

NOTE on Machine Learning with R

zy

2020 年 10 月 24 日

目录

1 机器学习简介	7
1.1 机器学习的起源	7
1.2 机器如何学习	7
1.2.1 抽象化和知识的表达	7
1.2.2 一般化	7
1.2.3 评估学习的成功性	8
1.3 将机器学习应用于数据中的步骤	8
1.4 选择机器学习算法	8
1.4.1 考虑输入的数据	9
1.4.2 考虑机器学习算法的类型	9
1.4.2.1 预测模型	9
1.4.2.2 描述性模型	9
1.4.3 为数据匹配合适的算法	10
2 数据的管理和理解	11
2.1 R 的数据结构	11
2.2 因子	11
2.2.1 列表	12
2.2.2 数据框	13
2.2.3 矩阵和数组	13
2.3 用 R 管理数据	13
2.3.1 保存和加载 R 数据结构	13
2.3.2 用 csv 文件导入和保存数据	13
2.3.3 从 SQL 数据库导入数据	14
2.4 探索和搜索数据	14
2.4.1 探索数据结构	14
2.4.2 探索数值型变量	14
2.4.2.1 测量中心趋势—平均数和中位数	15
2.4.2.2 测量数据的分散程度—四分位数和五数汇总	15
2.4.2.3 数值型变量可视化—箱图	16
2.4.2.4 数值型变量可视化—直方图	17
2.4.2.5 了解数值型数据—均匀分布和正态分布	17
2.4.2.6 衡量数据的分散程度—方差/标准差	17

2.4.3	探索分类变量	18
2.4.3.1	table() 函数	18
2.4.3.2	衡量中心趋势-众数	19
2.4.4	探索变量之间的关系	19
2.4.4.1	变量之间关系的可视化-散点图	19
2.4.4.2	检验变量之间的关系-双向交叉表	20
3	懒惰学习-使用近邻分类	22
3.1	理解使用近邻进行分类	22
3.1.1	kNN 算法	22
3.1.1.1	计算距离	24
3.1.1.2	选择一个合适的 k	24
3.1.1.3	准备 kNN 算法使用的数据	25
3.1.2	为什么 kNN 算法是懒惰的	26
3.2	用 kNN 算法诊断乳腺癌	26
3.2.1	第一步-收集数据	26
3.2.2	第二步-探索和准备数据	27
3.2.2.1	转换-min-max 标准化数值型数据	28
3.2.2.2	数据准备-创建训练数据集和测试数据集	29
3.2.3	第三步-基于数据训练模型	29
3.2.4	第四步-评估模型的性能	30
3.2.5	第五步-提高性能	31
3.2.5.1	转换-z-score 标准化	31
3.2.5.2	测试其他的 k 值	32
4	概率学习-朴素贝叶斯分类	34
4.1	理解朴素贝叶斯	34
4.1.1	贝叶斯方法的基本概念	34
4.1.1.1	概率/联合概率	34
4.1.1.2	基于贝叶斯定理的条件概率	35
4.1.2	朴素贝叶斯算法	35
4.1.2.1	朴素贝叶斯分类	35
4.1.2.2	拉普拉斯估计	37
4.1.2.3	在朴素贝叶斯算法中使用数值特征	37
4.2	例子-基于贝叶斯算法的收集垃圾短息过滤	38
4.2.1	第一步-收集数据	38
4.2.2	第二步-探索和准备数据	38
4.2.3	数据准备-处理和分析文本数据	38
4.2.3.1	数据准备-建立训练数据集和测试数据集	41
4.2.3.2	可视化文本数据-词云	42
4.2.3.3	数据的准备-为频繁出现的单词创建指示特征	44
4.2.4	第三步-基于数据训练模型	45

4.2.5 第五步–提升模型的性能	46
5 分而治之–应用决策树和规则进行分类	48
5.1 理解决策树	48
5.1.1 分而治之	49
5.1.2 C5.0 决策树算法	50
5.1.2.1 选择最佳的分割	51
5.1.2.2 修剪决策树	53
5.2 例子–使用 C5.0 决策树识别高风险银行贷款	53
5.2.1 第一步–收集数据	53
5.2.2 第二步–探索和准备数据	53
5.2.2.1 数据准备–创建随机的训练数据集和测试数据集	54
5.2.3 第三步–基于数据训练模型	55
5.2.4 第四步–评估模型性能	58
5.2.5 第五步–提高模型性能	59
5.2.5.1 提高决策树的准确性	59
5.2.5.2 犯一些比其他错误更严重的错误	60
5.3 理解分类规则	61
5.3.1 独立而治之	62
5.3.2 单规则 (1R) 算法	64
5.3.3 RIPPER 算法	64
5.3.4 来自决策树的规则	66
5.4 例子–应用规则学习识别有毒的蘑菇	66
5.4.1 第一步–收集数据	66
5.4.2 第二步–探索和准备数据	66
5.4.3 第三步–基于数据训练模型	68
5.4.4 第四步–评估模型的性能	69
5.4.5 第五步–提高模型性能	69
6 预测数值型数据–回归方法	73
6.1 理解回归	73
6.1.1 简单线性回归	74
6.1.2 普通最小二乘估计	74
6.1.3 相关系数	75
6.1.4 多元线性回归	75
6.2 例子–应用线性回归预测医疗费用	77
6.2.1 第一步–收集数据	77
6.2.2 第二步–探索和准备数据	77
6.2.2.1 探索特征之间的关系–相关系数矩阵	78
6.2.2.2 可视化特征之间的关系–散点图矩阵	78
6.2.3 第三步–基于数据训练模型	80
6.2.4 第四步–评估模型的性能	81

6.2.5 第五步-提高模型性能	82
6.2.5.1 模型的设定-添加非线性关系	82
6.2.5.2 转换-将一个数值型变量转换为一个二进制指标	83
6.2.5.3 模型的设定-加入相互作用的影响	83
6.2.5.4 全部放在一起-一个改进的回归模型	83
6.3 理解回归树和模型树	84
6.3.0.1 将回归加入到决策树	84
6.4 例子-用回归树和模型树估计葡萄酒的质量	86
6.4.1 第一步-收集数据	86
6.4.2 第二步-探索和准备数据	86
6.4.3 第三步-基于数据训练模型	88
6.4.3.1 可视化决策树	89
6.4.4 第四步-评估模型的性能	91
6.4.4.1 用平均绝对误差度量性能	91
6.4.5 第五步-提高模型的性能	92
7 黑箱方法-神经网络和支持向量机	94
7.1 理解神经网络	94
7.1.1 从生物神经元到人工神经元	94
7.1.2 激活函数	95
7.1.3 网络拓扑	97
7.1.3.1 层的数目	97
7.1.3.2 信息传播的方向	98
7.1.3.3 每一层的节点数	99
7.1.4 用后向传播训练神经网络	99
7.2 用人工神经网络对混凝土的强度进行建模	100
7.2.1 第一步-收集数据	100
7.2.2 第二步-探索和准备数据	100
7.2.3 第三步-基于数据训练模型	102
7.2.4 第四步-评估模型性能	103
7.2.5 第五步-提高模型的性能	103
7.3 理解支持向量机	105
7.3.1 用超平面分类	105
7.3.2 寻找最大间隔	106
7.3.2.1 线性可分的数据情况	106
7.3.2.2 非线性可分的数据情况	107
7.3.3 对非线性空间使用核函数	108
7.4 用支持向量机进行光学字符识别	109
7.4.1 第一步-收集数据	109
7.4.2 第二步-探索和准备数据	109
7.4.3 第三步-基于数据训练模型	110
7.4.4 第四步-评估模型的性能	112

7.4.5 第五步-提高模型的性能	114
8 探寻模式-基于关联规则的购物篮分析	115
8.1 理解关联规则	115
8.1.1 用于关联规则学习的 Apriori 算法	115
8.1.1.1 度量规则兴趣度-支持度和置信度	116
8.1.1.2 用 Apriori 原则建立规则	117
8.2 例子-用关联规则确定经常一起购买的食品杂货	117
8.2.1 第一步-收集数据	117
8.2.2 第二步-探索和准备数据	117
8.2.2.1 数据准备-为交易数据创建一个稀疏矩阵	117
8.2.2.2 可视化商品的支持度-商品的频率图	120
8.2.2.3 可视化交易数据-绘制稀疏矩阵	121
8.2.3 第三步-基于数据训练模型	122
8.2.4 第四步-评估模型性能	124
8.2.5 第五步-提高模型性能	125
8.2.5.1 对关联规则集合排序	125
8.2.5.2 提取关联规则的子集	126
8.2.5.3 将关联规则保存到文件或者数据框	127
8.3 总结	127
9 寻找数据的分组-k 均值聚类	128
9.1 理解聚类	128
9.1.1 聚类-一种机器学习任务	128
9.1.2 k 均值聚类算法	129
9.1.2.1 使用距离来分配和更新类	130
9.1.2.2 选择适当的聚类数	132
9.1.3 用 k 均值聚类探寻青少年市场细分	133
9.1.3.1 第一步-收集数据	133
9.1.3.2 第二步-探索和准备数据	133
9.1.3.3 数据准备-缺失值的虚拟编码	134
9.1.3.4 数据准备-插补缺失值	135
9.1.4 第三步-基于数据训练模型	136
9.1.5 第四步-评估模型性能	137
9.1.6 第五步-提高模型性能	138
9.2 总结	139
10 模型性能评价	140
10.1 度量分类方法的性能	140
10.1.1 在 R 中处理分类预测数据	140
10.1.2 深入讨论混淆矩阵	142
10.1.3 使用混淆矩阵度量性能	143

10.1.3.1 得到混淆矩阵的方法	143
10.1.4 准确度之外的其他性能评价指标	144
10.1.4.1 Kappa 统计量	145
10.1.4.2 灵敏度和特异性	147
10.1.4.3 精确度和回溯精确度	148
10.1.4.4 F 度量	149
10.1.5 性能权衡的可视化	149
10.1.5.1 ROC 曲线	149
10.2 评估未来的性能	152
10.2.1 保持法	152
10.2.2 交叉验证	153
10.2.3 自助法抽样	156
11 提高模型性能	157
11.1 调整多个模型来提高性能	157
11.1.1 使用 caret 进行自动参数调整	157
11.1.1.1 创建简单的调整的模型	159
11.1.1.2 定制调整的过程	161

Chapter 1

机器学习简介

1.1 机器学习的起源

1. 机器学习的研究领域：发明计算机算法，把数据转换为智能行为。
2. 机器学习与数据挖掘的差异：机器学习侧重执行一个已知的任务，数据挖掘侧重寻找有价值的信息。

1.2 机器如何学习

1. 基础的学习过程：

- 数据输入：利用观察，记忆储存和回忆来提供进一步推理的事实依据。
- 抽象化：涉及数据转换为更宽泛的形式。
- 一般化：应用抽象的数据来形成行动的基础。

1.2.1 抽象化和知识的表达

1. 把原始数据用一个结构化的格式来表达是学习算法的最典型任务。在知识表达的过程中，计算机把原始输入概括在一个模型里，这个模型是数据间结构化模式的一个显示表述。
2. 用一个特定的模型来拟合数据集的过程称为训练。

1.2.2 一般化

1. 一般化这个术语描述了把抽象化的知识转换成可以用于行动的形式。具体来说，假设有一个包含所有数据建立的可能模型或理论的集合，一般化就是把这个集合里的理论的数量减少到一个可以管理的数目。
2. 机器学习算法一般用一些能更快划分概念集合的捷径方法，为此算法应用启发式算法，或者有训练地猜测在何处能快速找到最重要的概念。

由于启发式方法利用相近原理和其他一些经验法则，所以他不能保证找到对数据建模的最优概念集。但不用的话，从大数据集中找到有用信息的任务变得不可能。

启发式方法通常用来快速地把经验推广到新的场景。如果你曾经在充分评估你的情况之前应用直觉来做出快速决定，那你已不自觉使用了精神启发方法。

3. 启发式方法可能会导致不合理地结论。如果算法地结论是系统性的不精确，就说该算法有偏差。

1.2.3 评估学习的成功性

偏差是与任何机器学习任务的抽象化和一般化这两个过程相联系的不可避免的谬误。一般化过程的最后一步就是在存在偏差的情况下判断模型的成功性。

在初始的数据集上训练模型之后，它要被一个新的数据集检验，并且判断从训练数据得到的特征推广到新数据的好坏程度。

在某种程度上，由于数据的噪声或者无法解释的波动导致模型不能完美地一般化。噪声数据是由看起来随机的事件所导致的，如

- 测量错误，传感器不精确；
- 报告的数据有问题，如被调查者随意答题；
- 由于数据记录不正确而导致错误，包括数据缺失，空值，截断，错误编码，取值冲突。

用模型拟合噪声称为过度拟合问题的基础，因为由定义可知，噪声是无法被解释的，尝试解释噪声会导致错误的结论，该结论不能推广到新的场景中。一个模型在训练时表现得很好，但是当测试时就表现得很差的现象，称为过度拟合了训练集数据集，即它不能很好的一般化。

