决策树大作业实验报告

杨庆龙 1901213306

[yangqinglong@pku.edu.cn](mailto:yangqinglong@pku.edu.cn)

# 实验要求

使用自己擅长的编程语言编写决策树生成与测试系统，对给出的四个数据集的训练集数据使用不同的参数构建决策树与决策森林，并对相应的测试集进行测试，比较不同参数对决策树系统的影响。

# 实验内容

## 决策树

决策树是一类被广泛使用的分类与回归算法，它通过对问题相关的特征进行一系列测试，并且利用这些测试的结果确定最终查询样本的属类（分类问题）或是预测值（回归任务）。据册数由根节点，中间节点，叶节点和分支节点构成，每一个非叶节点均对应着对一个特征的测试，被测试特征拥有的可能取值数目决定了其拥有的向下分支的数目。每一个叶节点都对应了一种预测结果。

## 增益计算算法

使用ID3算法构建决策树，并比较信息增益(Information Gain: IG)和信息增益比例(Information Gain Ratio: IGR)对构建出的决策树的影响。

### 信息增益

信息增益是一种对于某个特征所包含的信息量的度量。具体而言，某个特征的信息增益等于原数据的熵减去该特征进行测试并根据测试结果分割原数据集后各个子数据的熵的加权求和，公式如下所示。

### 信息增益比例

为减少更多取值的特征的属性的影响，后续的研究提出使用IGR进行特征选取，具体公式如下所示。

## 剪枝

决策树构建的过程中递归地划分了数据集的特性，会导致部分叶节点受噪声影响过大。为解决该问题，决策树剪枝成为广泛使用以防止模型过拟合，剪枝后的决策树不能完美拟合训练数据，但能滤除部分噪声样本，从而提供模型的泛化能力。

### 预剪枝

预剪枝是较为简单的剪枝方式，即在构建决策树的过程中及早停止决策树的构建过程，深度和叶节点包含的最少样本数是最常见的剪枝标准。

### 后剪枝

在后剪枝策略中，首选依据标准决策树的构建算法构建一棵未剪枝的决策树。随后从完整决策树的叶节点开始，使用从低向上的方法进行剪枝，剪枝过程需要使用测试集来决定一个节点是否应该被删除。

## 模型集成

只使用一棵决策树会导致模型的泛化能力较差，若使用同一数据集按照某种规则来产生一系列不同的模型来实现集体决策，并提升最终模型的泛化能力，以实现三个臭皮匠顶一个诸葛亮的效果。

### Boosting

给原数据集的样本赋予不同的权重，再使用带有权重的数据生成一个带有重复数据的数据集，每个样本重复的次数正比于它的权重，即权重越大，重复次数越多。构建出新决策树后，再依据模型错误率计算出树的权重，同时更新每一个样本的权重，为下一次模型构建做准备。重复以上过程直到森林中的决策树数目达到预设值。最终决策时，所有的树都计算出结果，并依据权重进行表决，权重最大的结果即为决策结果。

### Bagging

与Boosting不同，Bagging中每一个样本出现的概率都是相同的，因此每次构建树所用的数据集都可看作原有数据集的局部特征。最终决策时，所有树都进行判决，并按照多数优先的原则进行投票表决，得票最高的结果即为判决结果。

# 程序设计与实现

## 语言选择

考虑到构建决策树的过程，需要将当前节点的数据集分割后构建子节点，且子节点之间没有关系，为减少构建所花的时间，在构建时使用多线程是较好的选择。在权衡单进程的运行性能和多进程的易用性后，选择使用Golang作为开发语言。

Golang是一种编译型语言，编译后的运行速度仅比C语言慢3到5倍，但提供了goroutine作为协程开发的原生手段，在并行任务开发上有非常大的优势，非常适合我们的使用场景。

## 结构体说明

重要的结构体包括数据样本适配器Adapter，决策树中的节点泛型Node，决策树Tree和决策森林Forest四个，其中Node为泛型接口，其具体实现共有中间节点JudgeNode和叶节点LeafNode两种。

### Adapter

该结构体位于adapter包内，具体定义如图1所示。其中，Name字段为全局唯一的字符串，以唯一地表示数据；Data是一个以string为Key，string为Value的Map数据结构，用以存储该数据的各项特征；Class表示了Data中哪一项数据为标注结果，若该数据没有进行标注，则为空字符串；UsedKey用于记录哪些特征被使用过，以避免字段被重复使用。此外，为避免使用过程中对数据的频繁复制导致性能损失，在该项目中所有调用或返回Adapter的地方使用的都是指针。



图1 Adapter结构体定义图



图1 Adapter结构体定义图

### Node

Node为决策树节点的泛型接口类，该泛型接口位于tree包中，具体定义如图2所示。其中JudgeOne和Judge函数均用于判断数据所示的类，区别为一个只对一条数据进行判断，而另一个为对一个数据列表进行判断。ErrorNum用于计算输入数据的错误个数，ErrorRate计算错误率。AddNode函数用于在节点的相应判断准测下添加子节点。ToString函数则是打印单个节点的序列化结果，主要用于Debug。Serialize函数用于树的序列化结果，用于持久化，此处不使用Golang自带的序列化函数是出于序列化结果的可读化考虑。Optimize函数主要用于后剪枝，若该子树认为自己可以被精简掉，会返回新的叶节点，否则返回本身；而GetUID则用于获取该节点的唯一标识，用于序列化等操作。

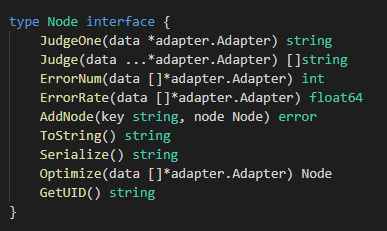


图2 Node接口定义图

### Tree

Tree代表一棵决策树，具体定义如图3所示。Root表示决策树的根节点；Weight表示该决策树的权重，用于不同的集成模型。

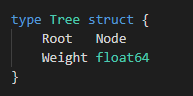


图3 Tree结构体定义图

### Forest

Forest是决策树进行判决的入口，该类位于tree包中，具体定义如图4所示。Type字段表示这是哪种森林，具体取值有single，boosting和bagging三种，分别对应一棵树，boosting集成和bagging集成；Trees则为该森林中所含有的所有树，在判断结果时，会对这些树进行遍历。



