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

[中国地质大学(武汉) 1](#_Toc99611631)

[机器学习第二次作业 1](#_Toc99611632)

[[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](#_Toc99611633)

[编程题目理解 3](#_Toc99611634)

[决策树算法原理阐述 3](#_Toc99611635)

[算法设计思路 5](#_Toc99611636)

[实验流程、测试结果及分析 7](#_Toc99611637)

[代码结构，核心代码简要分析 16](#_Toc99611638)

[本次实验解决的主要问题，主要收获 18](#_Toc99611639)

[编码及内容撰写中的参考来源 18](#_Toc99611640)

[[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。 19](#_Toc99611641)

[编程题目理解 19](#_Toc99611642)

[决策树算法原理阐述 19](#_Toc99611643)

[算法设计思路 19](#_Toc99611644)

[实验流程、测试结果及分析 19](#_Toc99611645)

[代码结构，核心代码简要分析 19](#_Toc99611646)

[本次实验解决的主要问题，主要收获 19](#_Toc99611647)

[编码及内容撰写中的参考来源 19](#_Toc99611648)

[[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。 19](#_Toc99611649)

[编程题目理解 19](#_Toc99611650)

[决策树算法原理阐述 20](#_Toc99611651)

[算法设计思路 20](#_Toc99611652)

[实验流程、测试结果及分析 20](#_Toc99611653)

[代码结构，核心代码简要分析 20](#_Toc99611654)

[本次实验解决的主要问题，主要收获 20](#_Toc99611655)

[编码及内容撰写中的参考来源 20](#_Toc99611656)

# [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。

## 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：

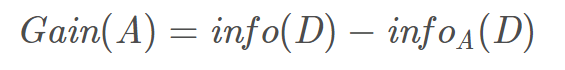


图表 1 决策树算法伪代码

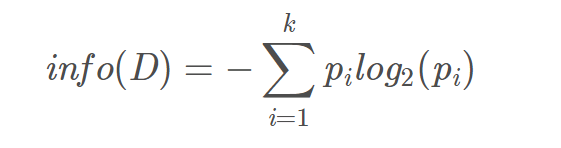
其中，根据数据特征函数的不同，可以划分不同的算法。比较经典的算法有ID3、C4.5、Cart算法。