过度拟合问题的解决办法视具体机器学习方法而不同。就现在而言，重要的是要意识到问题的存在。模型处理噪声数据的好坏是判断模型成功与否的一个重要方面。

1.3 将机器学习应用于数据中的步骤

1. 收集数据：转为适合分析的电子格式。数据将作为机器学习算法的学习材料，从而产生可行动的知识。
2. 搜索数据和准备数据：项目质量取决于数据质量。
3. 基于数据训练模型：具体的机器学习任务会告知你合适的算法，算法将会以模型的形式来表现数据。
4. 评价模型的性能：每个机器模型将会产生一个学习问题的有偏差的解决方法，所以评价算法从经验中学习的优劣是很重要的。根据使用模型的类型，你应该能用一个测试数据集来评价模型的精确性，或者你可能需要针对目标应用设计一个模型性能的检验标准。
5. 改进模型性能：有时需要完全更换为不同的模型，可能需要补充另外的数据，或者如这个过程的第二个步骤中做的那样，进行一些额外的数据准备工作。

在完成这些步骤之后，如果模型表现令人满意，就能将它应用到预期的任务中。

1.4 选择机器学习算法

机器学习算法的算法涉及衡量学习数据的特征和可以使用的方法所具有的偏差。

1.4.1 考虑输入的数据

输入训练数据就最基础的格式而言，输入数据是以 example(案例) 和 feature(特征) 组成的表格形式呈现的。

特征分为很多形式。如果一个特征所代表的特性是用数值来衡量的，此为数值型；若所代表的属性是通过一组类别表示的，此特征称为分类变量或者名义变量。分类变量中有一种特殊的类型：有序变量，如衣服的尺码 (大，中，小)。

1.4.2 考虑机器学习算法的类型

机器学习算法分两类：

- 用来建立预测模型的有监督学习算法
- 用来建立描述模型的无监督学习算法

用哪种取决于你需要完成的学习任务。

1.4.2.1 预测模型

用于这样的学习任务：利用数据集中的其他数值来预测另一个值。学习算法的目的是发现并且建模目标特征 (需要预测的特征) 和其他特征之间的关系。预测可以预测未来的事情，也可以是过去的。

因为预测模型对于学什么和怎么学有清晰的指导，所以训练一个预测模型的过程也称为有监督学习。监督并不是指需要人为干预，它是指让目标值担任监督的角色，让它告诉学习者要学习的任务是什么。具体来说，给一组数据，学习算法尝试最优化一个函数 (模型) 来找出属性值之间的组合方式，最终据此给出目标值。

常见的有监督学习任务是预测案例属于哪个类别，这类学习任务成为分类。如预测一个队会赢还是输，一个人会不会活过 100 岁等等。被预测的目标属性是一个称为类的分类属性，它可以被分为不同的类别，类别称之为水平。一个类能有两个或者更多的水平，水平不一定是有序的。

有监督学习算法也可以用来预测数值型数据，如收入，试验数据，考试成绩等等。为了预测这类数值型数值能够拟合输入数据的线性回归模型是一类常见的数值型预测，应用非常广泛。回归模型被广泛地应用于预测，因为它用表达式精确量化了输入数据和目标值之间的关系，包括该关系的强弱大小和不确定性。

注：因为可以轻易的将数值型数据转换为类别，类别转换为数值，故分类模型和数值预测模型之间的界限不太严格。

1.4.2.2 描述性模型

通过新而有趣的方式总结数据并获得洞察，从而学习任务从这些洞察中受益。与需要预测目标值的预测模型不同，在描述模型里，没有一个属性比其他属性更重要。因为没有一个要学习的目标，所以训练描述性模型的过程称为无监督学习。尽管思考描述性模型的应用更加困难，但是好的地方是学习者没有特定的学习任务，这种方法在数据挖掘中经常使用。

eg。称为模式发现的描述性模型任务，用来识别数据之间联系的紧密性，如购物篮分析。

描述性模型中把数据集按照相同类型分组的任务称为聚类。有时候用于细分分析，即根据相似的购买记录、捐赠记录或者人口普查信息进行人群分类，使得广告能针对目标受众。尽管机器能识别各个分组，但还是需要人工介入来解释各个分组。

1.4.3 为数据匹配合适的算法

表 1.1: 本书讨论的机器学习算法类型

模型	任务	章节
有监督学习算法		
最近邻	分类	3
朴素贝叶斯	分类	4
决策树	分类	5
分类器	分类	5
线性回归	数值预测	6
回归树	数值预测	6
模型树	数值预测	6
神经网络	双重用处	7
支持向量机	双重用处	7
无监督学习算法		
关联规则	模式识别	8
k 均值聚类	聚类	9

为了给学习任务找到相应的机器学习方法，需要从以上四种任务之一开始。确定的学习任务将使算法的选择变得简单。

对于分类来说，需要把精力花费在为学习问题找到合适的分类器。考虑到不同算法间的各种差别是很有帮助的。如，在分类问题中，决策树得到的模型很容易理解，而神经网络得到的模型很难解释。如信用评分模型，即使神经网络算法更好的预测贷款违约，但是如果不能解释清楚预测规则，再好也没用。

注：当预测准确性是主要考虑因素时，可能需要测试多个模型选一个最好的。

Chapter 2

数据的管理和理解

任何学习算法的好坏取决于输入数据的好坏. 在很多情况下, 输入数据是复杂的, 凌乱的, 来源渠道也未必相同, 所以投入到机器学习项目中的很大一部分精力要花在数据准备和探索过程中.

2.1 R 的数据结构

在机器学习中 R 的数据结构是: 向量, 因子, 列表, 数组, 数据框.

2.2 因子

回忆第一章 1.4.1, 用类别值来代表特征的属性称为名义属性. 尽管可以用一个字符型向量来储存名义属性数据, 但是 R 提供了称为因子 (factor) 的专用数据结构来表示这种属性数据. 其实因子是向量的一个特例, 它单独用来标识名义属性 (或变量).

为什么不用 character 字符型向量呢? 因为使用因子的一个优势在于: 因为类别标签只储存一次, 所以它们比一般的字符型向量更加有效. 不像字符型向量需要储存多个一样的 male,female, 在使用因子的时候计算机只需要储存多个 1,2, 节约内存.

另外, 一些机器学习算法是用特别的程序来处理分类变量的, 把分类变量编码为因子可以确保模型能够合理地处理这些数据.

```
1 > # add gender factor
2 > gender <- factor(c("MALE", "FEMALE", "MALE"))
3 > gender
4 [1] MALE   FEMALE MALE
5 Levels: FEMALE MALE
6 >
7 > # add blood type factor
8 > blood <- factor(c("O", "AB", "A"),
9 +                   levels = c("A", "B", "AB", "O"))
10 > blood
11 [1] O  AB A
12 Levels: A B AB O
```

2.2.1 列表

另一种特殊的向量, 称为列表, 它用来存储一组有序的值. 然而, 不像向量要求所有的元素都必须是同一种类型, 列表允许存储不同类型的值. 由于这个灵活性, 列表一直用于存储不同类型的输入和输出数据, 以及存储机器学习模型所使用的结构参数.

```
1 > # create vectors of data for three medical patients
2 > subject_name <- c("John Doe", "Jane Doe", "Steve Graves")
3 > temperature <- c(98.1, 98.6, 101.4)
4 > flu_status <- c(FALSE, FALSE, TRUE)
5 > gender <- factor(c("MALE", "FEMALE", "MALE"))
6 > blood <- factor(c("O", "AB", "A"),
7 +                     levels = c("A", "B", "AB", "O"))
8 >
9 > # create list for a patient
10 > subject1 <- list(fullname = subject_name[1],
11 +                     temperature = temperature[1],
12 +                     flu_status = flu_status[1],
13 +                     gender = gender[1],
14 +                     blood = blood[1])
15 >
16 > # display the patient
17 > subject1
18 $fullname
19 [1] "John Doe"
20
21 $temperature
22 [1] 98.1
23
24 $flu_status
25 [1] FALSE
26
27 $gender
28 [1] MALE
29 Levels: FEMALE MALE
30
31 $blood
32 [1] O
33 Levels: A B AB O
```

2.2.2 数据框

到目前为止, 机器学习中使用的最重要的 R 数据结构就是数据框. 在 R 术语中, 数据框定义为一个向量列表或者因子列表, 每一列的数值个数都相同. 因为数据框准确来说是一个向量列表, 所以它结合了向量和列表两个方面的特点.

```
1 pt_data <- data.frame(subject_name, temperature, flu_status, gender,
2 blood, stringsAsFactors = FALSE)
```

其中参数 stringsAsFactors = FALSE 值得注意. 如果不指定这个选项, R 将自动把每个字符向量转换为因子. 这个参数有时有用, 但有时候又显得很累赘. 这里 name 明显不是分类数据, 其中的取值也不是分类变量的类别.

2.2.3 矩阵和数组

向量, 矩阵不再赘叙, 自行查阅.

2.3 用 R 管理数据

当处理大量数据时, 面临的挑战包括从各种不同来源中收集, 准备和管理数据. R 提供了从诸多常见格式的数据源加载数据的工具.

2.3.1 保存和加载 R 数据结构

要想把一种特定的数据结构保存到一个文件里, 使它以后能被重新加载或者转移, 用 save() 函数.

如果我们有三个对象 x,y,z, 可以用下面的命令把它们保存到文件 mydata.RData

```
1 > save(x,y,z, file="mydata.RData")
```

load() 命令将会加载任何一种保存在以".RData" 格式的文件中的数据结构.

```
1 > load("mydata.RData")
```

注意, 用 load() 命令导入的文件中所保存的所有数据结构都会加载到工作区, 即使他们会覆盖工作区中其他一些你正在使用的东西.

如果你需要立即结束当前 R 会话, save.image() 命令会把你所有的会话写入一个.RData 文件. 默认情况下, 在 R 下次启动时自动寻找这个文件, 故看到的页面和当时离开一样.

2.3.2 用 csv 文件导入和保存数据

表格数据文件, 采用矩阵形式结构, 每行为一个 example, 每个 example 有相同数量的特征. 通常情况下, 表格数据文件的第一行给出数据每一列的名称, 称为标题行.

最常用的表格文本文件格式可能是逗号分隔值 (comma-separated value, csv) 文件. 加载 csv 文件:

```
1 > pt_data <- read.csv("pt_data.csv", stringsAsFactors = FALSE)
```

文件不用指定路径的前提是该文件在 R 工作目录中. 上面的会把 csv 文件读入到 pt_data 的数据框.

默认情况下,R 假设 csv 文件包含一个标题行, 标题行列出了数据集内特征的名字. 若没有, 要指定 header=FALSE, R 会用 V1,V2 等默认值来指定属性名.

```
1 > pt_data <- read.csv("pt_data.csv", stringsAsFactors=FALSE, header=FALSE)
```

read.csv() 函数是 read.table() 的特例,read.table() 能读取具有多种不同格式的表格数据, 用到再查.

保存数据框为 csv 文件:

```
1 > write.csv(pt_data, file="pt_data.csv")
```

保存到 R 工作目录.

2.3.3 从 SQL 数据库导入数据

还没学, 放着先....

2.4 探索和搜索数据

数据导入:

```
1 usedcars<-read.csv("D:/Program Files (x86)/RStudio/machinelearning/usedcars.csv",
2 stringsAsFactors = FALSE)
```

2.4.1 探索数据结构

调查的第一个问题, 数据是怎么组织的. 数据字典是一个描述数据特征的文档, 如果数据源并没有提供, 那就自己创建.

函数 str() 可以显示数据框结构, 能用来创建数据字典的基本轮廓:

```
1 > str(usedcars)
2 'data.frame': 150 obs. of 6 variables:
3 $ year      : int  2011 2011 2011 2011 2012 2010 2011 2010 2011 2010 ...
4 $ model     : chr  "SEL" "SEL" "SEL" "SEL" ...
5 $ price      : int  21992 20995 19995 17809 17500 17495 17000 16995 16995 16995 ...
6 $ mileage    : int  7413 10926 7351 11613 8367 25125 27393 21026 32655 36116 ...
7 $ color      : chr  "Yellow" "Gray" "Silver" "Gray" ...
8 $ transmission: chr  "AUTO" "AUTO" "AUTO" "AUTO" ...
```

2.4.2 探索数值型变量

调查数值型变量, 可以使用一组普遍使用的描述数值的指标, 称为汇总统计量. summary() 函数给出了几个常用的汇总统计量.

```

1 > summary(usedcars)
2 year          model          price          mileage
3 Min.   :2000  Length:150      Min.   : 3800  Min.   : 4867
4 1st Qu.:2008  Class  :character  1st Qu.:10995  1st Qu.: 27200
5 Median  :2009  Mode   :character  Median  :13592  Median  : 36385
6 Mean    :2009          Mode   :character  Mean    :12962  Mean    : 44261
7 3rd Qu.:2010          Mode   :character  3rd Qu.:14904  3rd Qu.: 55125
8 Max.    :2012          Mode   :character  Max.    :21992  Max.    :151479
9
10 color         transmission
11 Length:150      Length:150
12 Class  :character  Class  :character
13 Mode   :character  Mode   :character

```

汇总统计量可以分为两种类型: 数据的中心测度和分散程度测度.