图4 Forest结构体定义图

## 帮助函数

帮助函数都位于utils包中，为程序运行过程中需要用到的各种函数，具体定义与功能如表1所示。

|  |  |
| --- | --- |
| 函数定义 | 用途 |
| IG(data []\*adapter.Adapter, key string) float64 | 计算使用key进行分类的IG |
| IGR(data []\*adapter.Adapter, key string) float64 | 计算使用key进行分类的IGR |
| H(data []\*adapter.Adapter, key string) float64 | 计算指定key的熵 |
| GroupBy(data []\*adapter.Adapter, key string) map[string][]\*adapter.Adapter | 依据指定key对数据进行分组  并返回分组结果 |
| GroupCount(data []\*adapter.Adapter, key string) map[string]int | 依据指定key对数据进行分组  只返回每组数据的数量 |
| P(data []\*adapter.Adapter, key string) map[string]float64 | 计算指定key里不同取值的比例 |
| Rem(data []\*adapter.Adapter, key string) float64 | 计算指定key的加权熵之和 |

表1 帮助函数说明表

## 载入器

### 输入数据格式

使用该工具载入训练数据之前，需要将训练数据另存为csv格式的数据，以方便阅读与处理。此外，对于以及被标注的数据，应当在标注列使用”列名:class”的格式现式地指出标记结果，以方便后续处理。而对于标号列，应当使用”列名:id”的方式指明此列为标号列，不作为数据特征使用。

### 代码实现

载入器位于loader包的dataloader.go文件内，由于相关操作并不需要使用类进行状态存储，所以只提供函数调用，不提供接口类。具体工作时，该函数会首先处理第一行，获取各个特征的名称以及id列和标记列，之后会按照逗号分隔的方式将其余内容与列名一一对应，并为每一行都创建一个Adapter作为一条数据记录保存至内存中。

## 构造器

构造器位于builder包内，主要使用的有两个构造器，分别为TreeBuilder和ForestBuilder分别用于使用既定参数构造单棵决策树和决策森林。

### TreeBuilder

TreeBuilder的结构体定义如图5所示。其中，MaxDepth和MinNode分别用于预剪枝时限制树的深度和分配到节点的数据数目，若不需要使用相关参数进行预剪枝，则将其赋值为-1即可。ScoreFunc是一个函数指针，在构建决策树时计算增益所用。

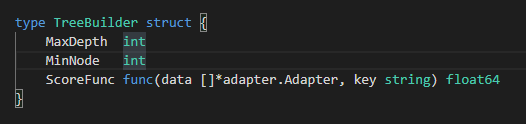


图5 TreeBuilder结构体定义图

### ForestBuilder

ForestBuilder的结构体定义如图6所示。其中，Builder参数表示构建单棵决策树时使用的TreeBuilder；Type表示决策森林的集成类型，可选值为single\_tree,boosting和bagging；Number为该集成模型中的最大决策树数量；Size为进行数据集抽样时，生成的数据集尺寸与原数据集的尺寸之比；AutoStop为是否提早结束决策树的构建，对于boosting类型的集成来说，当数据集的权重稳定后再进行抽样与构建其实意义不大，可以提前结束。

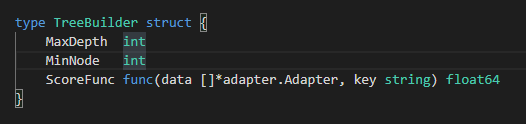


图5 TreeBuilder结构体定义图

### 调用流程

首先使用MakeTreeBuilder函数创建构造单棵决策树时所用的构造器，获得构造器后修改相关参数以满足需求；之后使用MakeForestBuilder函数创建集成模型所用的构造器，获得构造器后修改相关参数；之后调用ForestBuilder的BuildForest函数，并传入训练数据集，等待结果即可。

在TreeBuilder构建决策树的过程中，会先使用ScoreFunc的函数指针计算在该节点应当使用的最优判断依据，若该节点的所有数据的特征都一直，则以占比最大的class构建叶节点，返回给上层。找到最合适的特征后，会将数据依据该特征进行分组操作，并创建相应数量的GoRoutine，以并行化的方式构建子树。

## 抽样器

对于Boosting和Bagging集成操作，均需要使用抽样数据集进行训练，所以编写了DataSetProducer用于数据集的加权采样，该结构体位于Builder包内，具体定义如图6所示。Data为抽样前的数据，Weight为各个数据的权重。

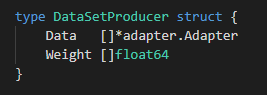


图6 DataSetProducer结构体定义图

## 持久化

为实现模型保存留备进一步的测试，在该项目中编写了saver包进行模型的保存和loader包进行模型的载入，相关功能均只提供函数，无需构建结构体。写功能的主要函数数均位于saver目录的saver.go文件中。读功能的主要函数均位于loader目录中。

### 持久化格式

持久化格式与内容定义如表2和表3所示，其中叶节点的格式如表4所示，非叶节点的格式如表5所示。

|  |
| --- |
| 集成模型的类型 |
| 树1的序列化结果 |
| 树2的序列化结果 |
| …… |
| 树N的序列化结果 |

表2 集成模型的序列化结果

|  |
| --- |
| 树的权重 |
| 节点1的序列化结果 |
| 节点2的序列化结果 |
| …… |
| 节点N的序列化结果 |

表3树序列化结果

|  |  |  |
| --- | --- | --- |
| UID | leafnode | 分类结果 |

表4 叶节点序列化结果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| UID | judegnode | 选取特征 | 特征1 | 子节点1 UID | …… | 特征N | 子节点N UID |

表5 非叶节点序列化结果

## 可视化

为方便Debug与展示，在saver包的dotsaver.go文件中，还提供了将结果保存为graphviz软件所能识别的dot文件格式。具体的工具安装与文件定义参见该软件的官网<http://www.graphviz.org/> ，样例结果如图7所示。由于节点的UID生成需要使用时间作为种子生成随机字符串，而windows上的时间戳只到毫秒级，所以在Windows上的可视化会出现问题。

