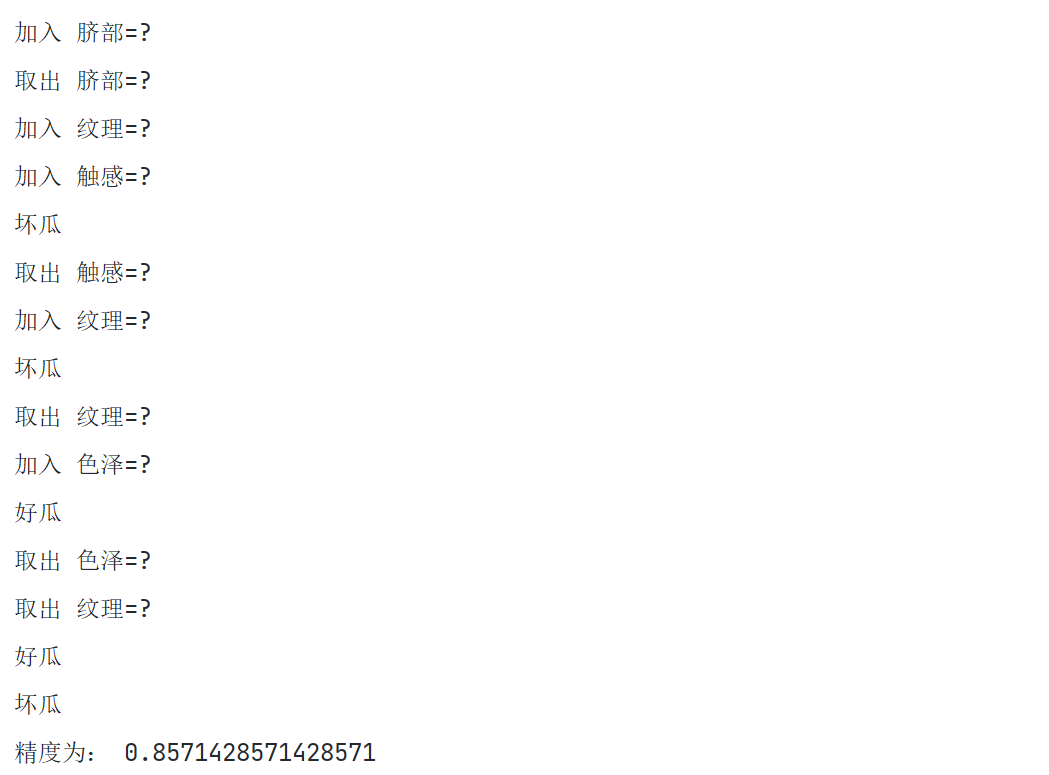
为了显示出广度遍历，将在每次入队，出队的时候输出对应的节点标签名，进行检查。并且，最后的验证集将会输入到决策树中，跟原本的验证集的标签集进行对比，计算精度。



图表 4.3‑1 递归生成的决策树



图表 4.3‑2 队列生成的决策树



图表 4.3‑3 对应的输出值以及精度

经过输出值的检查，可以发现确实是广度优先遍历，并且建树的结构和递归建树相同。精度和深度遍历一样，都为85.7%。

### 代码结构，核心代码简要分析

核心代码和深度遍历相同，唯一的不同在于修改了如下代码：

curnode = node\_queue.pop(0) # 出队，取当前节点为队顶元素

修改取元素的顺序，改为取队首元素。

### 本次实验解决的主要问题，主要收获

主要收获与问题已经在4.2中解决，唯一遇到的问题就是python List数据结构如何取队首元素，经过查阅已经解决该问题，list.pop(index)函数提供了index，可以弹出指定的索引元素。

### 编码及内容撰写中的参考来源

[1]<https://blog.csdn.net/qq_37691909/article/details/85235472>西瓜书课后题——第四章（决策树）

[2] 机器学习 周志华P73-P92