2.4.2.1 测量中心趋势—平均数和中位数

均值: mean()

中位数: median()

注意落在值域两端的值对均值和中位数的影响是不同的. 均值对异常值非常敏感.

2.4.2.2 测量数据的分散程度—四分位数和五数汇总

五数汇总是一组五个统计量, 都包含在 summary() 函数中, 见上面代码.

最值之差称为值域, range() 返回最值, diff() 返回差值, 结合可得值域.

```

1 > # the min/max of used car prices
2 > range(usedcars$price)
3 [1] 3800 21992
4 >
5 > # the difference of the range
6 > diff(range(usedcars$price))
7 [1] 18192

```

第一四分位数和第三四分位数之间的 50% 数据是数据分散程度的一个测试, 俩分位数的差称为四分位距 (inter quartile range, IQR)

```

1 > # IQR for used car prices
2 > IQR(usedcars$price)
3 [1] 3909.5

```

quantile() 函数返回五数汇总的值, 若指定 probs 参数, 可以用向量设置分位点. 序列函数 seq() 用来产生间距相同的向量.

```

1 > # use quantile to calculate five-number summary
2 > quantile(usedcars$price)
3 0%      25%      50%      75%      100%
4 3800.0 10995.0 13591.5 14904.5 21992.0
5 >
6 > # the 99th percentile
7 > quantile(usedcars$price, probs = c(0.01, 0.99))
8 1%      99%
9 5428.69 20505.00
10 > # quintiles
11 > quantile(usedcars$price, seq(from = 0, to = 1, by = 0.20))
12 0%      20%      40%      60%      80%      100%
13 3800.0 10759.4 12993.8 13992.0 14999.0 21992.0

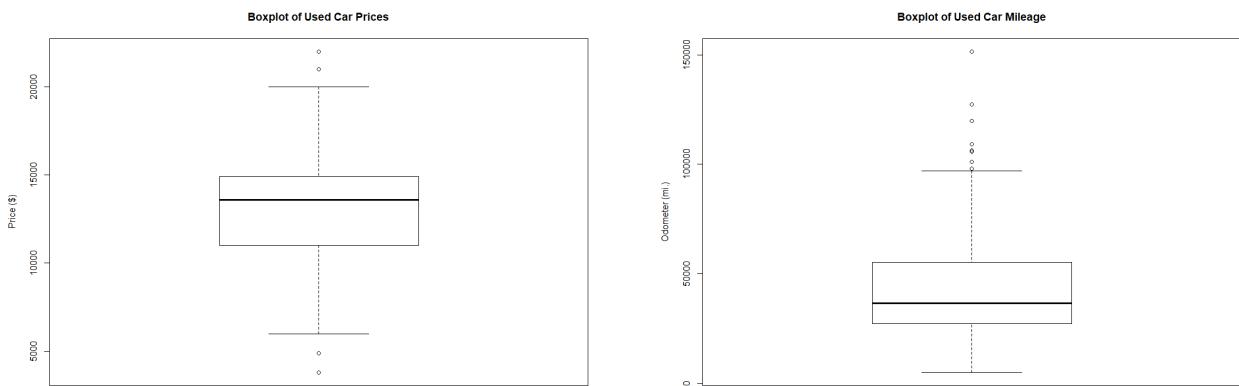
```

2.4.2.3 数值型变量可视化—箱图

```

1 > # boxplot of used car prices and mileage
2 > boxplot(usedcars$price, main="Boxplot of Used Car Prices",
3 +           ylab="Price ($)")
4 >
5 > boxplot(usedcars$mileage, main="Boxplot of Used Car Mileage",
6 +           ylab="Odometer (mi.)")

```



1. 箱子上三条线分别为第一分位数 Q1, 第二分位数 Q2(中位数), 第三分位数 Q3. 中位数用粗黑线.
2. 箱子上边和下边的两条线为最值线. 注意, 通常仅允许细线延伸到最小为低于 Q1 的 1.5 倍 IQR 的最小值, 和最大为高于 Q3 的 1.5 倍 IQR 的最大值. 超过这两个临界值都被称为异常值, 用圈圈表示.
3. 显然 IQR 为箱子宽度.

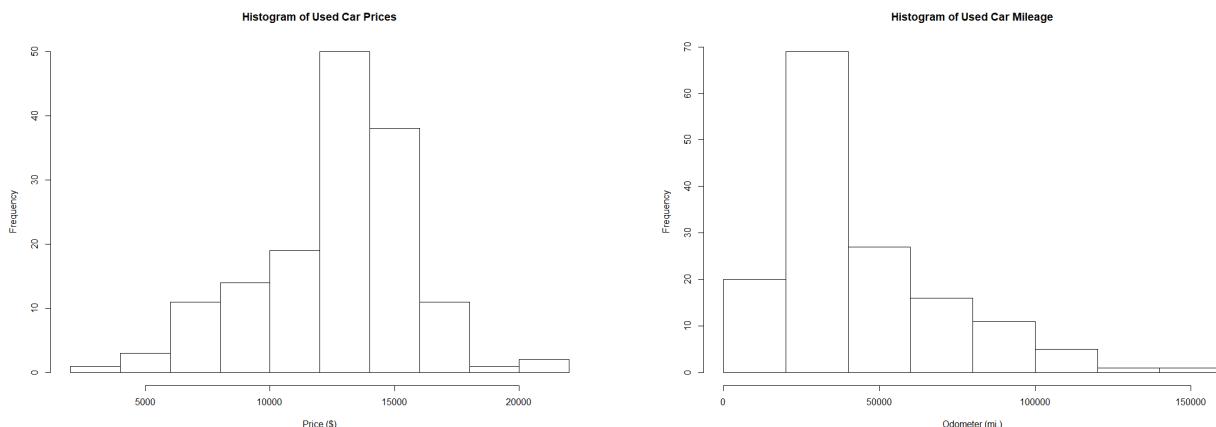
2.4.2.4 数值型变量可视化—直方图

箱线图将数据分成四部分, 每部分的数据量相同; 直方图按等宽分段, 每段数据量可以不同.

```

1 > # histograms of used car prices and mileage
2 > hist(usedcars$price, main = "Histogram of Used Car Prices",
3 +       xlab = "Price ($)")
4 >
5 > hist(usedcars$mileage, main = "Histogram of Used Car Mileage",
6 +       xlab = "Odometer (mi.)")

```



直方图是由一系列薯条构成, 其高度表示落在等长划分数据值的分段内的数据值的个数/频率.

左图稍微右偏, 右图明显左偏.

2.4.2.5 了解数值型数据—均匀分布和正态分布

变量的分布描述了一个值落在不同的值域里的可能性大小. 若所有值等可能, 那么为均匀分布, 如投骰子. 正态分布比较普遍.

2.4.2.6 衡量数据的分散程度—方差/标准差

```

1 > var(usedcars$price)
2 [1] 9749892

```

```

3 > sd(usedcars$price)
4 [1] 3122.482
5 > var(usedcars$mileage)
6 [1] 728033954
7 > sd(usedcars$mileage)
8 [1] 26982.1

```

注意 R 内置的函数用的是样本方差, 无偏的, 除以 $n-1$ 的.

方差越大, 表示数据在均值周围越分散; 标准差表示平均来看每个值和均值相差多少.

若为正态分布, 有个 69-95-99.7 rule, 正态分布中 68% 的值落在均值左右 1 个标准差的范围内, 95, 99.7 分别对应 2 个和 3 个.

2.4.3 探索分类变量

2.4.3.1 table() 函数

分类数据使用表格而不是汇总统计量进行检测的. 表示单个分类变量的表格称为一元表. 函数 table() 可以生成它.

```

1 > # one-way tables for the used car data
2 > table(usedcars$year)

3
4 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
5 3     1     1     1     3     2     6     11    14    42    49    16    1
6 > table(usedcars$model)

7
8 SE SEL SES
9 78 23 49
10 > table(usedcars$color)

11
12 Black   Blue   Gold   Gray   Green   Red   Silver   White   Yellow
13 35      17      1      16      5      25      32      16      3

```

类别下的数字表示不同类别的个数.

在 table() 产生的表格上使用函数 prop.table() 可以计算格子比例, 即不同类别的占比.

```

1 > # compute table proportions
2 > model_table <- table(usedcars$model)
3 > prop.table(model_table)

4
5 SE      SEL      SES
6 0.5200000 0.1533333 0.3266667

```

对求出来的格子比例/类别占比, 若想要用保留一位小数的百分数表示:

```

1 > # round the data
2 > color_table <- table(usedcars$color)
3 > color_pct <- prop.table(color_table) * 100
4 > round(color_pct, digits = 1)
5
6 Black   Blue   Gold   Gray   Green   Red   Silver   White   Yellow
7 23.3   11.3   0.7   10.7   3.3   16.7   21.3   10.7   2.0

```

2.4.3.2 衡量中心趋势-众数

众数通常出现在分类数据中, 因为均值和中位数不是为名义变量定义的. 一个变量可能有多个众数, 只有一个众数的变量称为单峰的, 两个就双峰的, 多个就多峰的.

统计量的众数只需要查看表格输出的结果中具有最大值的类别即可.

众数是从定性的角度来了解数据集的重要值, 但是太关注它可能有危险, 因为最常见的值未必就是绝大多数. 考虑众数时最好把它和其它类别联系起来, 是否有一个类别占主导地位, 或者多个? 据此, 我们可能会问: 最常见的值告诉我们被测量变量的哪些信息.

严格来说, 数值型的连续变量没有众数. 然而如果我们把众数考虑成直方图中的那个薯条, 就能讨论它们的众数了. 当探索数值型数据时, 考虑众数是很有帮助的, 特别要检验数据是否为多峰的.

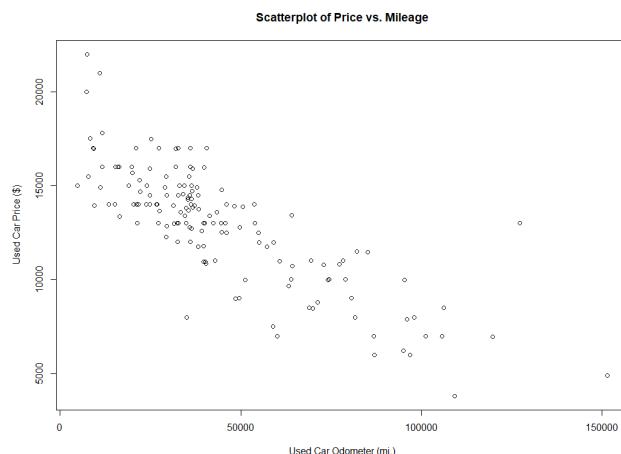
2.4.4 探索变量之间的关系

2.4.4.1 变量之间关系的可视化-散点图

```

1 > plot(x = usedcars$mileage, y = usedcars$price,
2 +       main = "Scatterplot of Price vs. Mileage",
3 +       xlab = "Used Car Odometer (mi.)",
4 +       ylab = "Used Car Price ($)")

```



2.4.4.2 检验变量之间的关系—双向交叉表

为了检验两个名义变量之间的关系, 使用双向交叉表. 交叉表和散点图相似, 允许你观察一个变量的值时如何随着另一个变化而变化的.

双向交叉表的格式是: 行是一个变量的水平, 列是另一个变量的水平. 每个表格的单元格中的值用来表明落在特定行, 列的单元格中的数量.

table() 也可以生成双向交叉表, 但一般更喜欢用 gmodels 包里的 CrossTable(), 因为它在一个表格中出现了行列和边际百分比.