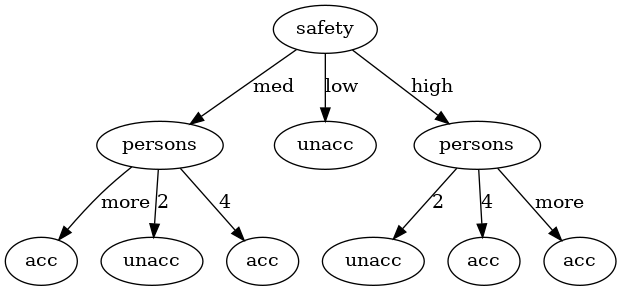


图7 可视化样例图

## 使用方法

在Linux下使用go build命令编译该项目后，会得到名为ID3Tree的可执行文件(Windows下为ID3Tree.exe)，该软件所支持的参数如表6所示。相对比较完整的样例命令如表7所示。

|  |  |
| --- | --- |
| 参数 | 说明 |
| help | 打印帮助信息 |
| autostop | boost集成的运算过程中  错误率稳定后自动停止 |
| build | 训练集数据路径 |
| depth | 早剪枝深度 |
| dot | 保存为dot文件的路径 |
| forest | 模型集成的种类 |
| func | 信息增益函数名 |
| leafsize | 最小叶节点数据数 |
| load | 载入模型的路径 |
| optimize | 用于优化的数据集路径 |
| output | 分类结果的输出路径 |
| run | 运行分类结果的测试集数据 |
| save | 模型的保存路径 |
| setsize | 集成模型的抽样比例 |
| trees | 集成模型的最大树数量 |
| withUID | 可视化文件中是否附带UID |

表6 工具参数说明表

此外，由于部分数据集没有提供现成的训练集和数据集，所以还在tools文件夹下编写了数据抽样工具，用于按照一定比例抽样生成训练集和测试集，可以使用的参数如表8所示。

|  |  |
| --- | --- |
| 命令 | 说明 |
| ./ID3Tree –build car.train -save car.model -run car.test | 使用car.train训练基本模型，模型结果输出到car.model文件，并使用car.test进行测试 |
| ./ID3Tree -load car.model -dot car.dot | 载入car.model并将以dot文件的格式保存到car.dot文件中 |

表7 基本命令说明表

# 实验结果与分析

## 基本功能

使用编写好的工具，运行以下命令，使用默认参数基于monks1训练集构建一棵决策树，并使用monks1测试集进行正确率测试，可得如图8所示的结果。将该树可视化后，可得如图9所示的结构。

从结果可以看到，该工具基本功能完整，具备使用默认参数构建决策树和进行测试的能力。

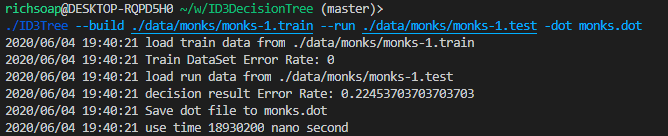


图8 运行结果图



图9 monks1决策树示意图

## 不同增益算法的比较

除增益算法外，其余参数均使用默认参数，基于不同数据集构建出不同的决策树，并对相应测试集进行测试后，可得结果如表8所示。分析结果后可知，对于前三类数据集，由于其特征数量并不是很多，所以使用IG算法足以构建出较好的决策树模型，使用IGR算法反而会影响结果的正确性。对于大豆数据，由于其特征众多，容易出现分支纯度过高的现象，所以使用IGR算法会比较好。

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集名称 | 增益算法 | 训练集错误率 | 测试集错误率 |
| breast-cancer | IG | 0.024038 | 0.392405 |
| IGR | 0.024038 | 0.392405 |
| car | IG | 0 | 0.08216 |
| IGR | 0 | 0.08216 |
| monks1 | IG | 0 | 0.238426 |
| IGR | 0 | 0.233796 |
| monks2 | IG | 0 | 0.337963 |
| IGR | 0 | 0.333333 |
| monks3 | IG | 0 | 0.057737 |
| IGR | 0 | 0.099307 |
| soybean | IG | 0 | 0.132979 |
| IGR | 0 | 0.077128 |

表8 默认参数增益算法比较表

## 预剪枝

使用IG算法计算信息增益，使用2到11的最大深度与1到15的最小节点数据数构建决策树，并对测试集进行测试后，可得如图10至图15所示的错误率热度图。从该热度图中可以得知，对于该实验中用到的四种数据集，当深度太小或最小节点数据数过大的情况，均会导致分类粒度过大，进而导致决策模型在训练集与测试集均不能较好地完成分类任务。随着深度变大，节点数据下限变小，决策模型在训练集和测试集的表现均能得到一定提升，但当深度变得过大时，会出现过拟合现象，即模型在训练集上的错误率已经非常低，但在测试集上的错误率却反而变大，说明模型缺乏泛化能力。此外，还能从图中看到，不同模型在测试集上得到最优结果的参数并不相同，这说明，使用预剪枝的方法改善模型泛化能力时，应当测试多种参数组合，以获得在当前情况下最好的参数组合和最小的测试错误率。