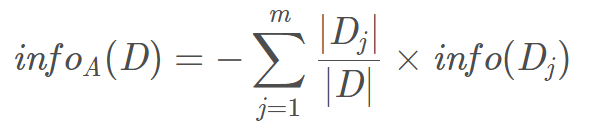
ID3：

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 2 基于对率回归算法

## 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。

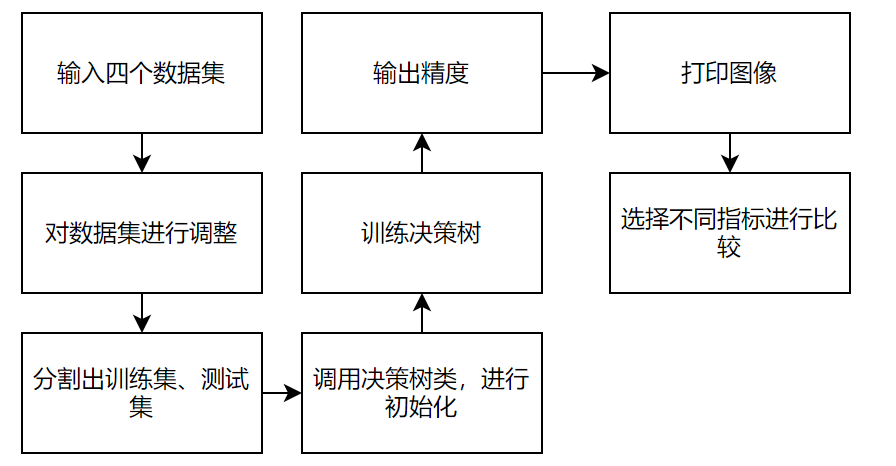
**对于如何选择最佳信息增益，分别根据离散值以及连续值进行判断:**

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。

**最后，利用matplotlib对该树进行打印。打印步骤如下：**

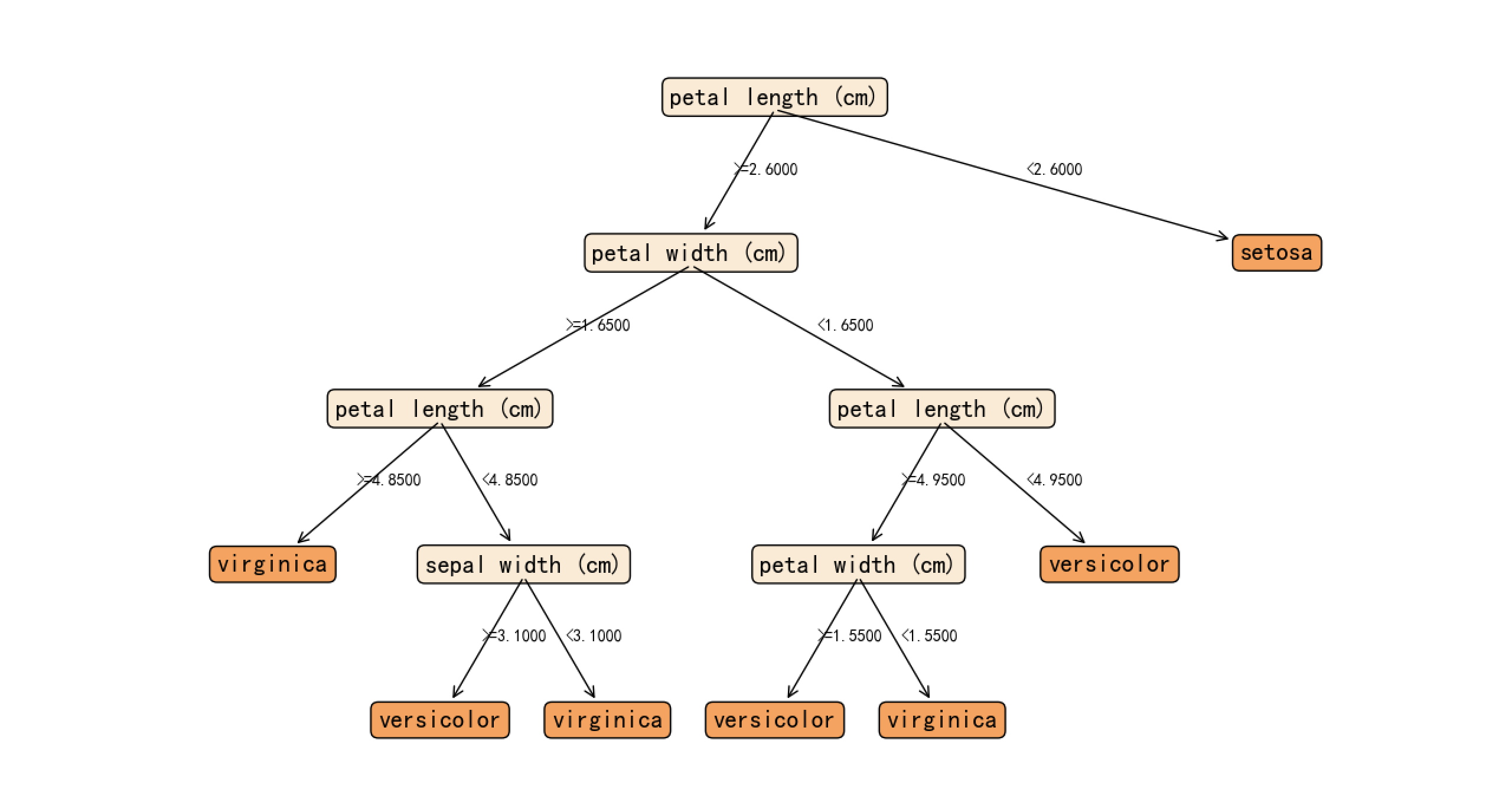
1. 根据叶子节点的个数，选择合适的x偏移量和y偏移量进行打印。
2. 输入节点，判定这个节点是不是叶子节点，如果是叶子节点，直接打印。
3. 如果不是叶子节点，递归调用该函数，增加x偏移量和y偏移量。并且打印线的名称。