在进一步分析之前, 我们决定将颜色 color 分为两类, 保守类和非保守类, 创建一个二元指示变量(常常称为哑变量). 根据我们的定义来表示汽车的颜色是否是保守的, 如果保守, 指示变量的值为 1, 否则为 0.

```
1 > library(gmodels)
2
3 > # new variable indicating conservative colors
4 > usedcars$conservative <-+
5 +     usedcars$color %in% c("Black", "Gray", "Silver", "White")
```

操作符%in%, 他根据左边的值是否在右边的向量中, 为操作符左边向量中的每一个值返回 TRUE/FALSE.

```
1 > # checking our variable
2 > table(usedcars$conservative)
3 FALSE  TRUE
4 51     99
```

现在看看交叉表中保守颜色与型号的关系, 因为假定了型号决定了颜色, 于是设保守色为 y, 型号为 x:

```
1 > CrossTable(x = usedcars$model, y = usedcars$conservative)
2
3 Cell Contents
4 |-----|
5 |           N |
6 | Chi-square contribution |
7 |           N / Row Total |
8 |           N / Col Total |
9 |           N / Table Total |
10 |-----|
11
12 Total Observations in Table:  150
13
14           | usedcars$conservative
15 usedcars$model | FALSE | TRUE | Row Total |
```

16	-----	-----	-----	-----
17	SE	27	51	78
18		0.009	0.004	
19		0.346	0.654	0.520
20		0.529	0.515	
21		0.180	0.340	
22	-----	-----	-----	-----
23	SEL	7	16	23
24		0.086	0.044	
25		0.304	0.696	0.153
26		0.137	0.162	
27		0.047	0.107	
28	-----	-----	-----	-----
29	SES	17	32	49
30		0.007	0.004	
31		0.347	0.653	0.327
32		0.333	0.323	
33		0.113	0.213	
34	-----	-----	-----	-----
35	Column Total	51	99	150
36		0.340	0.660	

第一个小小表格, 说明如何解释每个值.

第二个大表格, 行表示二手车三个型号, 最后一行是汇总; 表格的列是汽车颜色是否保守, 最后一列是对所有两种颜色求和. 每个格子第一行是该型号和颜色的数量, 比例分别表示这个格子的卡方统计量, 以及在行, 列和整个表格中的占比.

卡方值指出了两个变量中每个单元格在皮尔森卡方独立检验中的贡献. 这个检验测量了表格中每个单元格内数量的不同只是由于偶然的可能性有多大. 如果概率值是非常低的, 那么就提供了充足的证据表明两个变量是相关的.

在引用 `CrossTable()` 函数时增加一个额外的参数, 指定 `chisq=TRUE` 来获得卡方的结果.

```

1 Statistics for All Table Factors
2
3 Pearson's Chi-squared test
4 -----
5 Chi^2=0.1539564      d.f.=2      p=0.92591

```

第三章, 我们将用最近邻方法着手处理第一个分类任务.

Chapter 3

懒惰学习—使用近邻分类

相似的东西很可能具有相似的属性. 利用这个原理, 我们可以对数据进行分类, 将其划分到最相近的类别或者最接近的邻居. 本章将专门讨论这种方法进行分类, 你将学到:

- 定义近邻分类器, 以及为什么它们被认为时懒惰学习器
- 通过距离来测量两个案例相似性的方法
- 如何使用 KNN(k-Nearest Neighbors, K 邻近) 算法 R 语言实现来诊断乳腺癌.

3.1 理解使用近邻进行分类

一句话, 近邻分类器就是把未标记的案例归类为与他们最相似的带有标记的案例所在的类. 看起来想法很 easy, 但是近邻分类方法是极其强大的, 它们以及成功应用在下列领域:

- 计算机视觉应用, 包括在静止图像和视频中的光学字符识别和面部识别.
- 预测一个人是否喜欢推荐给他们的电影.
- 识别基因数据的模式, 用于发现特定的蛋白质或疾病.

3.1.1 kNN 算法

用于分类的邻近方法是通过 kNN 算法实现的.

kNN 算法开始于一个分成几个类别的案例所组成的训练数据集. 类别由名义变量来标识. 假设我们有一个由未标记的案例构成的测试数据集. 除去分类标记外, 测试数据集和训练数据集有相同的特征. 对于测试数据集中的每一个记录,kNN 确定训练数据集中与该记录相似度最近的 k 条记录, 其中 k 是一个预先指定的整数, 未标记的测试实例被分配到 k 个近邻中占比最大的那个类.

优点:

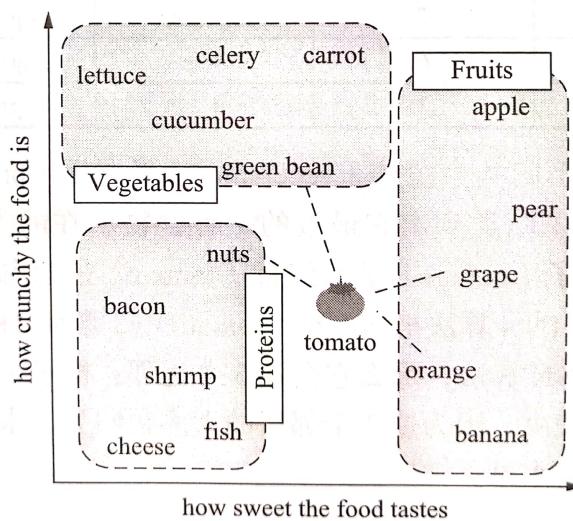
- 简单有效
- 对数据的分布没有要求
- 训练阶段很快

缺点:

- 不产生模型, 在发现特征之间关系上的能力有限
- 分类阶段很慢
- 需要大量内存
- 名义变量 (特征) 和缺失数据需要额外处理

以水果分类为例:

ingredient	sweetness	crunchiness	food
apple	10	9	fruit
bacon	1	4	protein
banana	10	1	fruit



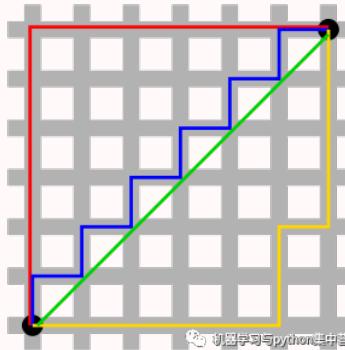
kNN 算法将特征处理为一个多维特征空间 (feature space) 内的坐标. 由于示例的数据集只包含两个特征, 故为二维的. 可以进行散点图绘制.

相似类型的食物趋向于聚集得更近. 我们决定用它来解决一个古老难题: tomato 水果还是蔬菜?(如图)

3.1.1.1 计算距离

定位 tomato 得近邻需要一个距离函数或者一个用来衡量两个实例 example 之间得相似性得公式. kNN 算法用 **欧氏距离 (Euclidean distance)**. 欧氏距离通过直线距离 (crow flies) 来度量, 即最短的直接路线.

另一种常见的距离度量是 **曼哈顿距离 (Manhattan distance)**, 该距离基于一个行人在城市街区步行所采取得路线. 曼哈顿距离是与欧式距离不同的一种丈量方法, 两点之间的距离不再是直线距离, 而是投影到坐标轴的长度之和.



公式:

$$dist(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} \quad (3.1)$$

其中 p 和 q 是需要比较的 example, 它们都有 n 个特征. 如: $dist(tomato, greenbean) = \sqrt{(6 - 3)^2 + (4 - 7)^2} = 4.2$

为了将 tomato 归类为蔬菜/水果/蛋白质, 先将它归类到离它最近得一种食物所在得类型开始, 即此时 $k=1$, 故称 1NN 分类. 计算结果是, 橙子是 tomato 得近邻, 距离 1.4, 橙子是水果, 所以 tomato 也是水果.

如果使用 $k=3$ 的 kNN 算法, 那么它会在 3 个近邻: 橙子, 葡萄和坚果之间进行投票表决. 因为这三个邻居的大多数是水果 (2/3), 因此 tomato 也是水果.

3.1.1.2 选择一个合适的 k

确定用于 kNN 算法的邻居数量将决定把模型推广到未来数据时模型的好坏. 过度拟合和低度拟合训练数据之间的平衡问题称为 **偏差-方差衡 (bias-variance tradeoff)**.

选择一个大的 k 会减少噪声数据对模型的影响或者减少噪声导致的模型波动, 但是它会使分类器产生偏差, 如, 它有忽视不易察觉但却很重要模式的风险.

如果设置一个非常大的 k 等于训练数据中所有观察的数量. 由于每一个训练案例都会在最后的投票表决中出现, 故最常见的训练类总会获得大多数票.

如果使用单一的近邻会使得噪声数据或者异常值过度影响案例的分类. 如, 假设一些训练案例被意外的贴错标签, 而另外一些未标记案例恰好时最接近于被错误标记的训练案例, 那么它就会被预测到错误的类中.

显然 k 最好取上述两者之间的某个值.

在实际中, k 的选取取决于要学习概念的难度和训练数据中案例的数量. 通常 k 为 3~10. 一种常见的作法就是设置 k 等于训练集中 example 的数量的平方根. 另一种方法是基于各种测试数据集来测试多个 k 值, 并选择一个可以提供最好分类性能的 k 值.

从另一方面来说, 除非数据的噪声非常大, 否则更大的更具代表性的训练数据集可以使 k 值的选择不那么重要. 这是因为即使是微小的概念, 也将有一个足够大的案例池来进行投票, 以便选出近邻.

一个不太常见但挺有趣的解决这个问题的方法是: 选择一个较大的 k , 但是同时权重投票, 就是较近的邻居比远邻的投票更加有权威

3.1.1.3 准备 kNN 算法使用的数据

在应用 kNN 算法之间, 通常将特征转换为一个标准范围内. 这个步骤的合理性在于, 距离公式依赖于特征是如何被度量的.

特别的, 如果某个特征的值具有比其他特征更大的值, 那么距离的度量就会强烈地被这个较大的值指配. 这时候我们需要的是一种收缩或者重新缩放各种特征值的方法, 以使得每个特征对距离公式的贡献相对平均.

对 kNN 算法的特征进行重新调整的传统方法是 **min-max 标准化**

该过程将特征的值转换为 0~1, 公式如下:

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

可以这样解释: 按 0% ~ 100% 来说, 在原始最小值和原始最大值范围内, 原始值到原始最小值的距离有多远.

另一种常见的变换为 **z-score 标准化**

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - \text{Mean}(X)}{\text{StdDev}(X)}$$

基于在第二章中介绍的正态分布性质, 即根据每一个特征的值均落在均值上下的标准差的数量来重新调整每一个特征的值. z-score 落在一个无解和负数和正数构成的范围内, 无预定的最值.

对于名义数据, 欧氏距离并不是为名义数据定义的. 因此, 为了计算名义特征之间的距离, 需要将他们转化为数值型格式. 一种典型的解决方案就是利用**哑变量编码 (dummy coding)**, 其中 1 表示一个类别, 0 是另一个.

eg.

$$male = \begin{cases} 1 & x \text{ 为 male,} \\ 0 & x \text{ 为其他.} \end{cases}$$

注意对含有两个可能取值的二元变量, 只需要其中一个就可以了, 因为是互斥的.

一个具有 n 个类别的名义特征可以通过对特征的 $(n-1)$ 个水平创建二元指示变量来进行哑变量编码. 例如, 为一个具有 3 个类别的温度变量进行哑变量编码 (如 hot, medium, cold), 可以用 3-1=2, 两个

特征来进行设置:

$$hot = \begin{cases} 1 & x \text{ 为 hot,} \\ 0 & x \text{ 为其他.} \end{cases}$$

$$medium = \begin{cases} 1 & x \text{ 为 medium,} \\ 0 & x \text{ 为其他.} \end{cases}$$

这里, 只要知道 hot 和 medium 的值都是 0 就足以说明温度为 cold, 故不需要为 cold 属性设置第三个特征.

哑变量编码的一个方便之处就在于特征之间的距离总是 1 或 0, 因此与 min-max 标准化的数值型数据一样, 这些值落在了同一个标度内, 不需要进行额外的变换.

如果名义特征是有序的, 如以上温度的例子, 那么一种哑变量编码的替代方法就是给类别编号并且应用 min-max 标准化. eg. cold,warm 和 cold 可以编号 1,2,3, min-max 标准化为 0,0.5,1.

使用这种方法要注意, 只有当你确信类别之间的步长相等, 你才可以应用该方法, 否则用原来的哑变量编码更保险.

3.1.2 为什么 kNN 算法是懒惰的

基于近邻方法的分类算法被认为是懒惰学习算法, 因为从技术上来说, 没有抽象化的步骤. 抽象过程和一般化过程都被跳过了, 这就破坏了第一章给出的学习的定义.

显然懒惰学习从学习的严格定义上来说, 并不是真正在学习什么, 仅仅是一字不差地存储训练数据, 这样地训练阶段就进行地很快, 同时伴随潜在不利因素: 进行预测的过程往往相对较慢.