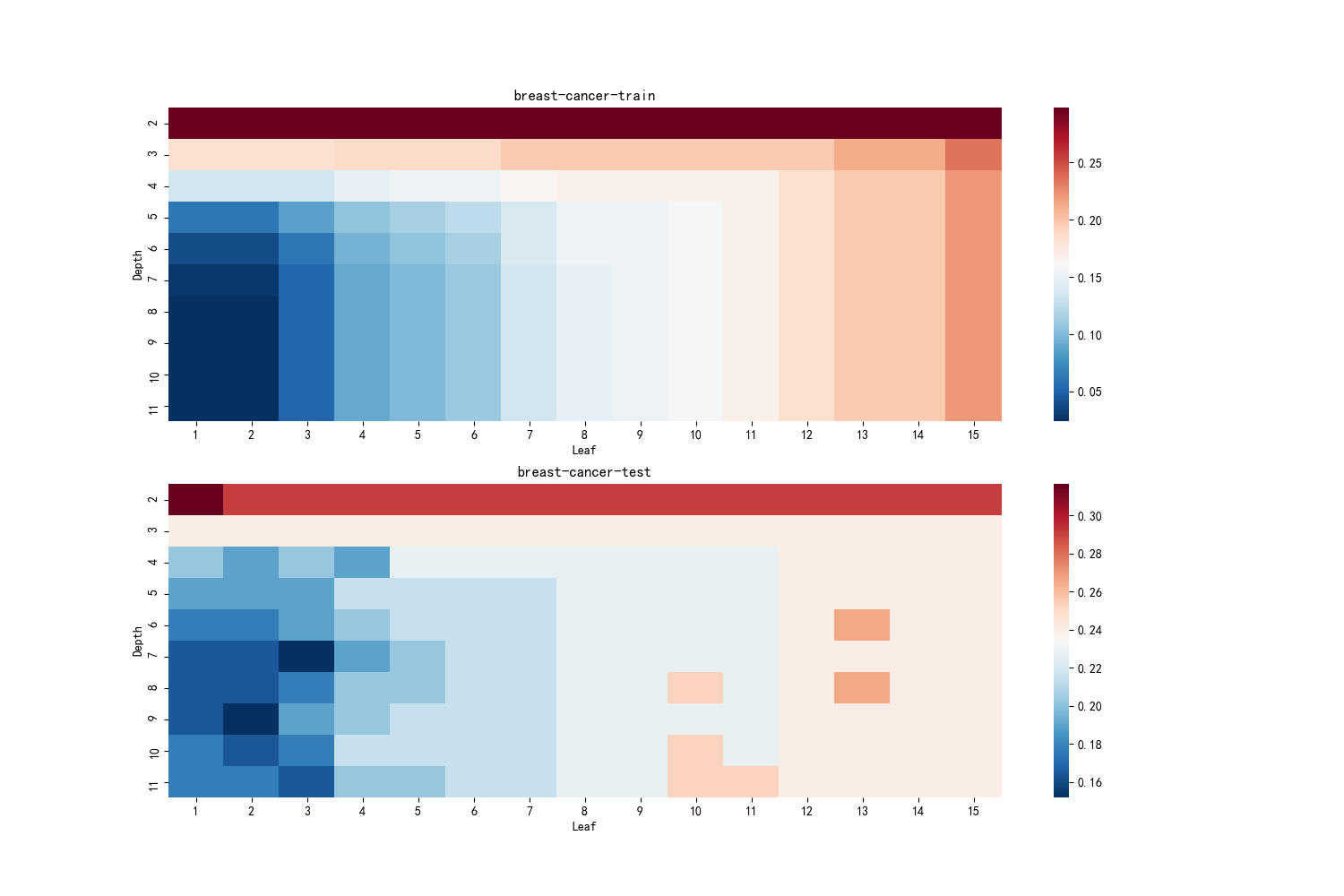


图10 breast-cancer预剪枝错误率热度图

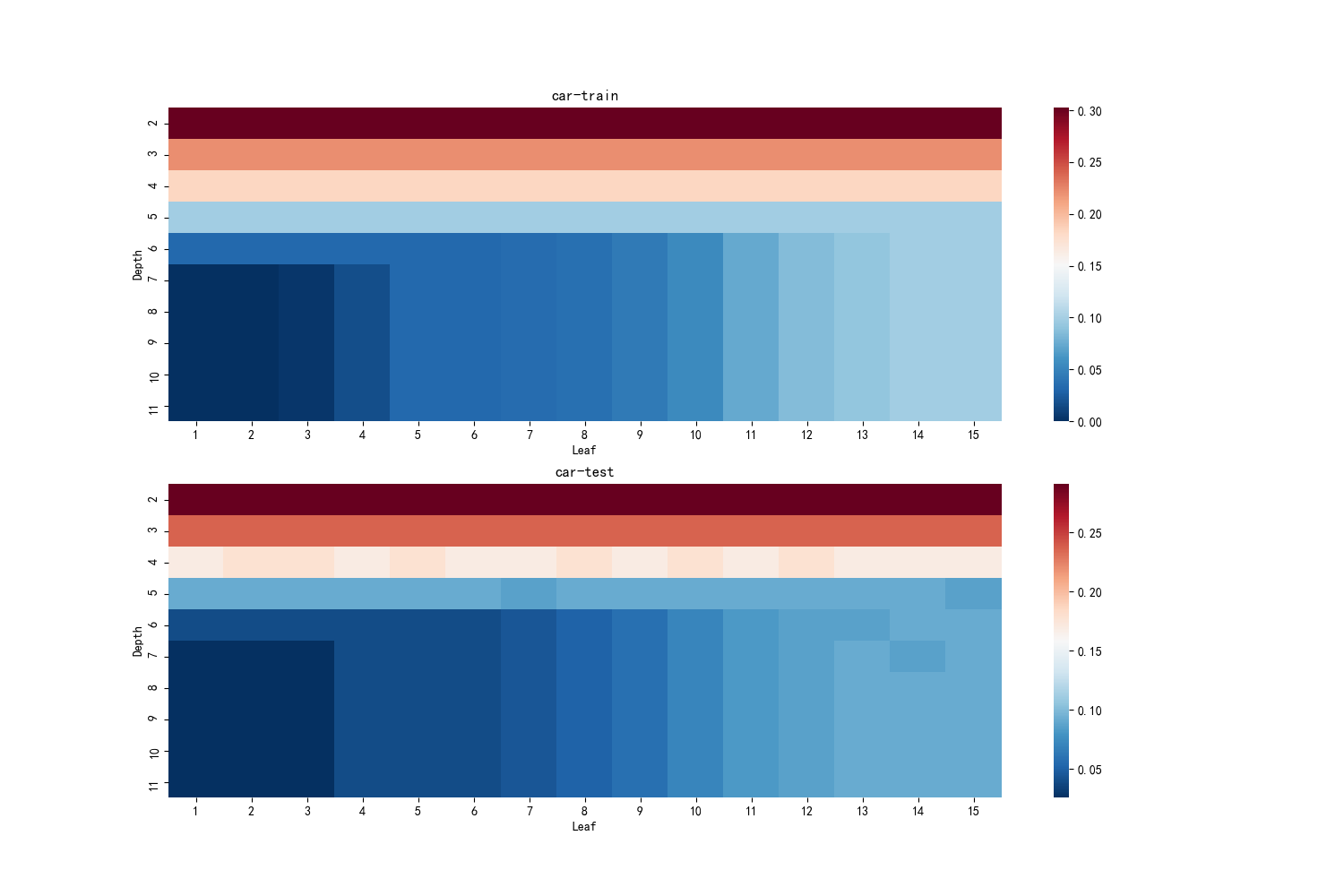


图11 car预剪枝错误率热度图

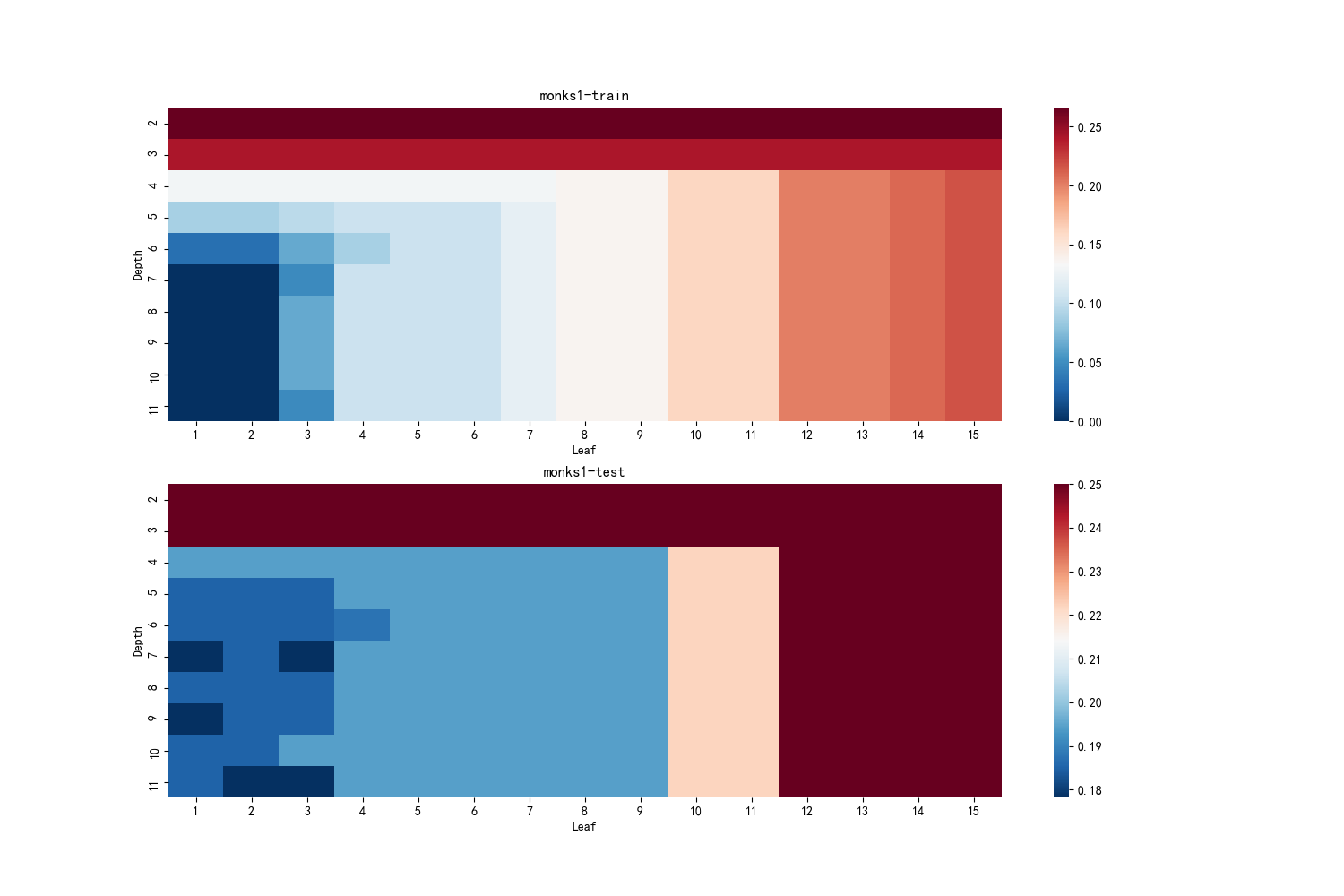


图12 monks1预剪枝错误率热度图

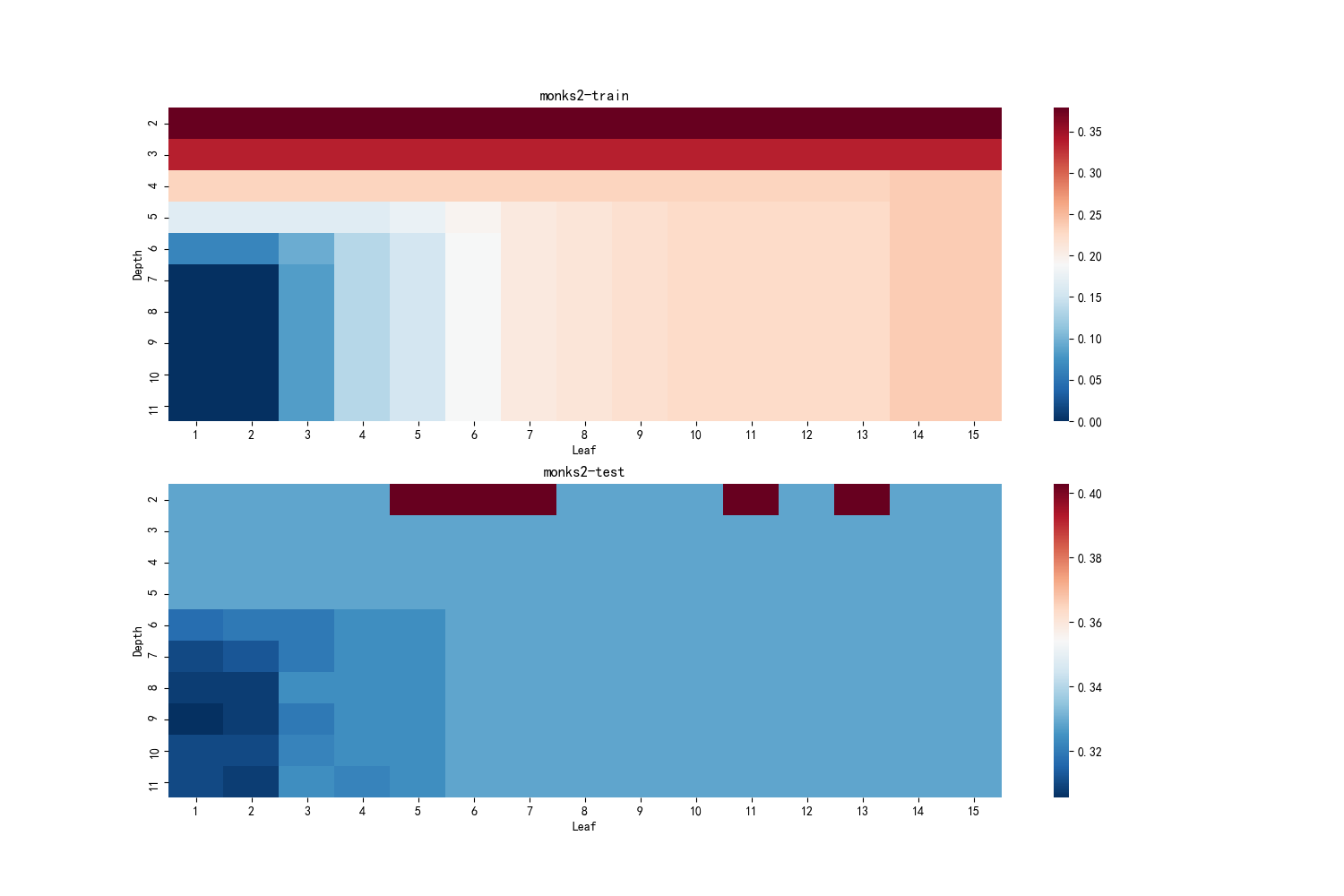


图13 monks2预剪枝错误率热度图

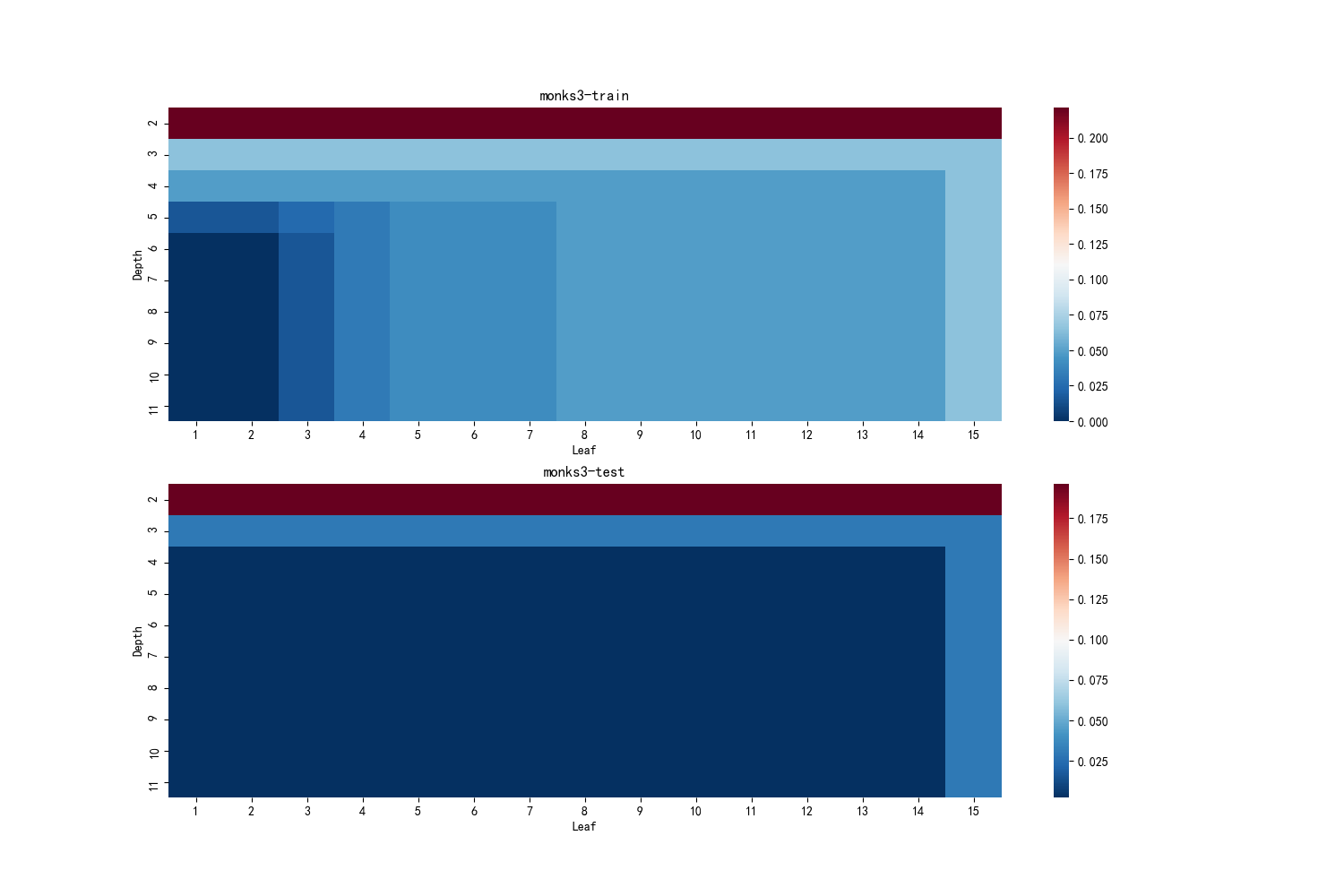


图14 monks3预剪枝错误率热度图

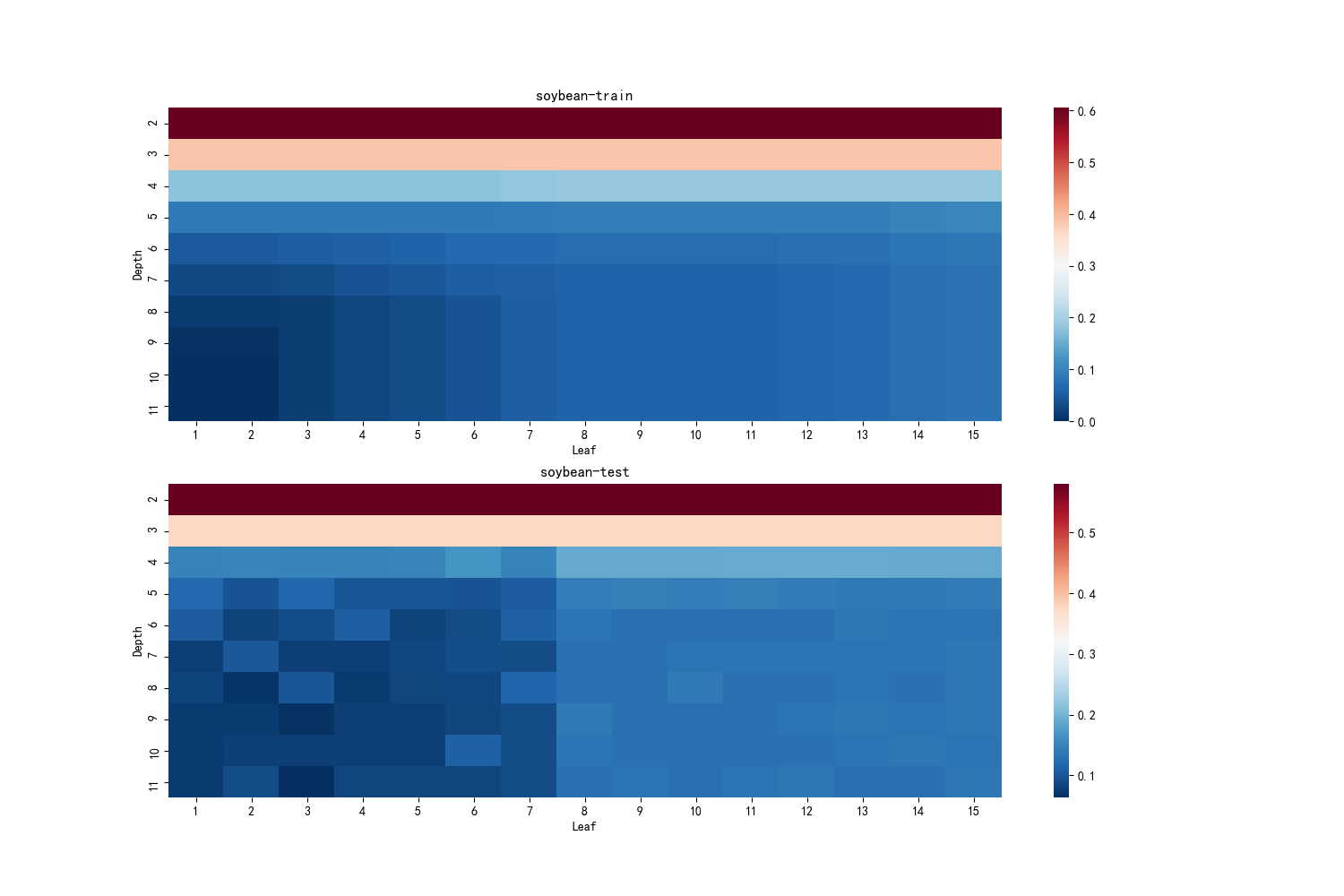


