数据科学导论第 8 讲——决策树与集成学习

王小宁

中国传媒大学数据科学与智能媒体学院

2025 年 05 月 22 日

目录

分类

- 当因变量取离散值时，我们称之为分类。模仿回归的表达式，我们可以将分类问题写成

$$y = f(x) + \varepsilon$$

- 其中，f 是关于 x 的函数，往往被称为分类器（classifier），比如 Logistic 模型、决策树、随机森林和支持向量机等都是经典的分类器。

分类问题

- 基于树的方法是数据科学、机器学习里最常用的方法之一，本质上它是一种非参数方法，因为我们不需要事先对总体的分布作任何假设。
- 决策树的算法很多，首先以经典的 CART 算法开始，基于树的分类和回归方法。为了防止过拟合，常常是先生成一棵大树，然后对生成的树进行剪枝。
- 最后，介绍几种基于决策树的组合学习方法（ensemble learning），因为基于树的方法简单且易于解释，但存在方差大、不稳定的问题。而通过组合算法常常能降低方差，大大提高预测效果，虽然与此同时会损失一些解释性。

问题的提出

例子 1

- MASS 库包含的乳腺癌（biopsy）数据集。该数据集是从威斯康星麦迪逊大学医院（University of Wisconsin Hospital, Madison）获得的，共有 699 个观察和 11 个变量。
- 除了不纳入分析的病人 ID 外，包含了病人乳腺肿瘤切片的诊断信息（class），和 9 个与判别是否为恶性肿瘤相关的检验指标，如肿块厚度、细胞大小均匀性等，每个病人在这些细胞特征上都有一个 1~10 的得分，1 为接近良性，10 位接近病变，下表给出了该数据集的部分信息。
- 若想根据 9 个检验指标的得分预测病人是否为恶性肿瘤，该如何建模?

数据 1

Id	V1	V2	V3	V4	V5	V6	V7	V8	V9	Class
100025	5	1	1	1	2	1	3	1	1	benign
102945	5	4	4	5	7	10	3	2	1	benign
1015425	3	1	1	1	2	2	3	1	1	benign
				...	...					

例子 2

- ISLR 库中包含的座椅（Carseats）数据集。该数据集是一个包含 400 个不同商店的儿童座椅销售情况的模拟数据集，共有 11 个变量。
- 我们感兴趣的是该儿童座椅的销售量（Sales，单位：千），可能与此相关的变量有地区的收入水平（Income，单位：千美元）、公司在每个地区的广告预算（Advertising，单位：千美元）以及区域的人口规模（Population，单位：千）等。
- 下表给出的是该数据集的部分信息，假如想要了解各因素对儿童座椅的销售量是否有影响，该如何建模?

数据 2

Id	Sales	C-PRI	Income	Ad	POP	Price	Shelve	Age	EDU	U
1	9.5	138	73	11	276	120	Bad	42	17	Y
2	11.22	111	48	16	260	83	Good	65	10	Y
3	10.06	113	35	10	269	80	Medium	59	12	Y
				...	...					

决策树

决策树与组合学习

- 决策树方法最早产生于 20 世纪 60 年代，其中 CART（Classification and Regression Tree）算法是决策树最经典和最主要的算法。
- CART 算法是 Breiman 等在 1984 年提出来的一种非参数方法，它可以用于解决分类问题（预测定性变量，或者说当因变量是因子时），又可以用于回归问题（预测定量变量，或者说当因变量是连续变量时），分别称为分类树（Classification tree）和回归树（Regression tree）。

- CART 算法的基本思想是一种二分递归分割方法，在计算过程中充分利用二叉树，在一定的分割规则下将当前样本集分割为两个子样本集，使得生成的决策树的每个非叶节点都有两个分裂，这个过程又在子样本集上重复进行，直至无法再分成叶节点为止。

基本概念

❶ 节点

- 决策树包含三种节点，分别是根节点、中间节点、叶节点。
  - 建模之初，全部样本组成的节点，没有入边，只有出边，称为根节点（root node）；
  - 不再继续分裂的节点称为树的终端节点（terminal node）或叶节点（leaf node），没有出边，只有入边，它的个数决定了决策树的规模（size）和复杂程度；
  - 根节点和叶节点之外的都称作中间节点或内节点（internal node），既有入边，又有出边。