由于高度依赖训练案例, 又称懒惰学习为基于实例的学习 (instance-based learning)/机械学习 (rote learning). 由于该学习并不会建立一个模型, 所以该方法被归类为非参数学习方法, 即没有需要学习的数据参数. 因为没有产生关于数据的理论, 所以非参数方法限制了我们理解分类器如何使用数据的能力. 另一方面, 它允许学习算法发现数据中的自然模式, 而不是试图将数据拟合为一个预先设定的形式.

尽管很懒惰, 但是还是很强大! 见下.

3.2 用 kNN 算法诊断乳腺癌

定期乳腺癌检查使得疾病在引起明显症状前得到治疗. 早期检查包括检查乳腺组织的异常肿块. 如果有, 细针抽吸活检, 确定为恶性还是良性. 从带有异常乳腺肿块的女性身上的活检细胞测度数据入手, 用 kNN 算法, 研究机器学习用于检测癌症的功效.

3.2.1 第一步—收集数据

乳腺癌数据包括 569 例细胞活检案例, 每个案例 32 个特征. 一个特征是 id, 一个特征是诊断结果, 用编码 M 表恶性,B 表良性. 余下 30 个数值型测量结果是由数字化细胞核的 10 个不同特征的均值, 标准差和最差值 (最大值) 构成. 这些特征包括:

radius 半径, texture 质地, perimete 周长, area 面积, smoothness 光滑度, compactness 致密性, concavity 凹度, concave points 凹点, symmetry 对称性, dimension 分形维度.

```
1 > wbcd <- read.csv("G:/机器学习/ml&r/Machine Learning with R/chapter 3/  
2 wisc_bc_data.csv", stringsAsFactors = FALSE)
```

3.2.2 第二步—探索和准备数据

```
1 > str(wbcd)  
2 'data.frame': 569 obs. of 32 variables:  
3 $ id : int 87139402 8910251 905520 868871 9012568 ...  
4 $ diagnosis : chr "B" "B" "B" "B" ...  
5 $ radius_mean : num 12.3 10.6 11 11.3 15.2 ...  
6 $ texture_mean : num 12.4 18.9 16.8 13.4 13.2 ...  
7 $ perimeter_mean : num 78.8 69.3 70.9 73 97.7 ...  
8 $ area_mean : num 464 346 373 385 712 ...  
9 $ smoothness_mean : num 0.1028 0.0969 0.1077 0.1164 0.0796 ...  
10 $ compactness_mean : num 0.0698 0.1147 0.078 0.1136 0.0693 ...  
11 $ concavity_mean : num 0.0399 0.0639 0.0305 0.0464 0.0339 ...  
12 $ points_mean : num 0.037 0.0264 0.0248 0.048 0.0266 ...  
13 $ symmetry_mean : num 0.196 0.192 0.171 0.177 0.172 ...  
14 $ dimension_mean : num 0.0595 0.0649 0.0634 0.0607 0.0554 ...  
15 $ radius_se : num 0.236 0.451 0.197 0.338 0.178 ...  
16 $ texture_se : num 0.666 1.197 1.387 1.343 0.412 ...  
17 $ perimeter_se : num 1.67 3.43 1.34 1.85 1.34 ...  
18 $ area_se : num 17.4 27.1 13.5 26.3 17.7 ...  
19 $ smoothness_se : num 0.00805 0.00747 0.00516 0.01127 0.00501 ...  
20 $ compactness_se : num 0.0118 0.03581 0.00936 0.03498 0.01485 ...  
21 $ concavity_se : num 0.0168 0.0335 0.0106 0.0219 0.0155 ...  
22 $ points_se : num 0.01241 0.01365 0.00748 0.01965 0.00915 ...  
23 $ symmetry_se : num 0.0192 0.035 0.0172 0.0158 0.0165 ...  
24 $ dimension_se : num 0.00225 0.00332 0.0022 0.00344 0.00177 ...  
25 $ radius_worst : num 13.5 11.9 12.4 11.9 16.2 ...  
26 $ texture_worst : num 15.6 22.9 26.4 15.8 15.7 ...  
27 $ perimeter_worst : num 87 78.3 79.9 76.5 104.5 ...  
28 $ area_worst : num 549 425 471 434 819 ...  
29 $ smoothness_worst : num 0.139 0.121 0.137 0.137 0.113 ...  
30 $ compactness_worst : num 0.127 0.252 0.148 0.182 0.174 ...  
31 $ concavity_worst : num 0.1242 0.1916 0.1067 0.0867 0.1362 ...  
32 $ points_worst : num 0.0939 0.0793 0.0743 0.0861 0.0818 ...  
33 $ symmetry_worst : num 0.283 0.294 0.3 0.21 0.249 ...  
34 $ dimension_worst : num 0.0677 0.0759 0.0788 0.0678 0.0677 ...
```

剔除第一列这个没啥用的 id:

```
1 > wbcn <- wbcn[-1]
```

许多 R 机器学习分类器要求将目标属性编码为因子型. 同时我们也会用 labels 参数对 B 值和 M 值给出含有更多信息地标签:

```
1 > wbcn$diagnosis <- factor(wbcn$diagnosis, levels = c("B", "M"),
2 +                               labels = c("Benign", "Malignant"))
```

看一看变量 diagnosis:

```
1 > table(wbcn$diagnosis)
2 B   M
3 357 212
4 > round(prop.table(table(wbcn$diagnosis))*100,digits = 1)
5 B   M
6 62.7 37.3
```

百分之 62.7 良性, 百分之 37.3 恶性.

其余三十个特征都是数值型, 与预期一样, 由十个细胞核特征的三种不同测量构成, 作为示例, 我们这里详细地看看三个特征.

```
1 > summary(wbcn[c("radius_mean", "area_mean", "smoothness_mean")])
2 radius_mean      area_mean      smoothness_mean
3 Min.   : 6.981   Min.   : 143.5   Min.   : 0.05263
4 1st Qu.:11.700   1st Qu.: 420.3   1st Qu.: 0.08637
5 Median :13.370   Median : 551.1   Median : 0.09587
6 Mean   :14.127   Mean   : 654.9   Mean   : 0.09636
7 3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.: 0.10530
8 Max.   :28.110   Max.   :2501.0   Max.   : 0.16340
```

3.2.2.1 转换—min-max 标准化数值型数据

需要创建一个 normalize() 函数

```
1 > normalize <- function(x) {
2 +   return ((x - min(x)) / (max(x) - min(x)))
3 + }
```

我们不需要手动逐个标准化, lapply() 函数可以输入一个列表, 然后将一个函数应用到列表的每一个元素. 注意算完了要把列表转回数据框:

```
1 > wbcn_n <- as.data.frame(lapply(wbcn[2:31], normalize))
```

3.2.2.2 数据准备—创建训练数据集和测试数据集

在训练期间得到的算法分类好坏的衡量指标可能是有误的, 因为我们不知道数据发生过度拟合的程度. 我们要了解对于一个没有标记数据的数据集, 学习算法的性能如何.

由于缺少这样的数据, 我们使用前 469 条数据作为训练数据集, 后 100 条数据作为测试数据集, 即模拟新病人.

```
1 > wbcn_train <- wbcn_n[1:469, ]  
2 > wbcn_test <- wbcn_n[470:569, ]
```

注意: 当构造训练数据集和测试数据集时, 保证每一个数据集都是数据全集的一个有代表性的子集是很重要的. 刚刚的案例中记录已经按照随机顺序排列, 故可以简单地提取创造数据集. 如果数据是按照非随机的模式排列, 如时间, 那么需要用到随机抽样方程.

当构建训练数据集和测试数据集时, 剔除了 diagnosis, 为了训练 kNN 模型, 需要将这些类的标签储存在一个因子类型的向量中, 然后把该向量划分为训练数据集和测试数据集:

```
1 > wbcn_train_labels <- wbcn[1:469, 1]  
2 > wbcn_test_labels <- wbcn[470:569, 1]
```

3.2.3 第三步—基于数据训练模型

对于 kNN 算法, 训练阶段实际上不包括模型的建立—训练一个懒惰学习算法的过程, 就像 kNN 算法仅涉及结构化格式存储输入的数据.

为了将测试实例分类, 使用来自 class 包的 knn() 函数, 该函数提供了一个标准的 kNN 算法实现, 对于测试数据中的每一个实例, 使用欧氏距离标识 k 个近邻. 通过 k 个近邻的投票来对测试个案分类. 如果票数相等, 则会被随机分类.

kNN 分类语法

应用 class 添加包中的函数 knn()

创建分类器并进行预测:

```
p <- knn(train, test, class, k)  
• train: 一个包含数值型训练数据的数据框  
• test: 一个包含数值型测试数据的数据框  
• class: 包含训练数据每一行分类的一个因子向量  
• k: 标识最近邻数目一个整数
```

该函数返回一个因子向量, 该向量含有测试数据框中每一行的预测分类。

例子:

```
wbcn_pred <- knn(train = wbcn_train, test = wbcn_test,  
                    cl = wbcn_train_labels, k = 3)
```

```
1 > library(class)  
2 > wbcn_test_pred <- knn(train = wbcn_train, test = wbcn_test,  
3 +                         cl = wbcn_train_labels, k=21)
```

由于测试数据集由 469 个, 故尝试 $k=21$, 大约等于 469 开方的奇数. 使用奇数是因为会减少各类票数相等这一情况发生的可能性. 最后返回一个因子向量, 为测试数据集中的每一个案例返回一个预测标签.

3.2.4 第四步-评估模型的性能

下一步就是评估 pred 和 labels 的匹配度了可以使用 gmodels 包中的 CrossTable(). 创建一个用来标识两个向量之间一致性的交叉表, 指定参数 prop.chisq=FALSE 除去不需要的卡方值.

```
1 > library(gmodels)
2 > CrossTable(x = wbcid_test_labels, y = wbcid_test_pred,
3 +             prop.chisq=FALSE)
4
5
6 Cell Contents
7 |-----|
8 |           N |
9 |           N / Row Total |
10 |          N / Col Total |
11 |          N / Table Total |
12 |-----|
13
14
15 Total Observations in Table: 100
16
17
18           | wbcid_test_pred
19 wbcid_test_labels | Benign | Malignant | Row Total |
20 |-----|-----|-----|-----|
21 Benign |      61 |        0 |      61 |
22 |      1.000 |  0.000 |  0.610 |
23 |      0.968 |  0.000 |        |
24 |      0.610 |  0.000 |        |
25 |-----|-----|-----|-----|
26 Malignant |      2 |      37 |      39 |
27 |      0.051 |  0.949 |  0.390 |
28 |      0.032 |  1.000 |        |
29 |      0.020 |  0.370 |        |
30 |-----|-----|-----|-----|
31 Column Total |      63 |      37 |     100 |
32 |      0.630 |  0.370 |        |
33 |-----|-----|-----|-----|
```

100 个肿块只有两个被错误分类.

在第十章, 你将学习更复杂的方法来度量预测的准确性, 根据每种错误类型的成本, 这些方法可以用来找出那些错误率可以被优化的地方.

3.2.5 第五步—提高性能

对于前面的分类器, 我们将尝试两种改变:1. 使用另一种方法重新调整数值特征;2. 尝试几个不同的 k 值.

3.2.5.1 转换—z-score 标准化

虽然 min-max 标准化是传统 kNN 分类的方式, 但是它并不一定总是最合适地调整特征的方法.

因为 z-score 标准化后的值没有预定义的最小值和最大值, 所以极端值不会被压缩到中心. 让异常值在距离计算中占有更大的权重有可能是合理的.

为了标准化一个向量, 使用 R 内置函数 `scale()` 函数, 该函数默认使用 z-score 标准化. 而且可以直接应用于数据框, 避免使用 `lapply()`.

```
1 > wbcz_z <- as.data.frame(scale(wbcd[-1]))
```

一个 z-score 标准化后的变量的均值应该始终为 0, 而且其值域非常紧凑, 一个大于 3 或小于 -3 的 z-score 表示一个极其罕见的值.

如同之前, 划分训练数据集和测试数据集, 再用 `knn()` 对测试实例进行分类. 再用 `CrossTable()` 比较预测的标签和实际的标签.

```
1 > wbcd_train <- wbcz_z[1:469, ]
2 > wbcd_test <- wbcz_z[470:569, ]
3 >
4 > wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
5 +                               cl = wbcd_train_labels, k=21)
6 >
7 > CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
8 +               prop.chisq=FALSE)
9
10 Cell Contents
11 |-----|
12 |           N |
13 |           N / Row Total |
14 |           N / Col Total |
15 |           N / Table Total |
16 |-----|
17
18 Total Observations in Table:  100
19
20 | wbcd_test_pred
```

wbcd_test_labels	Benign	Malignant	Row Total
Benign	61	0	61
	1.000	0.000	0.610
	0.924	0.000	
	0.610	0.000	
Malignant	5	34	39
	0.128	0.872	0.390
	0.076	1.000	
	0.050	0.340	
Column Total	66	34	100
	0.660	0.340	

很遗憾, 还不如不改.