图15 soybean预剪枝错误率热度图

## 后剪枝

使用默认参数构建决策树后，再使用测试集进行后剪枝操作，在不同数据集的表现如表9所示。从表中可以看到，后剪枝策略的确能够有效地降低模型在测试集上的错误率，是一种看上去行之有效的方法，但考虑到模型进行测试时不应当受测试数据影响，所以具体的实用价值还有待进一步的讨论。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据集名称 | 算法 | 训练集错误率 | 优化前错误率 | 优化后错误率 |
| breast-cancer | IG | 0.024038 | 0.367089 | 0.164557 |
| IGR | 0.024038 | 0.392405 | 0.227848 |
| car | IG | 0 | 0.077465 | 0.025822 |
| IGR | 0 | 0.08216 | 0.023474 |
| monks1 | IG | 0 | 0.236111 | 0.185185 |
| IGR | 0 | 0.247685 | 0.1875 |
| monks2 | IG | 0 | 0.340278 | 0.30787 |
| IGR | 0 | 0.335648 | 0.289352 |
| monks3 | IG | 0 | 0.062356 | 0.002309 |
| IGR | 0 | 0.099307 | 0.030023 |
| soybean | IG | 0 | 0.12766 | 0.071809 |
| IGR | 0 | 0.082447 | 0.045213 |

表9 后剪枝优化效果表

## 集成模型

集成模型主要有两方面的用途，首先是使用集体的决策结果改善模型的抗噪声能力，其次是将一棵很深的决策树改为一组较浅的决策树，在错误率没有明显变差的前提下，通过并行的方式提升模型的计算速度。在集成模型中，主要的参数有以下三个，采样数据集尺寸，深度与树的棵数，因此，下文将从这三个参数入手，试图总结出集成模型的一些规律，此外，由于不同数据集的特征大不相同，所以会对不同数据集进行测试与分析。

### breast-cancer数据集

该数据集是用于乳腺癌预测与分析的数据集，囊括了多项病理特征。从之前单棵树的预测结果来看，该数据集在后剪枝优化之前的最优错误率为0.2532，后剪枝优化之后为0.1519。