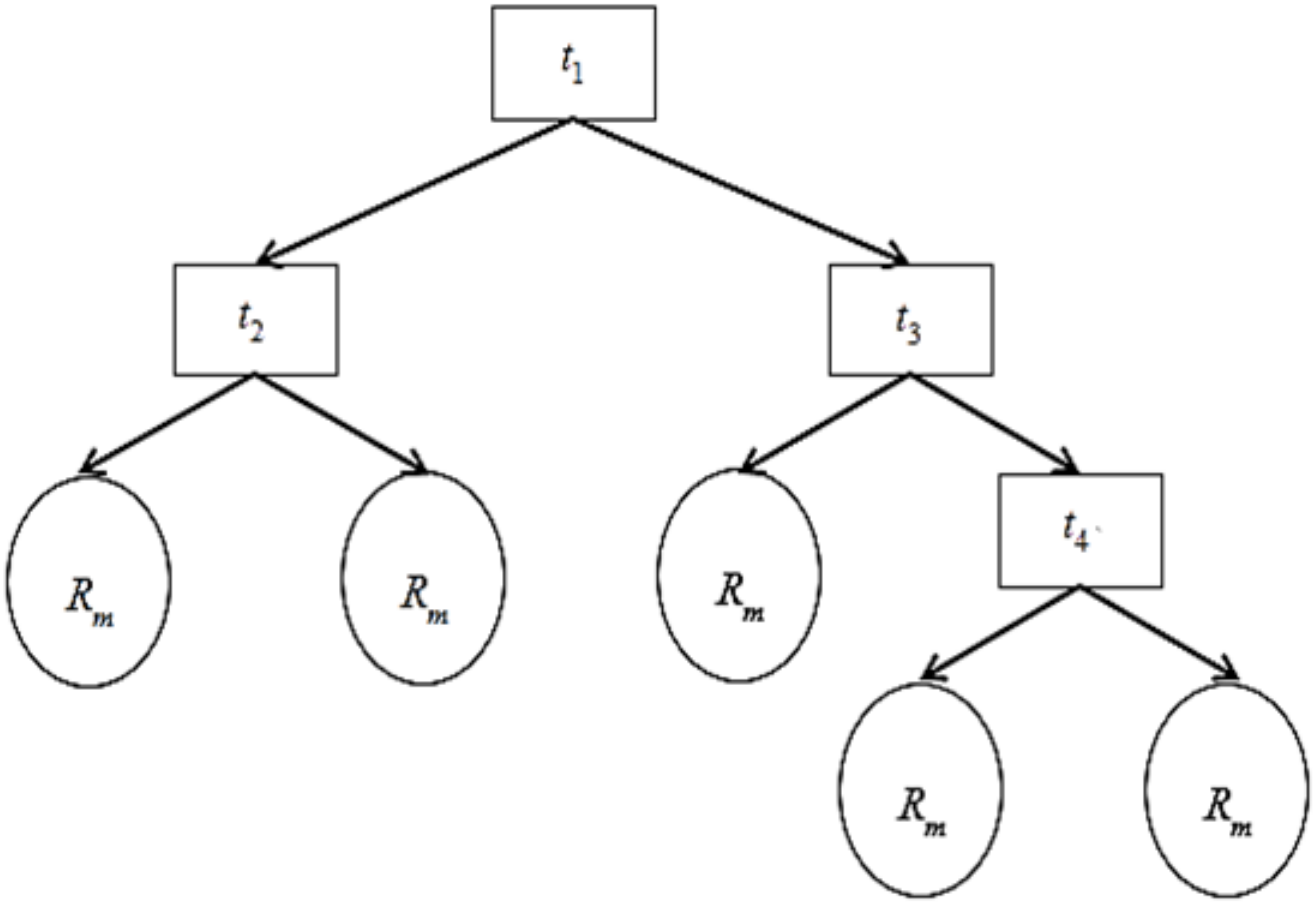
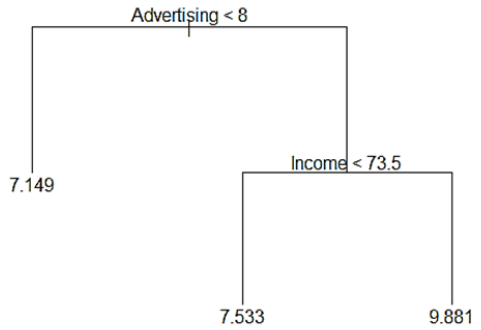


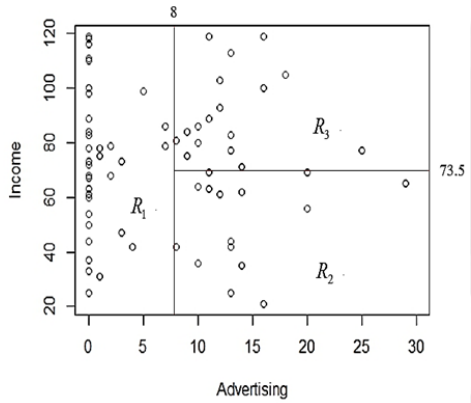
图 1: 决策树的分割过程

- 根节点在一定的分割规则下被分割成两个子节点，这个过程在子节点上重复进行，直至无法再分为叶节点为止。

决策树的建立

- 在 Carseats 数据集中，考虑自变量只有两维的情况，即只用地区的收入水平 Income 和公司在每个地区的广告预算 Advertising 来预测座椅的销售量 Sales，则可以生成一棵如下所示的含有三个叶节点的决策树。





- 这棵树是由从树顶端开始的一系列分裂规则构成的，根节点是 Advertising<8，则对于所有 Advertising<8 的座椅，其 Sales 都将被预测为 7.149，这是这棵树的第一个叶节点。
- 中间节点是 Income<73.5，则所有 Advertising>8 且 Income<73.5 的 Sales 都将被预测为 7.533，这是这棵树的第二个叶节点，最后，剩下的座椅的 Sales 都将被预测为 9.881，这是这棵树的第三个叶节点。
- 若将上述过程在二维的坐标轴中画出，则得到上图（右）。从这个图中就可以发现，决策树实际上就是对自变量空间的划分，且划分区域的数目就是叶节点的数目。

- 将上述过程推广至多维的情况，则可以将建立决策树的过程概括为以下两个步骤：

1. 将自变量空间（即 $X_1, X_2, \cdots, X_p$ 的所有可能取值构成的集合）分割成 J 个互不重叠的区域 $R_1, R_2, \cdots, R_J$
2. 对落入区域 R_j 的每个观测，都将其预测为 R_j 上训练集的响应值的简单算术平均。

- 因此，上图的第一个叶节点 7.149 就是满足 Advertising<8 的所有观测的 Sales 的平均值，且对任意给定的一个新的观测，若其落入此区域（即满足 Advertising<8），则我们也将把它的 Sales 预测为 7.149。