3.2.5.2 测试其他的 k 值

使用 min-max 标准化的训练数据集和测试数据集, 用几个不同的 k 对相同的 100 个记录进行分类.

```

1 wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
2                           cl = wbcd_train_labels, k=1)
3 CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
4
5 wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
6                           cl = wbcd_train_labels, k=5)
7 CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
8
9 wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
10                          cl = wbcd_train_labels, k=11)
11 CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
12
13 wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
14                          cl = wbcd_train_labels, k=15)
15 CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
16
17 wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
18                          cl = wbcd_train_labels, k=21)
19 CrossTable(x = wbcd_test_labels, y = wbcd_test_pred, prop.chisq=FALSE)
20

```

```
21 wbcn_test_pred <- knn(train = wbcn_train, test = wbcn_test,  
22                         cl = wbcn_train_labels, k=27)  
23 CrossTable(x = wbcn_test_labels, y = wbcn_test_pred, prop.chisq=FALSE)
```

虽然分类器永远不会完美, 重要的是记住, 为了过于准确预测测试来调整我们的方法是不明智的.

如果你需要确认一个学习算法能推广到未来的数据, 那么你可能需要随机的创建几组 100 位病人的记录, 并且在这些数据上重复测试模型结果. 第十章将深入讨论仔细评估机器学习模型性能的方法.

Chapter 4

概率学习—朴素贝叶斯分类

在本章, 我们将研究使用概率来估计一个观测值落入某些特定类别中的分类方法, 比较该方法与 kNN 算法有何不同会很有趣.

本章讲述的机器学习算法, 即依据概率原则进行分类朴素贝叶斯算法. 贝叶斯算法就是应用先前时间的有关数据来估计未来时间发生的概率.

你将学习:

- 用于朴素贝叶斯的几倍概率原则
- 基于 R, 用专门的方法, 可视化, 和数据结构分析文本数据
- 如何运用朴素贝叶斯分类器建立短信过滤器并通过 R 实现

4.1 理解朴素贝叶斯

基于贝叶斯方法的分类器是利用训练数据并根据特征的取值来计算每个类别被观察到的概率. 当分类器之后被应用到无标签数据时, 分类器就会根据观测到的概率来预测新的特征最有可能属于哪个类.

这是简单的想法, 但根据这种想法就产生了一种方法, 而且这种方法的到的结果与很多复杂方法得到的结果是等价的. 事实上, 贝叶斯分类器已经用于以下方面:

- 文本分类, 如垃圾邮件过滤
- 在计算机网络中进行入侵检测
- 根据一组观察到的症状, 诊断身体状况.

通常情况下, 贝叶斯分类器适用于解决这样一类问题: 为了估计一个结果的概率, 从众多属性中提取的信息应该被同时考虑. 很多其他算法忽略了具有弱影响的一些特征, 但是贝叶斯方法利用了所有可以获得的证据来巧妙地修正. 如果有大量特征产生地影响很小, 但将他们放在一起, 他们的影响可能会相当大.

4.1.1 贝叶斯方法的基本概念

4.1.1.1 概率/联合概率

何书元和茆师松的著作欢迎查看 ~

4.1.1.2 基于贝叶斯定理的条件概率

贝叶斯定理

$$\mathbb{P}(\theta|y) = \frac{\mathbb{P}(y|\theta) \times \mathbb{P}(\theta)}{\mathbb{P}(y)} \quad (4.1)$$

$$\text{后验概率}(posterior) = \frac{\text{似然函数}(likelihood) \times \text{先验概率}(prior)}{\mathbb{P}(y)} \quad (4.2)$$

垃圾邮件例子自行查阅书籍.

4.1.2 朴素贝叶斯算法

朴素贝叶斯 (naive bayes) 算法描述应用贝叶斯定理进行分类的一个简单应用. 尽管这不是唯一应用贝叶斯方法的机器学习方法, 但它是最常见的, 尤其在文本分类应用已经成为一种准则.

优点:

- 简单, 快速, 有效
- 能处理好噪声数据和缺失数据
- 需要用来训练的例子相对较少, 但同样能处理好大量的例子
- 很容易获得一个预测的估计概率值

缺点:

- 依赖于一个常用的错误假设, 即一样的重要性和独立特征
- 应用在含有大量数值特征的数据时并不理想
- 概率的估计值相对于预测的类而言更加不可靠

朴素贝叶斯算法之所以这样命名是因为关于数据有一对“简单”的假设: 朴素贝叶斯假设数据集的所有特征都具有相同的重要性和独立性. 显然在大量的实际应用中鲜有成立.

然而在大多数情况下, 当违背这些假设时, 朴素贝叶斯依然很好用, 甚至在极端事件中, 特征之间具有很强的依赖性时, 朴素贝叶斯也可以用. 由于该算法的通用性和准确性, 适用于很多类型的条件, 所以在分类学习任务中, 朴素贝叶斯算法往往是很强大的, 排在候选算法的第一位.

4.1.2.1 朴素贝叶斯分类

以垃圾邮件过滤为例.

我们通过对词语 viagra,money,groceries,unsubscribe 的检测来改善我们的垃圾邮件过滤器. 我们可以通过构建出现的四个单词 (记为 W1,W2,W3,W4) 的似然表来训练朴素贝叶斯算法, 对 100 封邮件分析后的似然表如下

似然	Viagra(W ₁)		Money(W ₂)		Groceries(W ₃)		Unsubscribe(W ₄)		总计
	Yes	No	Yes	No	Yes	No	Yes	No	
垃圾邮件	4/20	16/20	10/20	10/20	0/20	20/20	12/20	8/20	20
非垃圾邮件	1/80	79/80	14/80	66/80	8/80	71/80	23/80	57/80	80
总计	5/100	95/100	24/100	76/100	8/100	91/100	35/100	65/100	100

利用贝叶斯定理, 我们这样定义问题的概率: 一封邮件在给定 viagra=yes,money=no,groceries=no, unsubscribe=yes 条件下为垃圾邮件的概率:

$$\mathbb{P}(\text{垃圾邮件} | W_1 \cap \overline{W_2} \cap \overline{W_3} \cap W_4) = \frac{\mathbb{P}(W_1 \cap \overline{W_2} \cap \overline{W_3} \cap W_4 | \text{垃圾邮件}) \times \mathbb{P}(\text{垃圾邮件})}{\mathbb{P}(W_1 \cap \overline{W_2} \cap \overline{W_3} \cap W_4)} \quad (4.3)$$

有很多原因使这个公式在计算上非常困难, 但如果我们用朴素贝叶斯时间独立性的假设, 那么计算将会变得简单. 具体来说, 朴素贝叶斯假设类条件独立 (class-conditional independence), 这意味着只要事件在相同类取值的条件下, 这些事件就是独立的. 于是:

$$\mathbb{P}(\text{垃圾邮件} | W_1 \cap \overline{W_2} \cap \overline{W_3} \cap W_4) = \frac{\mathbb{P}(W_1 | \text{垃圾邮件}) \mathbb{P}(\overline{W_2} | \text{垃圾邮件}) \mathbb{P}(\overline{W_3} | \text{垃圾邮件}) \mathbb{P}(W_4 | \text{垃圾邮件}) \mathbb{P}(\text{垃圾邮件})}{\mathbb{P}(W_1) \mathbb{P}(\overline{W_2}) \mathbb{P}(\overline{W_3}) \mathbb{P}(W_4)} \quad (4.4)$$

类似得到非垃圾邮件的概率公式:

$$\mathbb{P}(\text{非垃圾邮件} | W_1 \cap \overline{W_2} \cap \overline{W_3} \cap W_4) = \frac{\mathbb{P}(W_1 | \text{非垃圾邮件}) \mathbb{P}(\overline{W_2} | \text{非垃圾邮件}) \mathbb{P}(\overline{W_3} | \text{非垃圾邮件}) \mathbb{P}(W_4 | \text{非垃圾邮件}) \mathbb{P}(\text{非垃圾邮件})}{\mathbb{P}(W_1) \mathbb{P}(\overline{W_2}) \mathbb{P}(\overline{W_3}) \mathbb{P}(W_4)} \quad (4.5)$$

利用似然表中的数据, 可以利用公式进行计算, 由于分母一样, 我们忽略它. 垃圾邮件的 likelihood 为 0.012, 非垃圾邮件的 likelihood 为 0.002. 所以我们可以认为该消息是垃圾邮件的可能性使非垃圾邮件的 6 倍.

然而将这些数值转换为概率还需要最后一步: 该消息是垃圾邮件的概率 = 该消息是垃圾邮件的似然除以该消息是垃圾邮件的似然与该校是非垃圾邮件的似然之和. i.e.

$$\frac{0.012}{0.012 + 0.002} = 0.857 \quad (4.6)$$

同理: 该消息是非垃圾邮件的概率 = 该消息是非垃圾邮件的似然除以该消息是垃圾邮件的似然与该校是非垃圾邮件的似然之和

$$\frac{0.002}{0.012 + 0.002} = 0.143 \quad (4.7)$$

因此给定该消息中四个单词出现的情况, 我们期望该消息是垃圾邮件的概率为 0.857, 不是的概率为 0.143

我们在前面的例子中使用的朴素贝叶斯分类算法可以总结为如下的公式。从本质上讲, 在给定特征 F_1 到 F_n 提供的证据的条件下, 类 C 中水平为 L 的概率等于每一条证据在类 C 的水平 L 下的条件概率的乘积, 再乘以类 C 的水平 L 的先验概率和尺度因子 $1/Z$, 尺度因子 $1/Z$ 将把上述结果转换为一个概率值:

$$P(C_L | F_1, \dots, F_n) = \frac{1}{Z} p(C_L) \prod_{i=1}^n p(F_i | C_L)$$

4.1.2.2 拉普拉斯估计

你会发现, 如果是另一封邮件, 正好包含了四个单词, 利用之前的贝叶斯算法, 我们计算出垃圾邮件的似然函数: $(4/20)(10/20)(0/20)(12/20)(20/100)=0$

非垃圾邮件的似然为: $(1/80)(14/80)(8/80)(23/80)(80/100)=0.00005$

显然非垃圾邮件的概率就是 100%. 这样的预测很可能没有意义. 对于类中一个或多个水平, 如果一个事件从来没有发生过, 就有可能出现这样的问题. 例如, *groceries* 没在以前的垃圾邮件中出现过, 那在有 *groceries* 的条件下该邮件为垃圾邮件的概率为 0.

这个问题的解决涉及使用一种叫 **Laplace estimator** 的方法. 拉普拉斯估计本质上是给频率表中的每个计数加上一个较小的数, 保证了每一类中每个特征发生的概率不会为 0. 通常情况下, 拉普拉斯估计中加上的数值设定为 1, 这样就保证了每一类特征的组合至少在数据中出现一次.

利用拉普拉斯修改后:

垃圾邮件的似然函数: $(5/24)(11/24)(1/24)(13/24)(20/100)=0.0004$

非垃圾邮件的似然为: $(2/84)(15/84)(9/84)(24/84)(80/100)=0.0001$

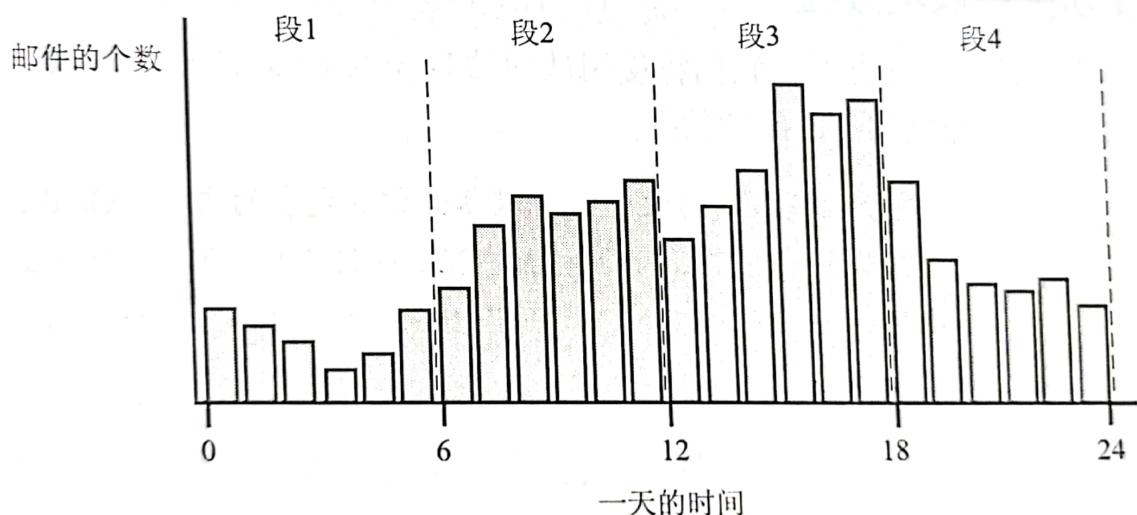
表名该消息是垃圾邮件的概率为 0.8. 显然更合理啦.