## 实验流程、测试结果及分析



图表 3 决策树实验流程图

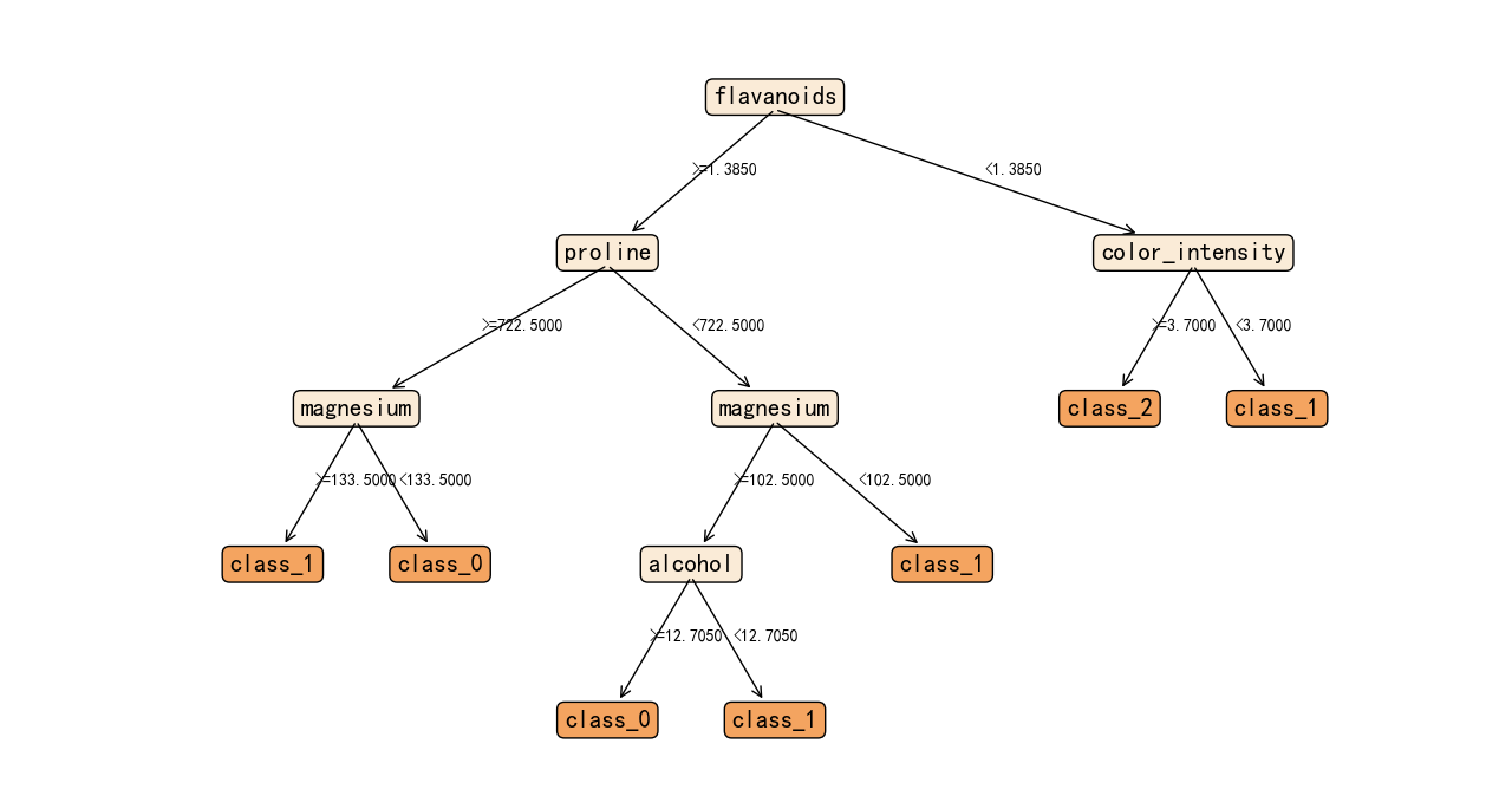
对于信息增益：



图表 4 信息增益IRIS数据集



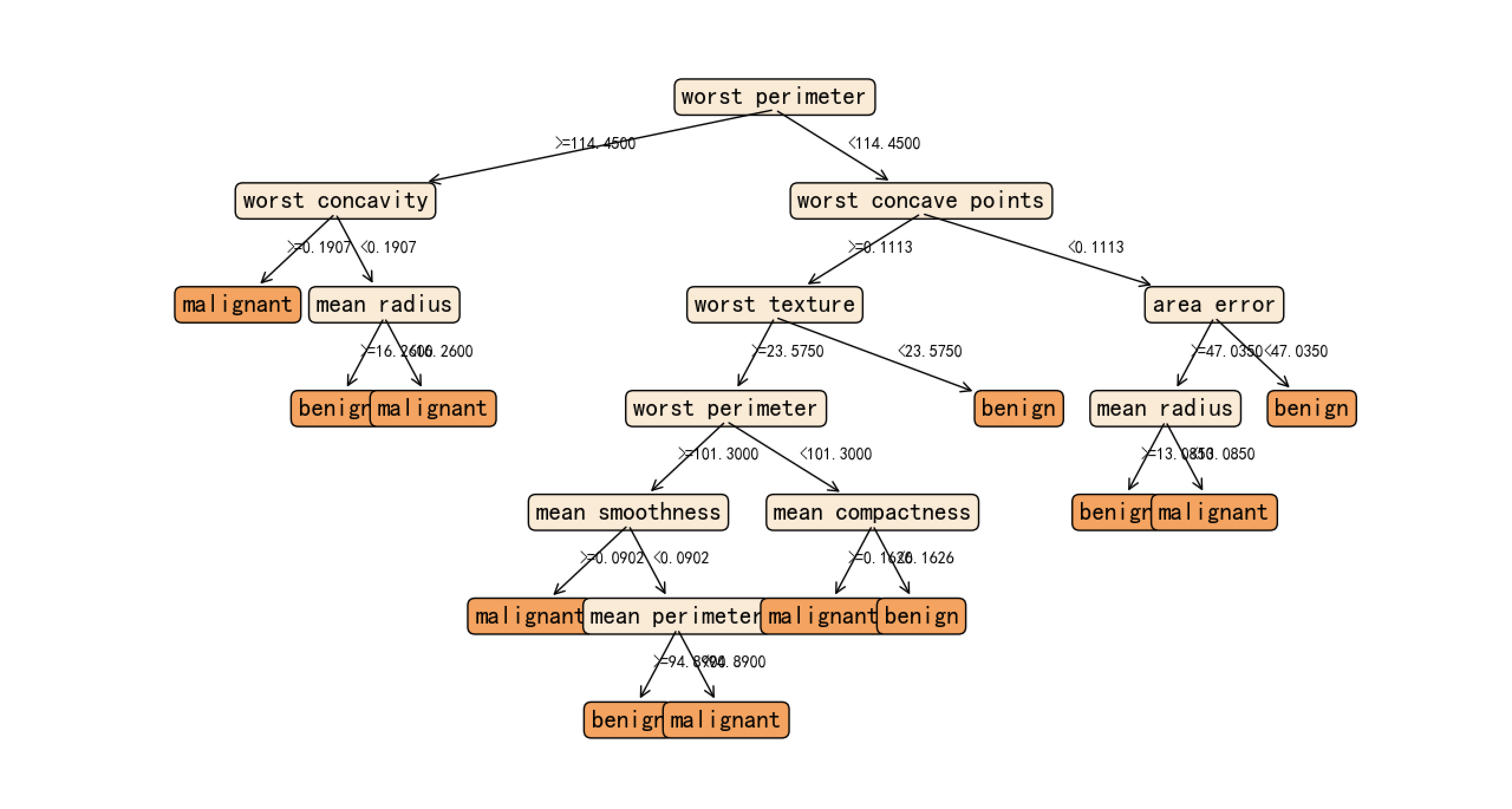
图表 5 iris数据集正确率



图表 6 wine数据集