如何构建区域

- 理论上，可以将区域的形状作任意分割，但出于模型的简化和可解释性考虑，一般只将区域划分为高维矩形。
- 若要将自变量空间划分为个矩形区域的所有可能性都进行考虑，这在计算上是不可行的。
- 对区域的划分一般采用一种自上而下（top-down）、贪婪（greedy）的方法：递归二叉分裂（recursive binary splitting）。“自上而下”指的是它从树顶端开始依次分裂自变量空间，每个分裂点都产生两个新的分裂。“贪婪”意指在建立树的每一步中，最优分裂确定仅限于某一步进程，而不是针对全局取选择那些能够在未来进程中构建出更好的树的分裂点。

- 对每一个节点重复以上过程，寻找继续分割数据集的最优自变量和最优分裂点。此时被分割的不再是整个自变量空间，而是之前确定的两个区域之一。这一过程不断持续，直到符合某个停止准则：譬如，当叶节点包含的观测值个数低于某个最小值时，分裂停止。
- 如何确定最优的分裂方案？我们基于不纯度的减少来作为分裂准则，即通过最小化节点不纯度的减少来确定最优分裂变量和最优分裂点。对于分类树和回归树有不同的衡量节点不纯度的指标我们将在后面的分类树和回归树部分进行具体的描述。

剪枝

- 决策树虽能在训练集中取得良好的预测效果，却很有可能造成数据的过拟合，导致在测试集上效果不佳。为了防止决策树过度生长、出现过拟合现象，我们需要对决策树进行剪枝。
- 剪枝方法一般有事先剪枝（pre-prune）和事后剪枝（post-prune）两种。事先剪枝方法在建树过程中要求每个节点的分裂使得不纯度下降超过一定阈值。这种方法具有一定的短视，因为很有可能某一节点分裂不纯度的下降没超过阈值，但是其在后续节点分裂时不纯度会下降很多，而事先剪枝法则在前一节点就已经停止分裂了

复杂性剪枝

- 实际中，更多的是使用事后剪枝法，其中的代价复杂性剪枝（cost complexity pruning）是最常用的方法。
- 复杂性剪枝先让树尽情生长，得到 T_0, 然后再在 T_0 基础上进行修剪。设 $||$ 表示子树 T 的叶节点数目，n_m 表示叶节点 R_m 的样本量，则代价复杂性的剪枝法为:

$$C_\alpha(T) = \sum_{m=1}^{|T|} n_m Q_m(T) + \alpha|T|$$

- 其中每一个 α 的取值对应一颗子树 $T \subset T_0$, 我们的目标是确定 α 使得字数 T_α 最小化 $C_\alpha(T)$, 可以用交叉验证法确定 α，然后在整个数据集中找到与之对应的子树。

分类树

- 当因变量为定性变量时，可建立分类树模型。分类树是一种特殊的分类模型，是一种直接以树的形式表征的非循环图。它的建模过程就是自动选择分裂变量，以及根据这个变量进行分裂的条件。
- 假设数据 (x_i, y_i) 包含 p 个输入变量和一个分类型的因变量 $y \in 1, 2, \cdots, K$, 样本量为 n。现在我们想把数据分成 M 个区域（或节点），第 m 个区域 R_m 的样本量为 n_m，可建立如下的分类树算法。

分类树算法

1. 采用递归二叉分裂法在训练集中生成一棵分类树。最优的分裂方案是使M得个节点的不纯度减少达到最小，其中，衡量节点不纯度的指标Q_m有：

$$\text{错分率：} \quad \frac{1}{n_m}\sum_{x_i \in R_m} I\left(y_i \neq k(m)\right) = 1 - \hat{p}_{m,k(m)}$$

$$\text{Gini指数：} \quad \sum_{k \neq k'} \hat{p}_{m,k}\hat{p}_{m,k'} = \sum_{k=1}^{K} \hat{p}_{m,k}(1-\hat{p}_{m,k})$$

$$\text{熵：} \quad -\sum_{k=1}^{K} \hat{p}_{m,k}\log \hat{p}_{m,k}$$

2. 剪枝。
1）对大树进行代价复杂性剪枝，得到一系列最优子树，子树是α的函数；
2）利用K折交叉验证选择α，找出此α对应的子树即可。

图 2: 分类树算法

分类树算法

3. 预测。令 $\hat{p}_{m,k}=\frac{1}{n_m}\sum_{x_i\in R_m} I(y_i=k)$ 表示在节点m中第k类样本点的比例，则预测节点m的类别为：