#### boosting

使用不同尺寸的采样数据集，树的深度以及树的数目进行测试后，可得如附录表1所示的错误率热度图。可以看到，随着采样数据集的尺寸增大，模型在训练集的错误率逐渐减少，且呈现出树的深度约深，效果越好的特点。然而，在测试集上却呈现出完全不同的特征，模型的最小错误率并不随采样数据集的尺寸变化而变化，而且，随着采样数据的尺寸变大，模型更容易出现过拟合的情况。此外，收集所有boosting模型的结果后可得，最优错误概率为0.1519，此时参数为树深11，15棵树，样本比例0.1，在没有经过后剪枝的情况下达到了和后剪枝一样的结果。

#### bagging

使用不同尺寸的采样数据集，树的深度以及树的数目进行测试后，可得如附录表2所示的错误率热度图。可以看到，随着采样数据集的尺寸增大，模型在训练集的错误率逐渐减少，且呈现出低错误率模型集中在一起的特点。由于bagging是多数投票表决，所以当树变多之后反而会出现无意义的表决变多的情况，进而影响系统正确率。随着采样数据集的尺寸变大，单棵树能学习到的特征就越多，所以这个较为良好的模型集群会随着数据集尺寸变大而向上移动。此外，测试集的热度图出现了和训练集热度图互补的情况，说明右上角的数据出现了过拟合现象，在测试集表现较好的模型主要集中在分界线附近，在进行模型训练时可以在这些范围内进行较为细致的参数调整。收集所有bagging模型的结果后可得，最优错误概率为0.1519，此时参数为深度4，13棵树，样本比例0.4.

### car数据集

car数据集提供了车辆的各项特征，用于判断车辆价值，是一组较为简单的数据集。在之前的测试中，单棵树预剪枝的情况下，最优错误率为0.0493，后剪枝后为0.0235.

#### boosting

该数据的boosting结果如附录表3所示。可以看到，该模型在训练集与测试集上均呈现树深比树的数目更加重要的特征。当测试集的热度图在样本比例达到0.7之后，可以明显地看到，不论树的数目有多少，均在深度为6时取得最优解。而训练集则会由于过拟合的问题随着树深的增加，错误率持续降低。Boosting下最优错误率为0.0423，此时参数为深度为6，28棵树，采样比例为2.

#### Bagging

该数据集的bagging结果如附录表4所示。可以看到，模型在训练集和测试集上均呈现右上角较好的特点，测试集的较优区域会出现由于过拟合导致的红点。随着采样比例的增大，单树能学到的特征越来越多，此时进行集体表决反而会影响最终结果的正确率，即出现一群诸葛亮讨论不出一个所以然的情况。Bagging的最优错误率为0.0516，此时参数为深度6，1棵数，采样比例1.4。

### monks数据集

monks是首个进行国际化学习算法比较的数据集，该数据集分为三部分分别具有不同的特征。

#### boosting

如附录表5至7所示，为monks三个数据集在不同参数下的结果，从图中可以看到，三个模型在训练集的表现和breast-cancer数据集变化差别不大，都呈现采样比例越大，树越多越深，错误率越小的特点。但测试集的表现却截然不同，monks1的测试集图片都比较凌乱，缺乏明显边界，说明相关模型均没有特别突出的地方，随机性占据的影响过大；monks2的测试集图片则在采样比例0.6左右的地方出现除右上角过拟合导致错误率过大外，其余参数均能获得较好结果的特点；monks3的测试集图片则大部分时候的错误率都比较低。初步认为这些特征与数据集本身的特征关系非常大。

#### bagging

如附录表8至10所示三个数据集的训练集表现差别不大，均呈现采样比例增大后错误率较低的模型都集中在右上角的情况。而训练集则呈现出相反的特征，说明此时出现了过拟合现象。

### soybean数据集

soybean提供了大量与大豆有关的特征，用于分辨大豆品质。

#### boosting

如附录表11所示，该数据集的boosting集成在测试集和训练集上具有高度的一致性，在所有测试到的采样比例下，均在深度较深，而树的数目又不至于太多的情况下获得较好的分类结果。由于该数据特征数目很多，所以在树深较浅的情况下表现得不太好。

#### bagging

bagging集成模型的结果如附录表12所示，该数据集的bagging集成特征与训练集一直，故不再赘述。

### 总结

将不同数据集的最优结果列在表10中可得，并不存在某种集成方式绝对优于另一种集成方式的情况，但总体来说，boosting的表现较好，bagging表现不太稳定。

|  |  |  |  |
| --- | --- | --- | --- |
| 数据集 | 集成方式 | 训练集错误率 | 测试集错误率 |
| breast-cancer | 单树预剪枝 | 0.221154 | 0.265823 |
| boosting | 0.293269 | 0.151899 |
| bagging | 0.230769 | 0.151899 |
| car | 单树预剪枝 | 0.033026 | 0.049296 |
| boosting | 0.036098 | 0.042254 |
| bagging | 0.043011 | 0.051643 |
| monks1 | 单树预剪枝 | 0 | 0.022685 |
| boosting | 0 | 0 |
| bagging | 0 | 0.027778 |
| monks2 | 单树预剪枝 | 0.378698 | 0.328704 |
| boosting | 0.18939 | 0.268519 |
| bagging | 0.207101 | 0.247685 |
| monks3 | 单树预剪枝 | 0.04918 | 0.002309 |
| boosting | 0.04918 | 0.002309 |
| bagging | 0.04918 | 0.002309 |
| soybean | 单树预剪枝 | 0.045603 | 0.058511 |
| boosting | 0.045603 | 0.045213 |
| bagging | 0.052117 | 0.045213 |

表10 最优结果比较图

# 项目链接

<https://github.com/richsoap/ID3DecisionTree>

# 附录

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表1 breast-cancer boosting错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表2 breast-cancer bagging错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表3 car boosting错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表4 car bagging错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表5 monks1 boosting错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表6 monks1 bagging错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表7 monks2 boosting错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表8 monks2 bagging错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表9 monks3 boosting错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表10 monks3 bagging错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表11 soybean boosting错误率比较表

|  |  |
| --- | --- |
| 0.2采样率训练错误率 | 0.2采样率测试错误率 |
| 0.5采样率训练错误率 | 0.5采样率测试错误率 |
| 1.1采样率训练错误率 | 1.1采样率测试错误率 |
| 1.9采样率训练错误率 | 1.9采样率测试错误率 |

附表12 soybean bagging错误率比较表