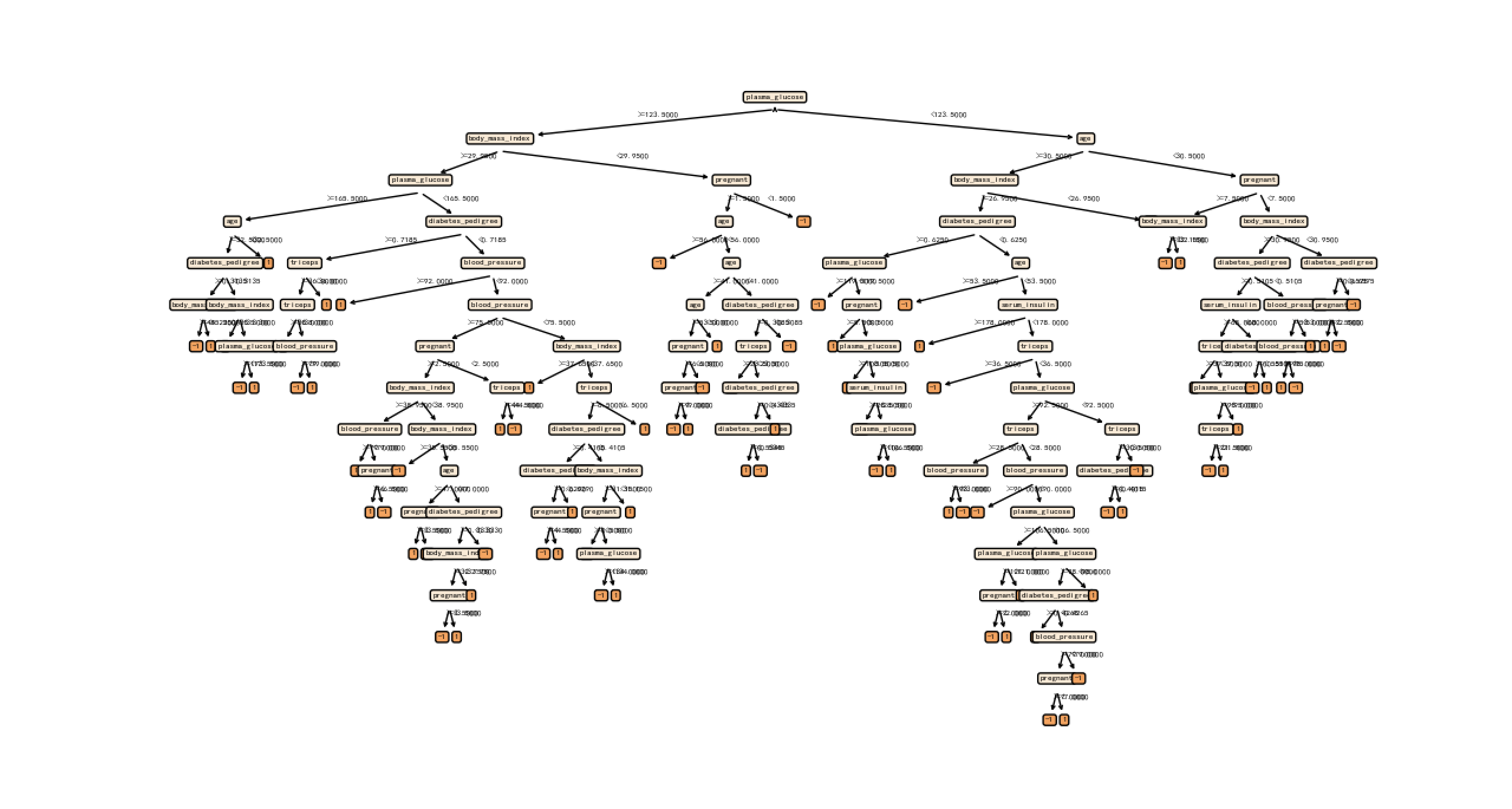
图表 7 wine数据集正确率



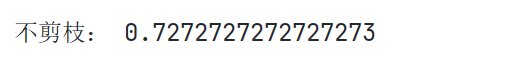
图表 8 breast\_cancer数据集



图表 9 breast\_cancer数据集正确率



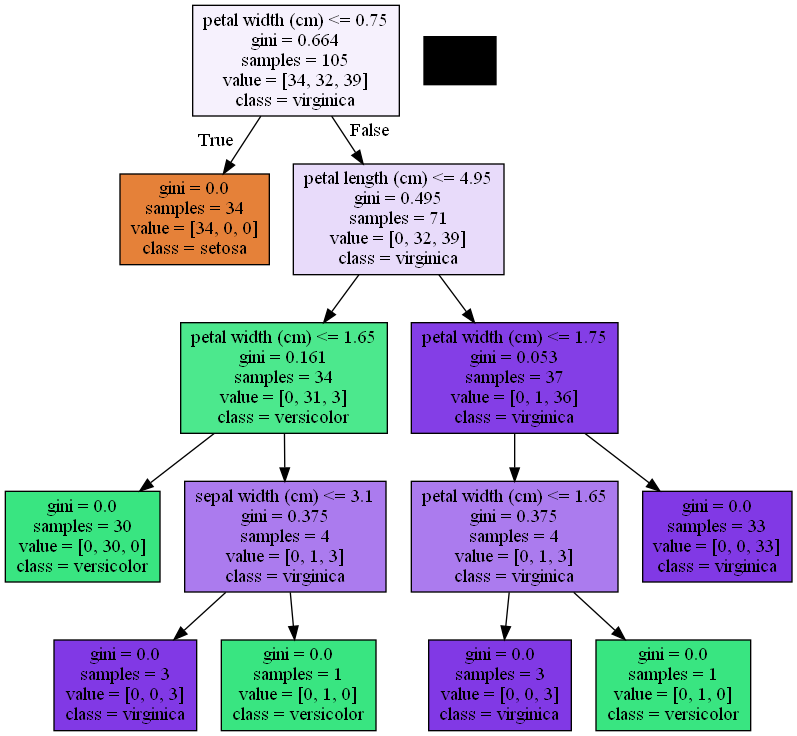
图表 10 diabetes数据集