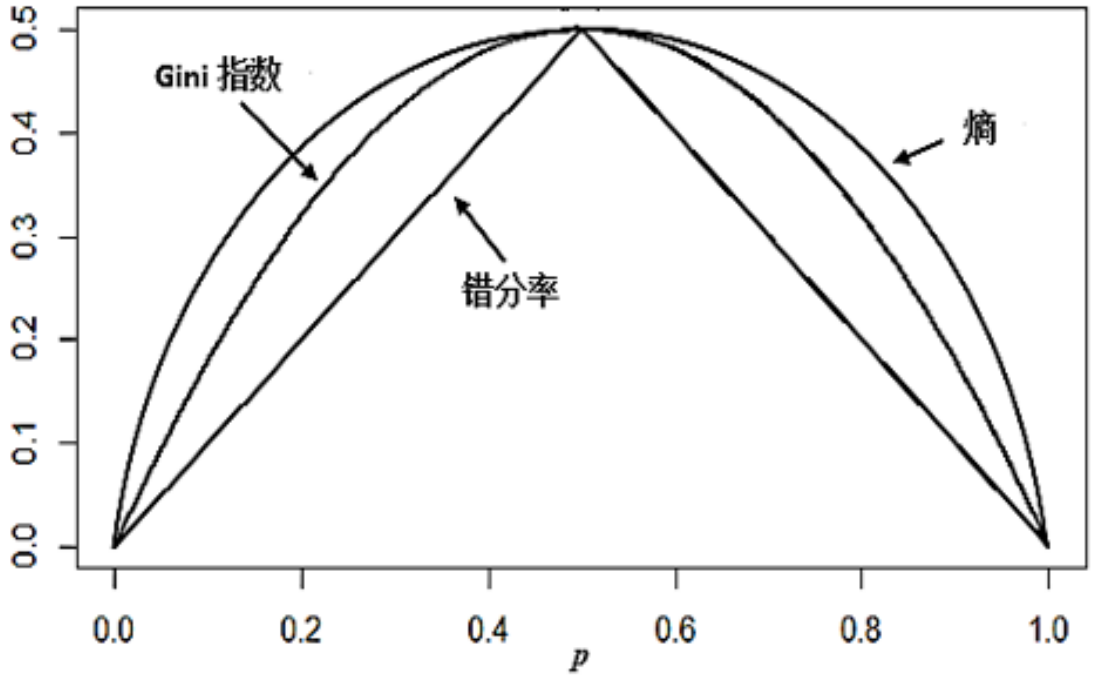
$$k(m)=\arg\max_k \hat{p}_{m,k}$$

即节点m中类别最多的一类。

其中，Gini指数衡量的是K个类别的总方差，若所有 $\hat{p}_{m,k}$ 的取值都接近0或1，则Gini指数会很小，因此它被视为衡量节点纯度的指标；类似的，对于熵而言，若所有 $\hat{p}_{m,k}$ 的取值都接近0或1，则熵接近于0。所以，第m个节点的纯度越高，Gini指数和熵的值都是越小的。事实上，Gini指数和熵在数值上是相当接近的，且相比于错分率，这两种方法对节点纯度更敏感，因此在实际中使用也更多。特别的，对于二分类的情况，若用p表示节点m包含其中一类的比例，则错分率、Gini指数和熵的取值分别为 $1-\max(p,1-p)$，$2p(1-p)$，$-p\log p-(1-p)\log(1-p)$

图 3: 分类树算法

三指标关系



衡量节点不纯度的三种指标

图 4: 关系图

分类图

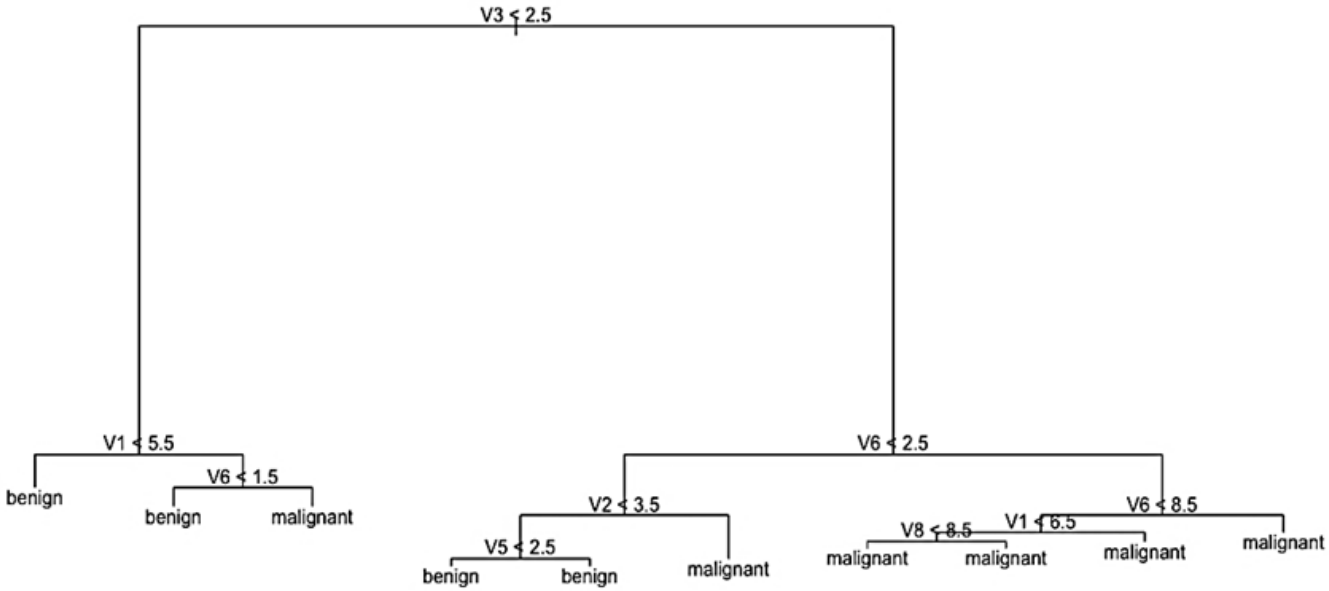

V3 < 2.5
V1 < 5.5
V6 < 2.5
benign
V6 < 1.5
benign
malignant
V2 < 3.5
V5 < 2.5
benign
benign
malignant
V6 < 8.5
V8 < 8.5
V1 < 6.5
malignant
malignant
malignant
malignant

图 5: 分类图

回归树

- 回归树和分类树的思想类似，只在分裂准则的确定上略有差异。
- 当因变量为连续型变量时，可建立回归树模型。对于分类树，将落在该叶节点的观测点的最大比例类别作为该叶节点预测值，而对于回归树，则是将落在该叶节点的观测点的平均值作为该叶节点预测值。