4.1.2.3 在朴素贝叶斯算法中使用数值特征

由于在朴素贝叶斯算法使用学习数据的频率表, 所以为了创建类和特征值的组合所构成的矩阵, 每个特征必须是分类变量. 因为数值型特征没有类别值, 所以之前的算法不能直接应用于数值型数据. 然而有一些方法可以解决这个问题.

一个简单而有效的方法就是将数值型特征值离散化, 这就意味着将数值分到不同分段 (bin) 中, 这些分段就是离散化的类的水平. 基于这个原因, 有时候离散化也称为分段.

另外还有几种不同的方法可以将数值型特征离散化. 最常见的方法就是探索于自然分类的数据或者数据分布中的分割点. 如你将垃圾邮件的发送时间作为特征:



图中的虚线即为分割点.

如果没有很明显的分割点, 一种选择就是利用分位数将数值型特征离散化. 但要记住, 将数值特征离散化总是会导致信息量的减少, 因为特征的原始粒度减少为几个数目较少的类别. 在处理这个问题, 重要的是取得平衡.

4.2 例子—基于贝叶斯算法的收集垃圾短息过滤

背景: 广告商利用短信服务 (SMS) 文本信息, 以潜在消费者为目标, 给他们发送不需要的广告. 由于短信字数限制 160, 所以可以用来确定一条消息是否是垃圾消息的文本量减少了, 而且这也导致很多人采用短信术语简写, 进一步模糊合法消息和垃圾消息的界限.

4.2.1 第一步—收集数据

本次使用的数据集包含短息的文本信息, 而且带有表明该短信是否为垃圾短信的标签, 垃圾为 spam, 非垃圾为 ham.

我们朴素贝叶斯分类器将利用词频中这种模式的优势来确定短信消息是更像垃圾短信还是非垃圾短信. 尽管可以想象单词 free 可以出现在非垃圾短信中, 但是一条合法的短信很可能根据上下文提供额外的单词信息. 例如: 一条非垃圾短信可能有 are you free on sunday? 而一条垃圾短信可能是用 free ringtones

朴素贝叶斯分类器将根据短信中所有的单词提供的证据计算垃圾短信的非垃圾短信的概率.

4.2.2 第二步—探索和准备数据

构建分类器的第一步涉及原始数据的处理和分析, 文本数据的准备具有挑战性, 因为将词和句子转化为计算机能够理解的形式是很重要的. 我们将数据转化为一种称为词袋的表示方法, 这种表示方法忽略了单词出现的顺序, 只是简单的提供一个变量用来表示单词是否会出现.

```
1 > sms_raw<-read.csv("D:/Program Files (x86)/RStudio/machinelearning/04.csv",
2                         stringsAsFactors = FALSE)
3 > str(sms_raw)
4 'data.frame': 5559 obs. of 2 variables:
5 $ type: chr "ham" "ham" "ham" "spam" ...
6 $ text: chr "Hope you are having a good week. Just checking in"...
```

将分类变量 type 转换为因子:

```
1 > sms_raw$type<-factor(sms_raw$type)
```

用函数 str() 和 table() 研究研究 type:

```
1 > str(sms_raw$type)
2 Factor w/ 2 levels "ham","spam": 1 1 1 2 2 1 1 1 2 1 ...
3 > table(sms_raw$type)
4
5   ham  spam
6   4812  747
```

4.2.3 数据准备—处理和分析文本数据

短信由词, 空格, 数字, 标点组成的文本字符串, 处理这种类型的复杂数据非常麻烦, 比如需要思考如何去除非数字和标点和没啥用的单词 eg.and but or... 还好 R 有个 tm 包专门处理它.

```
1 > library(tm)
2 载入需要的程辑包： NLP
3 > library(NLP)
```

处理文本数据的第一步涉及创建一个语料库, 即一个文本文件的集合. 本例中, 一个文本文件即是一条短信.

```
1 > sms_corpus <- Corpus(VectorSource(sms_raw$text))
```

函数 Corpus() 创建了一个 R 对象来存储文本文档, 这个函数通过一个参数来指定所加载的文本文档的格式. 因为我们已将短信读入 R 并存储在一个向量中, 所以我们用函数 VectorSource() 来指示函数 Corpus() 是用向量 sms_raw\$text 的信息. 函数 Corpus() 将结果存储在一个名为 sms_corpus 的对象中.

函数 Corpus() 很灵活, 可以读入很多不同来源的文档如 pdf, word

用 print() 输出我们刚刚创建的语料库:

```
1 > print(sms_corpus)
2 <<SimpleCorpus>>
3 Metadata: corpus specific: 1, document level (indexed): 0
4 Content: documents: 5559
```

可查看语料库的内容, 用 inspect(), 如看前三条短信:

```
1 > inspect(sms_corpus[1:3])
2 <<SimpleCorpus>>
3 Metadata: corpus specific: 1, document level (indexed): 0
4 Content: documents: 3
5
6 [1] Hope you are having a good week. Just checking in
7 [2] K..give back my thanks.
8 [3] Am also doing in cbe only. But have to pay.
```

在将文本内容分解为单词之前, 需要进行一些清理步骤以去除标点和其他一些可能影响结果的其他字符. 如将单词 hello!, HELLO 和 Hello 都作为 hello 的实例.

函数 tm_map() 提供了一种用来转换 (即映射)tm 语料库的方法, 我们将使用一系列转换函数清理语料库.

```
1 corpus_clean <- tm_map(sms_corpus, tolower)
2 corpus_clean <- tm_map(corpus_clean, removeNumbers)
3 corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())
4 corpus_clean <- tm_map(corpus_clean, removePunctuation)
5 corpus_clean <- tm_map(corpus_clean, stripWhitespace)
```

tolower: 将所有大写字母变成小写;

removeNumbers: 去除所有数字;
removeWords, stopwords(): 去除填充词如 to and or... 使用 tm 包中的 stopwords(), 这个函数包含了大量停用词;
removePunctuation: 去除标点符号;
stripWhitespace: 去除了以上的东西, 就会留下空格, 我们要去除多余的空格使得每个词之间只有一个空格.

注意: 敲完以上 5 行代码都出现了以下状况:

Warning message:

.....
transformation drops documents

該代碼應該仍然有效。您得到警告, 而不是錯誤。僅當使用語料庫而不是 VCorpus 時, 只有結合使用 VectorSource 的語料庫時, 才會出現此警告。

原因是在基礎代碼中進行了檢查, 以查看語料庫內容的名稱數量是否與語料庫內容的長度匹配。將文本作為矢量讀取時, 沒有文檔名稱, 並且會彈出此警告。這只是一個警告, 沒有文檔被丟棄。

于是修改了前面的 Corpus 为 VCorpus.

```
1 > inspect(sms_corpus[1:3])
2 <<VCorpus>>
3 Metadata: corpus specific: 0, document level (indexed): 0
4 Content: documents: 3
5
6 [[1]]
7 <<PlainTextDocument>>
8 Metadata: 7
9 Content: chars: 49
10
11 [[2]]
12 <<PlainTextDocument>>
13 Metadata: 7
14 Content: chars: 23
15
16 [[3]]
17 <<PlainTextDocument>>
18 Metadata: 7
19 Content: chars: 43
20
21 > inspect(corpus_clean[1:3])
22 <<VCorpus>>
23 Metadata: corpus specific: 0, document level (indexed): 0
```

```

24 Content: documents: 3
25
26 [[1]]
27 [1] hope good week just checking
28
29 [[2]]
30 [1] kgive back thanks
31
32 [[3]]
33 [1] also cbe pay

```

最后的步骤就是通过一个所谓的标记化过程将消息分解成由几个单词组成的组. 一个记号 (token) 就是一个文本字符串的单个元素, 本例中的记号就是单词.

tm 包提供了标记短信语料库的功能. 函数 DocumentTermMatrix() 将一个语料库作为输入, 创建一个称为稀疏矩阵的数据结构, 其中矩阵的行表示文档, 即短信, 列表示单词. 每个单元存储一个数字, 表示由列标识的单词出现在由行所表示的文档中的次数.

```

1 > sms_dtm <- DocumentTermMatrix(corpus_clean)
2 Error in UseMethod("meta", x) : "meta" 没有适用于"character"目标对象的方法

```

这里报错了, corpus_clean 的数据结构没有满足 DocumentTermMatrix() 的代入要求. 在 tm 0.6.0 之后 tm_map() 不再返回 TextDocument, 而是返回 character.

此时需要借助一个函数 PlainTextDocument()

```

1 > corpus_clean <- tm_map(corpus_clean, PlainTextDocument)
2 > sms_dtm <- DocumentTermMatrix(corpus_clean)
3 > sms_dtm
4 <<DocumentTermMatrix (documents: 5559, terms: 7893)>>
5 Non-/sparse entries: 42607/43834580
6 Sparsity : 100%
7 Maximal term length: 40
8 Weighting : term frequency (tf)

```

4.2.3.1 数据准备—建立训练数据集和测试数据集

因为短信的排序是随机的, 所以我们可以简单地取前 4169 条短信用于训练, 剩下的 1390 条短信用于测试.

先分解原始数据框, 然后输出文档-单词矩阵, 最后再分解语料库:

```

1 > sms_raw_train <- sms_raw[1:4169, ]
2 > sms_raw_test <- sms_raw[4170:5559, ]
3 >

```

```

4 > sms_dtm_train <- sms_dtm[1:4169, ]
5 > sms_dtm_test <- sms_dtm[4170:5559, ]
6 >
7 > sms_corpus_train <- corpus_clean[1:4169]
8 > sms_corpus_test <- corpus_clean[4170:5559]

```

为了确认上述子集是一组完整的短信数据的代表, 可以通过比较垃圾短信在训练数据集和测试数据集中的比例:

```

1 > prop.table(table(sms_raw_train$type))
2 ham      spam
3 0.8647158 0.1352842
4
5 > prop.table(table(sms_raw_test$type))
6 ham      spam
7 0.8683453 0.1316547

```

4.2.3.2 可视化文本数据-词云

词云, 一种可视化地描绘单词出现在文本数据中频率地方式. wordcloud 包提供了 r 函数来创建这种图形. 我们将使用他来比较垃圾短信和非垃圾短信地词云, 有助于我们了解朴素贝叶斯短信过滤器是否有可能成功.

```

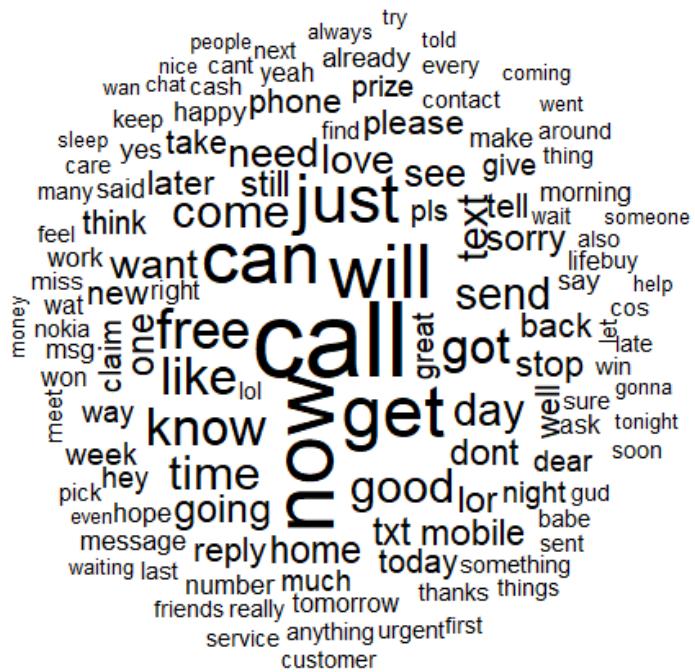
1 > library(wordcloud)
2 载入需要的程辑包: RColorBrewer
3 > library(RColorBrewer)
4 > wordcloud(sms_corpus_train, min.freq = 40, random.order = FALSE)

```

参数 random.order = FALSE 是设定词云以非随机地顺序排列, 出现频率越高的单词越靠近中心.

参数 min.freq = 40 是用来设定显示在词云中的单词必须满足在语料库中出现的最小次数. 通用规则是, 开始时设置参数 min.freq 的值为语料库中文档总数的 10%.