图表 11 diabetes数据集正确率

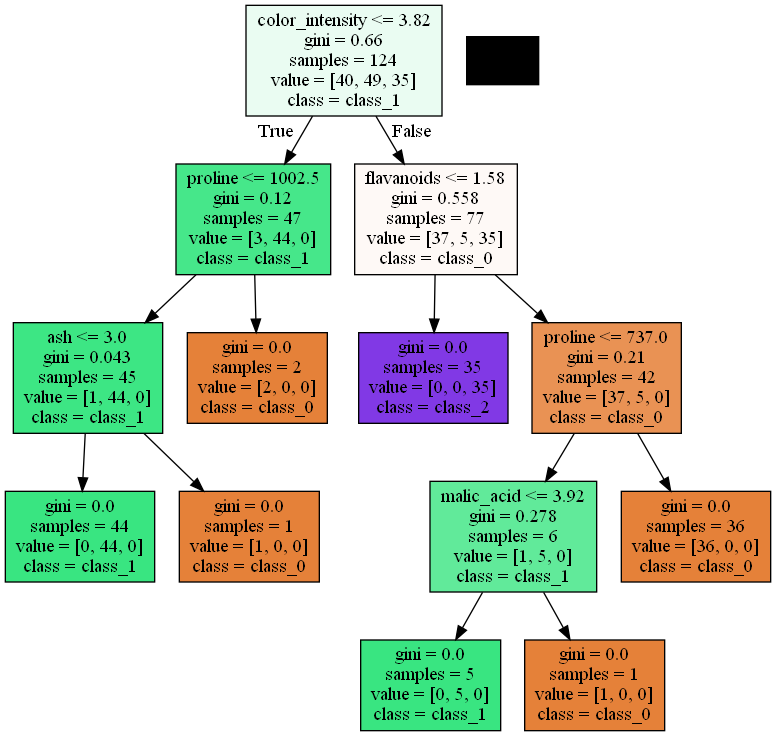
对于基尼指数：

未剪枝：



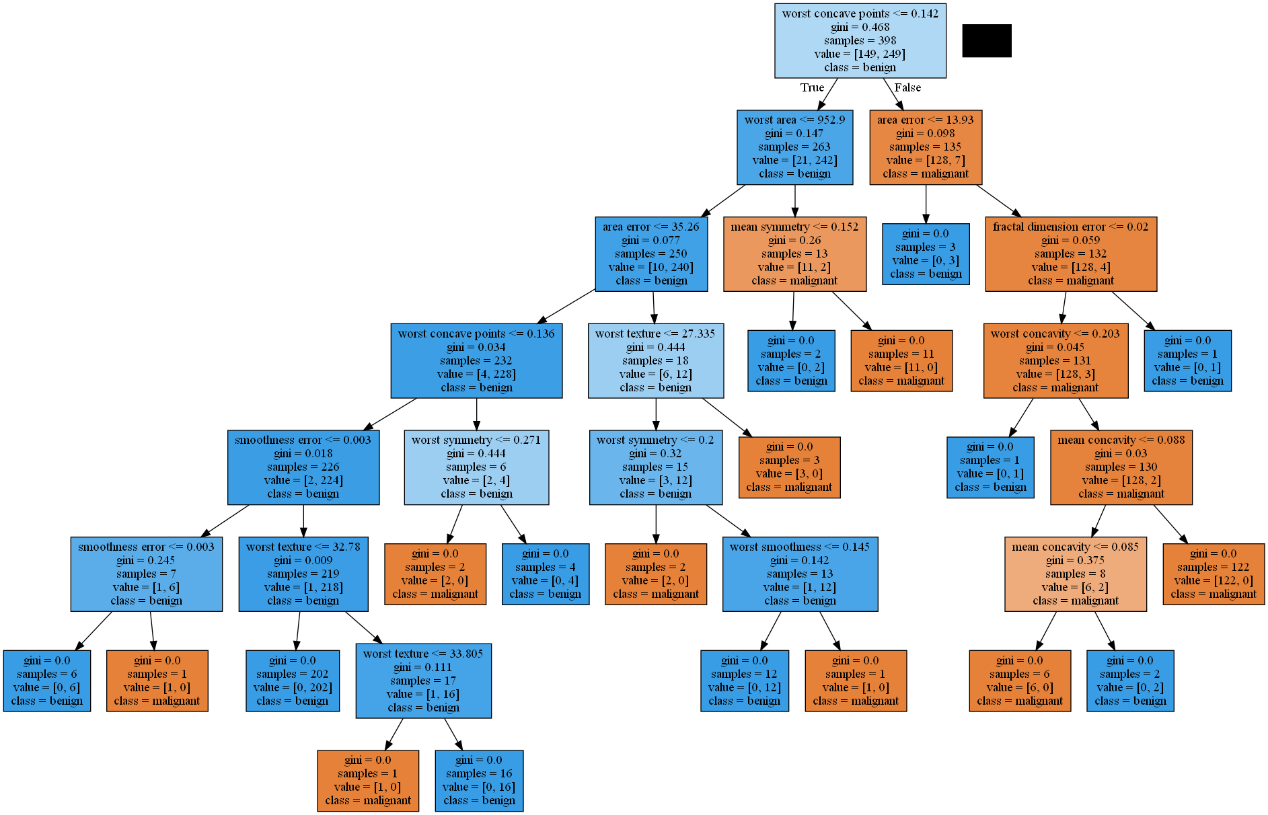


图表 12 iris数据集



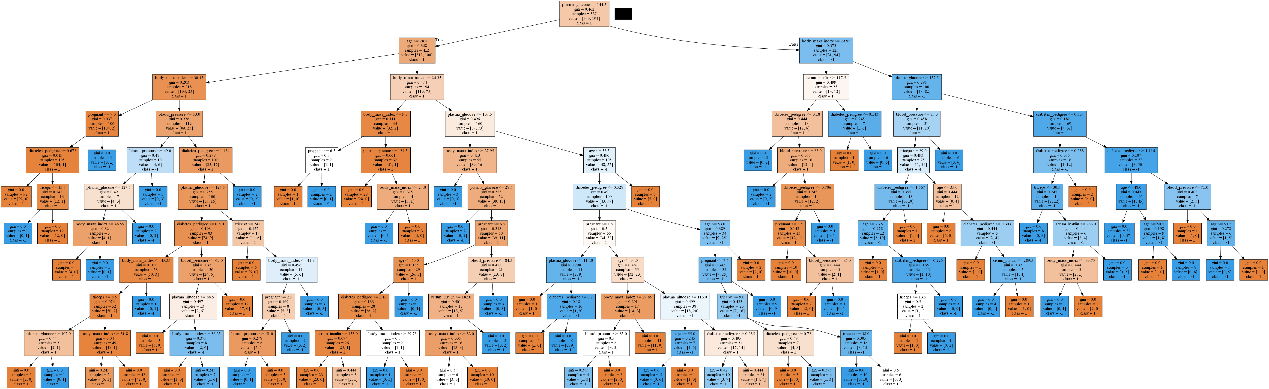