回归树算法

1. 采用递归二叉分裂法在训练集中生成一棵分类树。最优的分裂方案是使M得个节点的不纯度减少达到最小，其中，衡量节点不纯度的指标Q_m为内样本残差平方和的平均：

$$Q_m(T)=\frac{1}{n_m}\sum_{x_i\in R_m}(y_i-\hat{y}_{R_m})^2$$

2. 剪枝。
   1）对树进行代价复杂性剪枝，得到一系列最优子树，子树是α的函数；
   2）利用K折交叉验证选择α，找出此α对应的子树即可。

3. 预测。区域划分好后，可以确定某一给定预测数据所属的区域，并用这一区域的训练集的平均响应值对其进行预测：$\hat{y}_{R_m}=\frac{1}{n_m}\sum_{x_i\in R_m}y_i$

图 6: 回归树算法

回归树

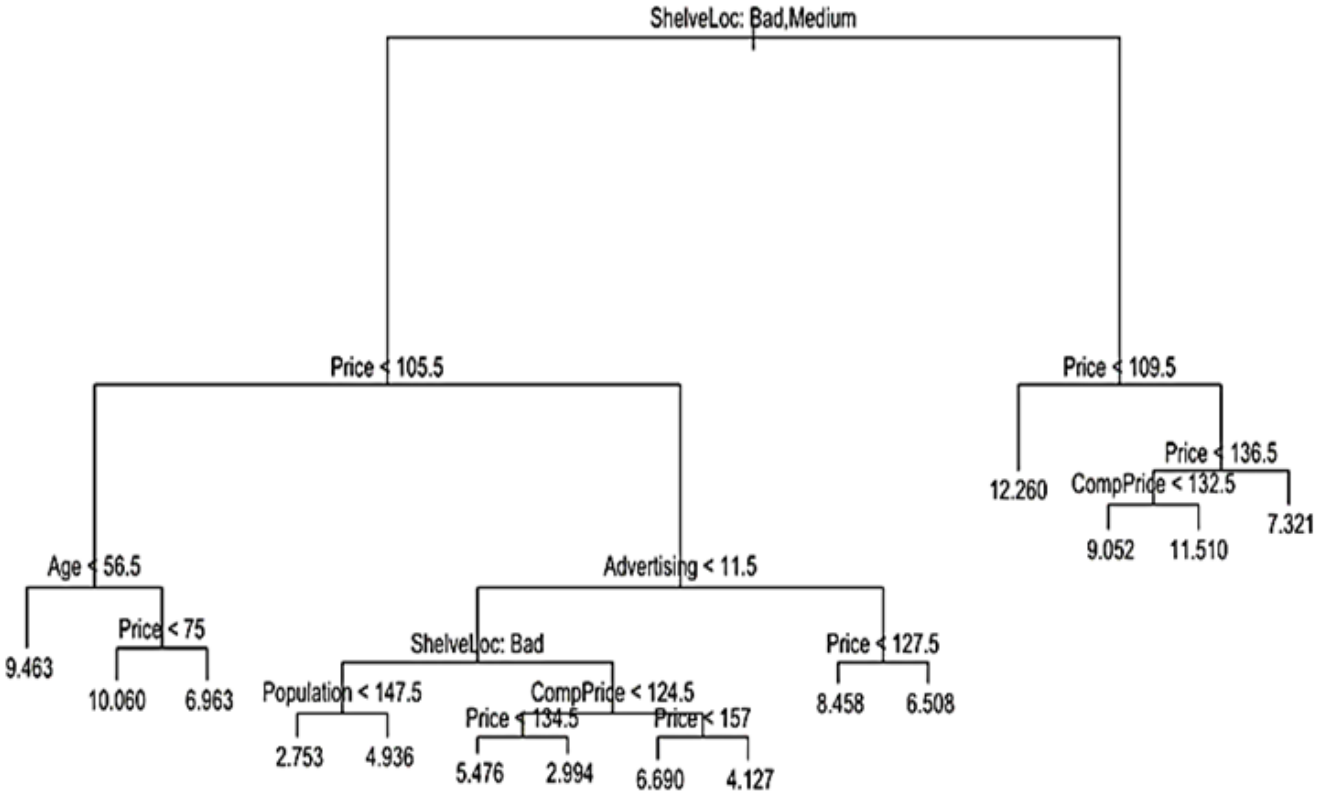

ShelveLoc: Bad,Medium
Price < 105.5
Price < 109.5
Age < 56.5
Advertising < 11.5
Price < 136.5
12.260
CompPrice < 132.5
7.321
9.052
11.510
Price < 75
ShelveLoc: Bad
Price < 127.5
9.463
10.060
6.963
Population < 147.5
CompPrice < 124.5
8.458
6.508
Price < 134.5
Price < 157
2.753
4.936
5.476
2.994
6.690
4.127

图 7: 回归树结果

树的优缺点

- 优点
- 易理解、解释性强；
- 不需要任何先验假设；
- 与传统的回归和分类方法相比，更接近人的大脑决策模式；
- 可以用图形展示，可视化效果好，非专业人士也可轻松解释；
- 可以直接处理定性的自变量，而不需像线性回归那样将定性变量转换成虚拟变量。

树的优缺点

- 缺点
- 决策树方差大、不稳定，数据很小的扰动可能得到完全不同的分裂结果，有可能是完全不同的决策树。
- 组合算法，即组合大量决策树（回归或分类模型）可以降低方差，显著提升预测效果。目前，常用的组合算法主要有装袋法（Bagging）、随机森林（Random Forest, RF）和提升法（Boosting）等。