图表 13wine数据集





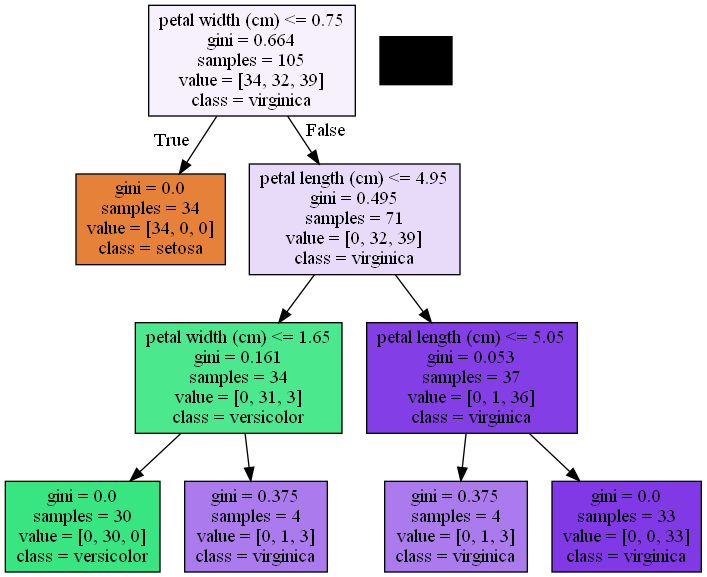
图表 14 breast\_cancer数据集





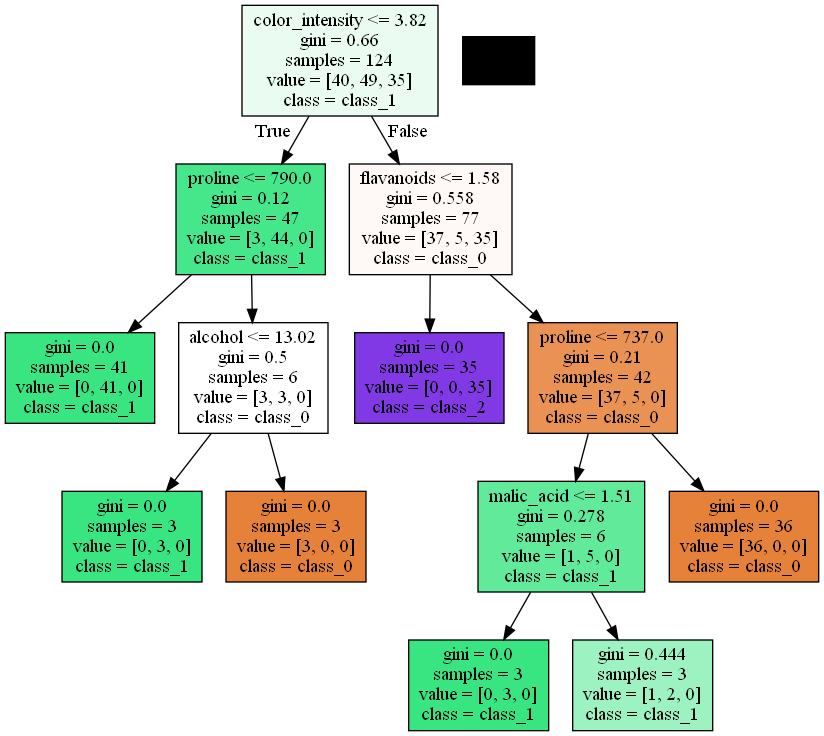
图表 15 breast\_cancer数据集

预剪枝：



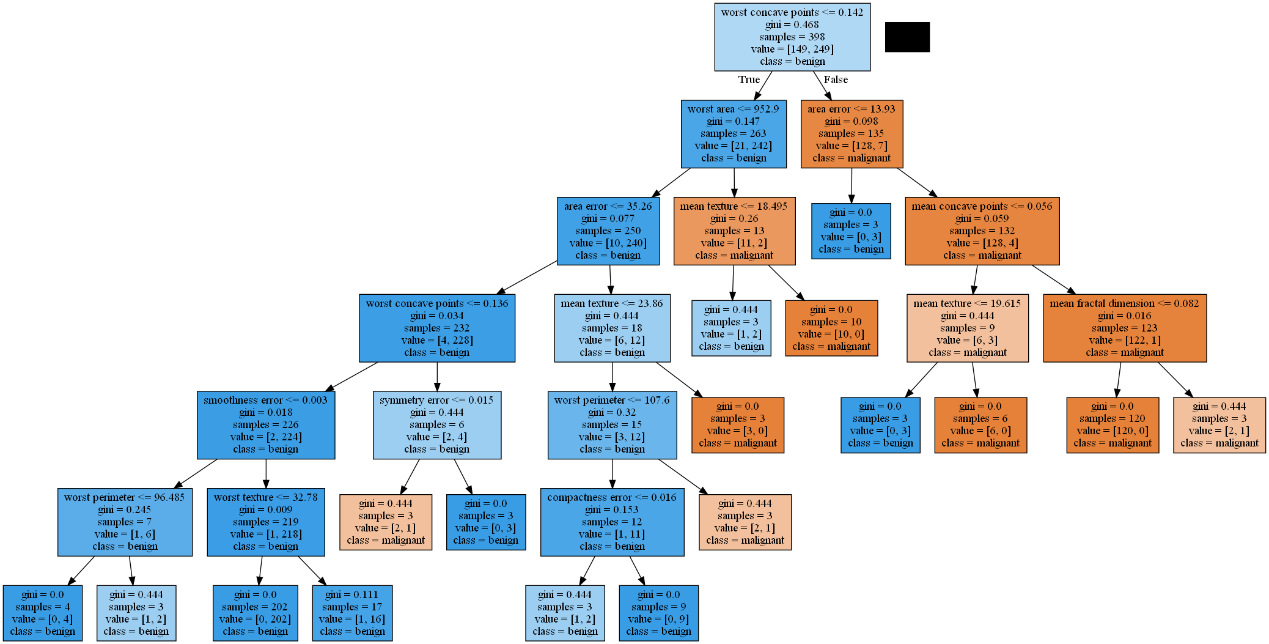