Bagging

基本算法

- Bagging 是 bootstrap aggregating 的缩写，它指的是利用 Bootstrap 抽样方法对训练集进行抽样，得到一系列新的训练集，对每个训练集构建一个预测器，最后组合所有预测器得到最终的预测模型。
- 给定 n 个独立同分布 (iid) 的观测值 $Z_1, Z_2, \cdots, Z_n$，每个观测值的方差都是 σ^2，则它们的平均值的方差为 σ^2/n。也就是说，对一组观测值求平均可以减小方差。所以设训练样本集 T 为 (x_i, y_i)，其中 x_i 为 p 维自变量，y_i 为因变量。
- 对此数据集，若能从总体中抽取多个训练集，对每个训练集分别建立预测模型，再对由此得到的多个预测值进行“投票”便能得出最后的结果

- 遗憾的是，一般情况下我们只有一个训练集，对于一个训练集，如何建立多个预测模型呢？这时可以用 Bootstrap 抽样法解决，，此处即对　个样本点进行概率为 1/ 的等概率有放回抽样，样本量仍为　。这就是 Bagging 方法的基本思想。
- 事实证明，通过成百甚至上千棵树的组合，Bagging 方法能大幅提升预测准确性。

Bagging 算法

1. 对一个训练集T，我们进行Bootstrap抽样，得到B个样本量为n的训练样本集$\{T_1, T_2, \cdots T_B\}$；
2. 用这个训练集进行决策树生成，得到B个决策树模型：
   (1). 对于回归问题，我们得到回归树，记为 $f(x;T_b)$，$b=1,2,\cdots,B$
   (2). 对于分类问题，我们得到分类树，记为 $h(x;T_b)$，$b=1,2,\cdots,B$
3. 当对新样本进行预测时，由每个决策树得到一个预测结果，再进行“投票”得出最后的结果：
   （1）对于回归问题，最后的预测结果为所有决策树预测值的平均数

$$F(x)=\sum_{b=1}^{B}\left(f(x;T_b)\right)/B$$

   （2）对于分类问题，最终的预测结果为所有决策树预测结果中最多的那类

$$H(x)=\arg\max_j N_j$$

其中，$N_j=\sum_{b=1}^{B}\{I(h(x;T_b)=j)\}$，$I(\square)$ 为示性函数

图 8: Bagging

说明

- 树的棵数 B 并不是一个对 Bagging 法起决定作用的参数，B 值很大时也不会产生过拟合，但会增加计算量。
- 在实践中，我们取足够大的 B 值，比如 200 左右，就可以使误差能够稳定下来。
- 需要说明的是，Bagging 是一种组合学习的算法，不仅仅用在决策树模型上，也可以用到其他模型，但我们主要以决策树模型为例。

变量重要性的度量

- 与单棵树相比，Bagging 通常能提高预测的准确性。但遗憾的是，由此得到的模型可能难以解释。
- 决策树的优点之一是它能得到漂亮且易于解释的图形。然而，当大量的树被组合后，就无法仅用一棵树展现结果，也无法知道在整个过程中哪些变量最为重要。组合方法对预测准确性的提高是以牺牲解释性为代价的。
- 我们可以使用 RSS（针对回归树）或 Gini 指数（针对分类树）对自变量的重要性做出整体概括。
- 对于回归树，我们可以打乱给定任一变量的顺序，这样该变量与因变量就没有任何关系，实际上是一个无任何作用的自变量，计算该自变量打乱前后而减小的 RSS 的总量，对每个减小总量在所有　棵树上取平均，值越大说明自变量越重要。
- 对于分类树，可以对某一给定的自变量在一棵树上因分裂而使 Gini 指数的减小量加总，再取所有 B 棵树的平均，值越大说明自变量越重要。

重要性

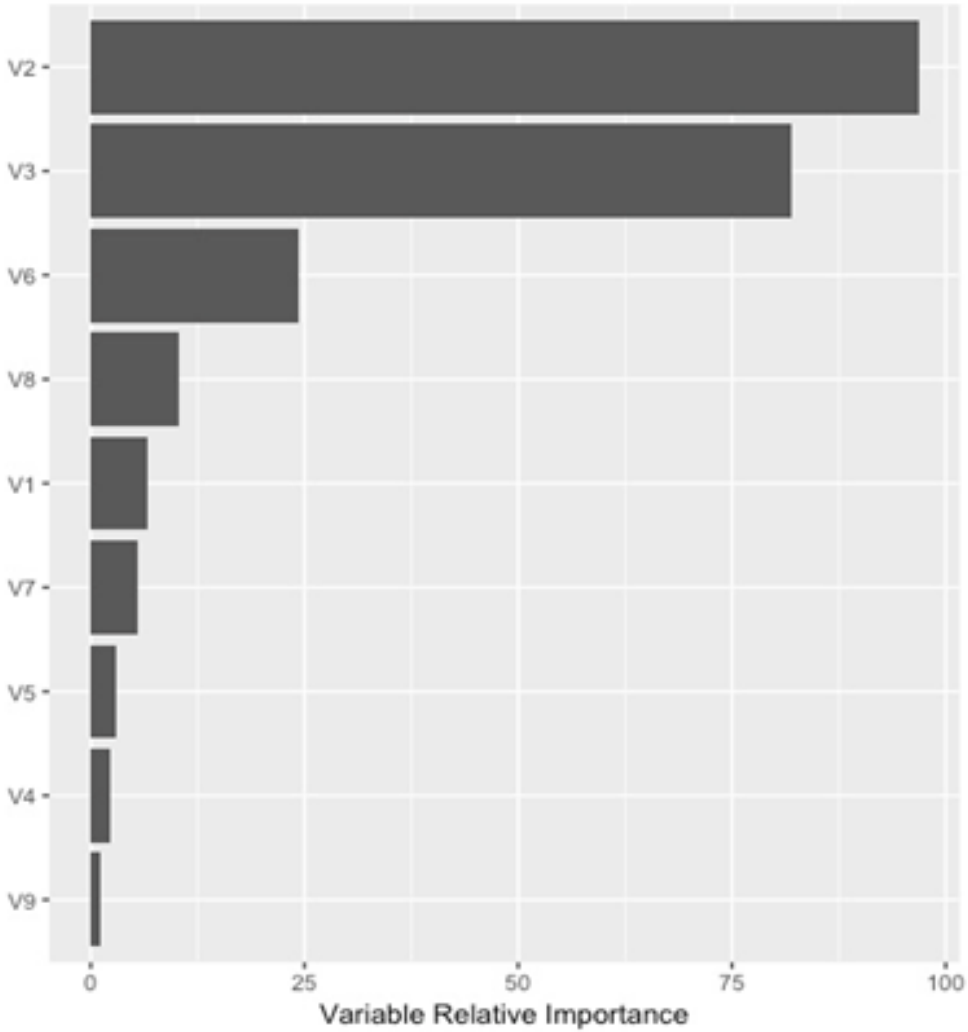


图 9: 基于 biopsy 数据集的变量重要性

随机森林

随机森林（RF）

- RF 是由加州大学伯克利分校统计系教授 Breiman 于 2001 年提出来的一种统计学习理论。
- 与 Bagging 算法类似，随机森林算法首先建立若干互不相关的树，再对各树的结果进行“投票”得到最终的预测结果。
- 我们可以发现，上面的描述与 Bagging 的区别在于建立的树是“互不相关”的，顾名思义，随机森林是通过对树做了去相关（decorrelate）处理，从而实现对 Bagging 改进的一种算法。

随机森林算法

1. 对于每棵树$b = 1,2,\cdots,B$;

1）从包含全部样本的训练集T中进行Bootstrap抽样，得到一个样本量为n的训练样本集T^*。

2）利用T^*建立一棵决策树，对于树上的每一个节点，重复以下步骤，直到节点的样本数达到指定的最小限定值n_{min}:

2.1）从全部p个随机变量中随机取$m(m < p)$个;

2.2）从这个变量中选取最优分裂变量，将此节点分裂成两个子节点。

注：对于分类问题，构造每棵树时默认使用$m = \sqrt{p}$个随机变量，节点最小样本数为1;

对于回归问题，构造每棵树时默认使用$m = p/3$个随机变量，节点最小样本数为5。

2. 当对新样本进行预测时，由每个决策树得到一个预测结果，再进行“投票”得出最后的结果，“投票”的含义同Bagging算法。

图 10: 随机森林算法

说明

- Bagging 和随机森林最大的不同就在于自变量子集的规模 m。
- 若取 $m = p$ 建立随机森林，则等同于建立 Bagging 树。另外，和 Bagging 一样，随机森林也可以使用袋外（OOB）预测值估计预测误差，也能得到变量的重要性排序，并且也不会因为 B 的增大而造成过拟合，所以在实践中应取足够大的 B ，比如 $B > 200$ 就能使分类错误率降低到一个稳定的水平。

提升法

提升法

- 提升法（Boosting）也是一种非常通用的将多个弱预测器组合成强预测器的方法。
- Bagging 通过 Bootstrap 抽样得到多个训练样本，对每个训练样本建立决策树，最后将这些树结合起来建立一个预测模型。
- 每一棵树都是建立在各自的训练样本集上, Boosting 也采用类似的方法，只是这里的树都是顺序生成的，即每棵树的构建都需要用到之前生成的树中的信息。
- 最经典的几个 Boosting 算法，Adaboost、GBDT 以及近年来被广泛应用的 XGBoost。

Adaboost 算法

1. 初始化样本抽样权重 $w_i = \frac{1}{n}, i = 1,2,\cdots,n$
2. 对 $m = 1,2,\cdots,M$:
   1). 以 w_i 为样本权重对于训练集 T_m 建模并得到分类器 $h(x,T_m)$
   2). 应用 $h(x,T_m)$ 预测训练集 T 中所有样本点，计算 $err_m = \frac{\sum_{i=1}^{n} w_i I\left(y_i \neq h(x,T_m)\right)}{\sum_{i=1}^{n} w_i}$
   3). 计算 $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$
   4). 重新计算样本抽样权重 $w_i \leftarrow w_i \exp\left[\alpha_m I\left(y_i \neq h(x,T_m)\right)\right], i = 1,2,\cdots,n$
3. 通过投票建立加权函数 $H(x) = \arg\max_{y\in\{1,\cdots K\}}\left[\sum_{m=1}^{M} \alpha_m I\left(y = h(x,T_m)\right)\right]$

模型中参数 M 可通过交叉验证法进行选择。

图 11: AdaBoost

GBDT

- AdaBoost 算法通过给已有模型预测错误的样本更高的权重，使得先前的学习器做错的训练样本在后续受到更多的关注的方式来弥补已有模型的不足。
- 与 AdaBoost 算法不同，GBDT（Gradient Boost Decision Tree）在迭代的每一步构建一个能够沿着梯度最陡的方向降低损失（steepest-descent）的学习器来弥补已有模型的不足。
- 经典的 AdaBoost 算法只能处理采用指数损失函数的二分类学习任务，而 GBDT 算法通过设置不同的可微损失函数可以处理各类学习任务（多分类、回归、Ranking 等），应用范围大大扩展。