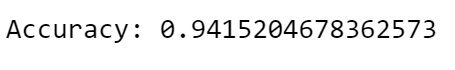
图表 16 预剪枝后的iris数据集



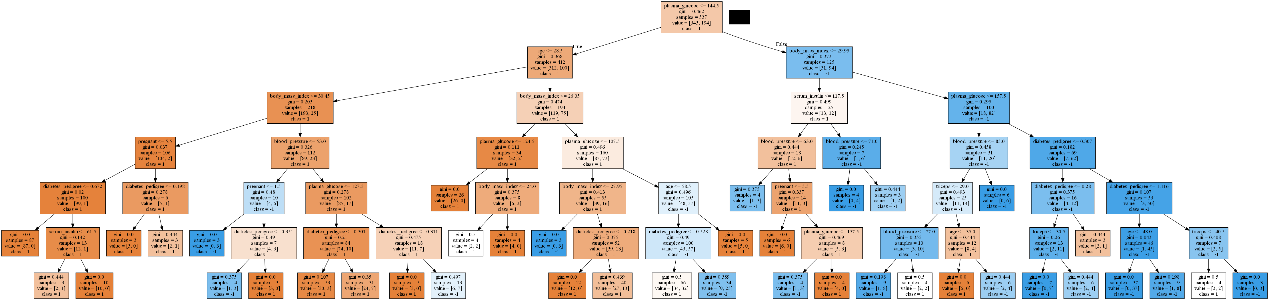


图表 17 预剪枝后的wine数据集





图表 18预剪枝后的bresat\_cancer数据集





图表 19 预剪枝后的diabetes数据集

可以画出如下表格:

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

## 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，没有自己实现后剪枝的代码，也没有依靠对率回归进行划分。但是对率回归本质上和信息增益是一样的，都是为了找出最好的划分点，来确保纯度最高。

## 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

# [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。

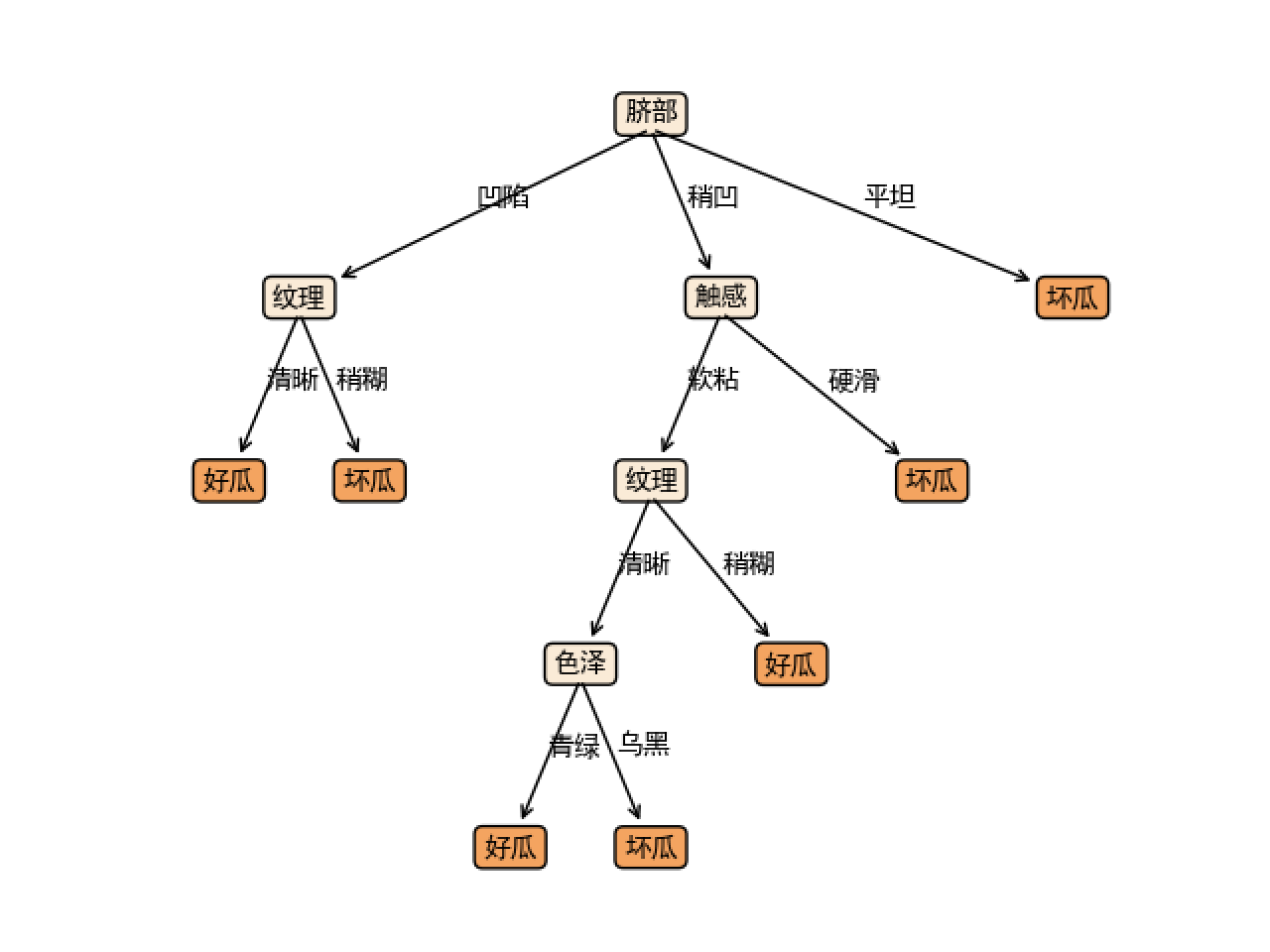
## 编程题目理解

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

## 算法设计思路

## 实验流程、测试结果及分析



## 代码结构，核心代码简要分析

## 本次实验解决的主要问题，主要收获

## 编码及内容撰写中的参考来源

# [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。

## 编程题目理解

题目要求利用

## 决策树算法原理阐述

## 算法设计思路

## 实验流程、测试结果及分析

## 代码结构，核心代码简要分析

## 本次实验解决的主要问题，主要收获

## 编码及内容撰写中的参考来源