GBDT 算法

1. 给定损失函数$L(y,\gamma)$, 初始化 $f_0(x)=\arg\min_{\gamma}\sum_{i=1}^{n}L(y_i,\gamma)$ 得到只有一个根节点的树，即γ是一个常数值。
2. 对$m=1,2,\cdots,M$:
   1）计算损失函数的负梯度在当前模型的值，将它作为残差的估计值:

$$r_{i,m}=-\left[\frac{\partial L(y_i,f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}},\quad i=1,2,\cdots,n$$

   2）根据$r_{i,m}$训练回归树得到叶节点区域$R_{j,m}, j=1,2,\ldots,J_m$。
   3）利用线性搜索估计叶节点区域的值，使损失函数极小化:

$$\gamma_{j,m}=\arg\min_{\gamma}\sum_{x_i\in R_{j,m}}L\left(y_i,f_{m-1}(x_i)+\gamma\right),\ j=1,2,\cdots,J_m$$

   4）更新回归树 $f_m(x)=f_{m-1}(x)+\sum_{j=1}^{J_m}\gamma_{j,m}I(x\in R_{j,m})$
3. 输出最终模型 $\hat{f}(x)=f_M(x)$

图 12: GBDT

XGBoost

- XGBoost 是 Extreme Gradient Boosting 的简称，是陈天奇在 2016 年提出的，兼具线性模型求解器和树学习算法。它是在 Gradient Boosting Decision Tree（GBDT）算法上的改进，更加高效。
- 传统的 GBDT 方法只利用了一阶的导数信息，XGBoost 则是对损失函数做了二阶的泰勒展开，并在目标函数之外加入了正则项整体求最优解，用以权衡目标函数的下降和模型的复杂程度，避免过拟合，使求得模型的最优解的效率更高。
- 在实际应用中，XGBoost 是 Gradient Boosting Machine 的一个 C++ 实现，最大的特点在于，它能够自动利用 CPU 的多线程进行并行，同时在算法上加以改进提高了精度。它是目前最快最好的开源 boosted tree 工具包，比常见的工具包快 10 倍以上。XGBoost 提供多种目标函数，包括回归，分类和排序等。
- 由于在预测性能上的强大表现，XGBoost 成为很多数据挖掘比赛的理想选择。在优化模型时，这个算法有非常多的参数需要调整。

1. 假设我们有K个基分类器：

（1）模型为：$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$

（2）目标函数为：$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$

注：目标函数第一部分是训练误差，可采用平方误差等形式，第二部分是基分类器的复杂程度，以决策树为基分类器，可以用叶节点个数或树的深度来衡量。这是一个加法模型，类似前向分布算法。

2. 每次保留原来的模型不变，加入一个新的基分类器到模型中：

$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$
$$\cdots$$
$$\hat{y}_i^{(t)} = \sum_{k=1}^{K} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

图 13: XGBoost

3. 选取每一轮的新的基分类器，这个新的基分类器使得目标函数尽量最大地降低。

因为： $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

所以 $Obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{K} \Omega(f_k) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + const$

4. 将目标函数做泰勒展开，并引入正则项，

$$Obj^{(t)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + const$$

$$\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + const$$

其中，$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ ， $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

图 14: XGBoost

本周推荐

1. 一本书：《番茄工作法》，弗朗西斯科西里洛，北京联合出版公司，2019
2. 一部电影：《时间旅行者的妻子》，2009
3. 练习：决策树和集成学习方法的实现，2024 年

```r
library(MASS)
data("biopsy")
library(rpart)
library(rpart.plot)
# 将数据分为训练集和测试集
set.seed(13) # 设置随机种子以保证结果可复现
train_indices <- sample(1:nrow(biopsy), nrow(biopsy) * 0.8)
biopsy <- biopsy[,-1]
train_data <- biopsy[train_indices, ]
test_data <- biopsy[-train_indices, ]
```

```
# 构建决策树模型
tree_model <- rpart(class ~ ., data =
                      train_data, method = "class")
print(tree_model)

## n= 559
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 559 199 benign (0.64400716 0.35599284)
##    2) V2< 2.5 337  10 benign (0.97032641 0.02967359)
##      4) V1< 6.5 330   4 benign (0.98787879 0.01212121) *
##      5) V1>=6.5 7   1 malignant (0.14285714 0.85714286) *
##    3) V2>=2.5 222  33 malignant (0.14864865 0.85135135)
##      6) V3< 2.5 18   4 benign (0.77777778 0.22222222) *
##      7) V3>=2.5 204  19 malignant (0.09313725 0.90686275)
##       14) V2< 4.5 61  15 malignant (0.24590164 0.75409836)
##         28) V6< 3.5 16   6 benign (0.62500000 0.37500000) *
##         29) V6>=3.5 45   5 malignant (0.11111111 0.88888889)
```

```r
# 在测试集上进行预测
predictions <- predict(tree_model, test_data, type = "class")
#print(predictions)

# 评估模型性能
confusion_matrix <- table(Predicted = predictions, Actual = test
print(confusion_matrix)
```

```
##             Actual
## Predicted    benign malignant
##   benign         96         5
##   malignant       2        37
```

```r
# 计算准确率
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
cat("Accuracy:", accuracy, "\n")
```

```
## Accuracy: 0.95
```

谢　谢!