如果得到一条无法显示所有单词的警告, 就把参数 min.freq 的值设置更大一些, 从而减少词云中单词的数量. 另外, 使用参数 scale 设置字体大小也很好用.



由于我们没有对垃圾短信和非垃圾短信分别建立语料库, 这时就应该用函数 `wordcloud()` 一个非常有用的功能. 给定原始文本, 在建立语料库和显示词云之前, 它会自动应用文本转换过程.

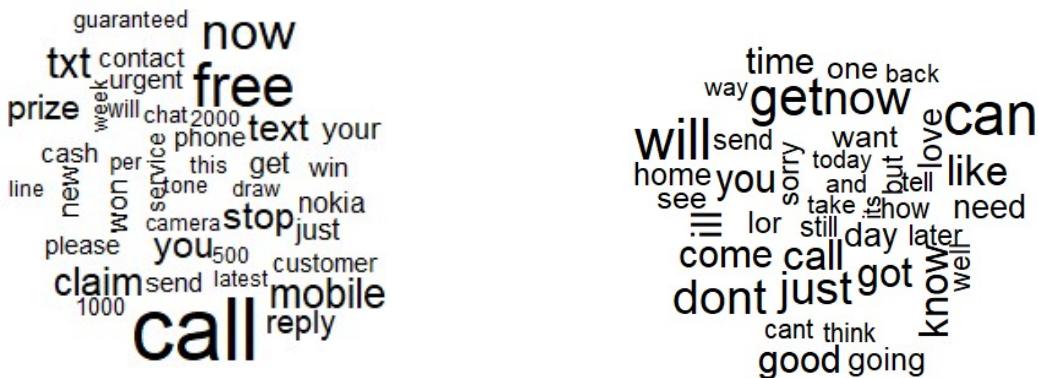
我们可以根据短信类型, 通过函数 `subset()` 来获取其中一种类型的短信数据:

```
1 > spam <- subset(sms_raw_train, type == "spam")
2 > ham <- subset(sms_raw_train, type == "ham")
```

注意使用双等号 `==`, 他表示相等

现在得到了两个短信数据框, 每一个都包含原始文本字符串的文本特征. 创建词云就像之前一样简单. 这一次我们使用参数 `max.words`, 用来显示两个集合的任何一个集合中最常见的 40 个单词, 参数 `scale` 可以调整词云中单词最大字体和最小字体.

```
1 > wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
2 > wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```



4.2.3.3 数据的准备—为频繁出现的单词创建指示特征

数据准备的过程的最后一步就是把稀疏矩阵转换成可用于训练朴素贝叶斯分类器的数据结构. 目前, 该数据结构超过 7000 个特征, 为了减少特征的数量, 我们将剔除训练数据中出现少于 5 条短信中或者少于记录总数的 0.1% 的所有单词.

tm 包中的 findFreqTerms() 函数, 输入一个文档-单词矩阵, 输出一个字符向量, 包含出现次数不少于指定次数的单词.

```

1 > findFreqTerms(sms_dtm_train, 5)
2 [1] "abiola"          "able"           "abt"           "accept"
3 [5] "access"          "account"        "across"        "activate"
4 .....
5 [993] "swing"          "system"         "take"          "takes"
6 [997] "taking"         "talk"           "talking"       "tampa"
7 [ reached getOption("max.print") -- omitted 213 entries ]

```

保存这个频繁出现的单词列表以备用:

```

1 > sms_dict <- findFreqTerms(sms_dtm_train, 5)

```

为了限制我们的训练矩阵和测试矩阵只包含前面筛选出来的单词, 我们使用以下命令:

```

1 > sms_train <- DocumentTermMatrix(sms_corpus_train, list(dictionary = sms_dict))
2 > sms_test <- DocumentTermMatrix(sms_corpus_test, list(dictionary = sms_dict))

```

朴素贝叶斯分类器通常是训练具有明确特征的数据, 因为稀疏矩阵中的元素表示一个单词出现在一条消息中的次数, 于是需要改变成为因子变量, 根据单词是否出现, 简单的标识为 yes 或者 no.

下面代码定义了一个函数 convert_counts(), 将计数转换为因子.

```

1 convert_counts <- function(x) {
2   x <- ifelse(x > 0, 1, 0)
3   x <- factor(x, levels = c(0, 1), labels = c("No", "Yes"))
4 }

```

现在我们只需要将 convert_counts 应用于稀疏矩阵的每一列.

apply() 函数是包括 lapply() 和 sapply 函数在内的函数族中的一员. 这些函数可以作用于 R 数据结构中的每一员, 而且这些函数是 R 语言的关键词之一. 经验丰富的 R 程序开发人员使用这些函数而不使用循环.

apply() 函数允许一个函数作用于一个矩阵的每一行或者每一列, 参数 MARGIN 来指定作用的对象是行还是列. 本例我们感兴趣列 (1 行 2 列).

```
1 > sms_train <- apply(sms_train, MARGIN = 2, convert_counts)
2 > sms_test <- apply(sms_test, MARGIN = 2, convert_counts)
```

4.2.4 第三步-基于数据训练模型

我们采用 e1071 包中的朴素贝叶斯算法实现, 这个包含有用于机器学习的多种函数.

朴素贝叶斯分类语法

应用 e1071 添加包中的函数 naiveBayes()

创建分类器:

```
m <- naiveBayes(train, class, laplace = 0)
• train: 数据框或者包含训练数据的矩阵
• class: 包含训练数据每一行的分类的一个因子向量
• laplace: 控制拉普拉斯估计的一个数值 (默认为 0)
```

该函数返回一个朴素贝叶斯模型对象, 该对象能够用于预测。

进行预测:

```
p <- predict(m, test, type = "class")
• m: 由函数 naiveBayes() 训练的一个模型
• test: 数据框或者包含测试数据的矩阵, 包含与用来建立分类器的训练数据相同的特征
• type: 值为 “class” 或者 “raw”, 标识预测是最可能的类别值或者原始的预测概率
该函数将返回一个向量, 根据参数 type 的值, 该向量含有预测的类别值或者原始预测的概率值。
```

例子:

```
sms_classifier <- naiveBayes(sms_train, sms_type)
sms_predictions <- predict(sms_classifier, sms_test)
```

```
1 library(e1071)
2 sms_classifier <- naiveBayes(sms_train, sms_raw_train$type)
3
4 sms_test_pred <- predict(sms_classifier, sms_test)
```

比较预测值和真实值. 这一次我们增加了一些参数来去除一些不必要的元素的比例, 以及使用参数 dnn(维度名称) 来重新标记行和列.

```
1 > library(gmodels)
2 > CrossTable(sms_test_pred, sms_raw_test$type,
3 +             prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
4 +             dnn = c('predicted', 'actual'))
```

```

5
6 Cell Contents
7 |-----|
8 | N |
9 | N / Col Total |
10|-----|
11
12 Total Observations in Table: 1390
13
14 | actual
15 predicted | ham | spam | Row Total |
16|-----|-----|-----|-----|
17 ham | 1202 | 32 | 1234 |
18 | 0.996 | 0.175 | |
19|-----|-----|-----|-----|
20 spam | 5 | 151 | 156 |
21 | 0.004 | 0.825 | |
22|-----|-----|-----|-----|
23 Column Total | 1207 | 183 | 1390 |
24 | 0.868 | 0.132 | |
25|-----|-----|-----|-----|

```

4.2.5 第五步-提升模型的性能

设置 laplace=1:

```

1 > sms_classifier2 <- naiveBayes(sms_train, sms_raw_train$type, laplace = 1)
2 > sms_test_pred2 <- predict(sms_classifier2, sms_test)
3 > CrossTable(sms_test_pred2, sms_raw_test$type,
4 +     prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
5 +     dnn = c('predicted', 'actual'))
6
7 Cell Contents
8 |-----|
9 | N |
10| N / Col Total |
11|-----|
12
13 Total Observations in Table: 1390
14
15 | actual
16 predicted | ham | spam | Row Total |

```

17					
18	ham	1204	31	1235	
19		0.998	0.169		
20					
21	spam	3	152	155	
22		0.002	0.831		
23					
24	Column Total	1207	183	1390	
25		0.868	0.132		
26					

Chapter 5

分而治之—应用决策树和规则进行分类

本章介绍两种机器学习方法—决策树和规则学习. 这两种方法应用一个类似的策略: 通过将数据集划分为较小的自己来确定用于预测的方式, 然后以逻辑结构的形式呈现, 不需要任何统计知识就可以理解. 这方面使得这些模型对改进企业战略和业务流程特别有用.

你将会学习:

- 每一种方法是采用什么策略将数据划分成令人感兴趣的类别的.
- 决策树和规则分类的几种实现方法, 包括 C5.0 算法, 1R 算法和 RIPPER 算法.
- 如何使用这些算法进行现实世界的分类任务, 如确定高风险银行贷款.

我们首先研究决策树, 其次研究规则分类.

5.1 理解决策树

决策树学习算法以树形结构建立模型. 类似于流程图, 该模型本身包含一系列逻辑决策, 带有表示不过根据某一属性做出决定的决策节点. 从这些决策节点引出的分支表示可做出的选择, 决策树由叶节点 (leaf notes, 也称为终端节点) 终止, 叶节点表示遵循决策组合的结果.

数据的分类从根节点开始, 根据特征值遍历树上的各个决策点. 数据采用的是一个漏斗形的路径, 它将每一条记录汇集到一个叶节点, 在叶结点为该记录分配到一个预测类.

因为决策树本质就是一个流程图, 所以决策树特别适合应用于由于法律因素需要透明化的分类机制以及为了便于决策需要共享结果的分类.

潜在用途:

- 信用评估模型中, 其中导致申请被拒绝的准则必须明确规定;
- 客户流失或者客户满意度的市场调查将与管理机构或者广告公司共享.
- 基于实验测量, 症状或者疾病进展率的医疗条件诊断.

事实上, 决策树可能是最广泛使用的机器学习技术之一, 他几乎可以用关于任何类型的数据建模, 并且具有无与伦比的性能.

但是在一些案例中, 决策树可能不是一个理想的选择, 尤其是数据有的莱昂的多层次的名义特征, 或者数据有大量的数值特征, 这些案例可能生成一个数量庞大的决策和过于复杂的决策树.

5.1.1 分而治之

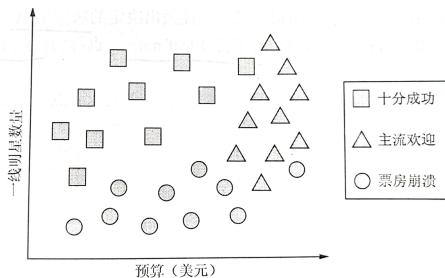
决策树的建立使用一种递归划分 (recursive partitioning) 的探索法. 这种方法又称为分而治之 (divide and conquer), 因为它利用特征的值将数据分解成具有相似类的较小的子集.

从代表整个数据集的根节点开始, 该算法选择最能预测目标类的特征, 然后这些案例将被划分到这一特征的不同值得组中, 这一决定形成了第一组树枝. 该算法继续分而治之其他节点, 每次选择最佳的候选特征, 直到达到停止的标准. 如果一个节点停止, 它可能具有下列情况:

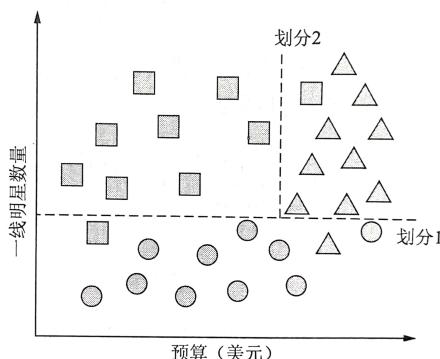
- 节点上没有 (几乎所有) 得案例都属于一类;
- 没有剩余得特征来分辨案例之间的区别;
- 决策树已经达到预先定义的大小限制.

为了说明决策树的建立过程, 我们来考虑一个简单的例子. 想象你正在一家好莱坞电影制片厂工作, 你的办公桌上堆满了剧本, 你决定研究一个决策树算法来预测一部有潜力的电影是否会落入受主流欢迎 (mainstream hit)、批评家的选择 (critic's choice) 和票房崩溃 (box office bust) 这三大类中, 而不是从头到尾把每个剧本读一遍.

为了收集数据来建立你的模型, 你转向工作室的档案去研究过去 10 年电影发行的情况. 在查看了 30 个不同电影剧本的数据后, 一个模型出现了, 在电影的拍摄预算中一线名人排队等候主演角色的数量与电影成功的类别之间似乎有一种关系. 该数据的散点图可能类似于下面这幅图:



为了使数据建立一个简单的决策树, 应用一个分而治之得策略. 首先将表示名人数量得特征进行划分 (划分 1), 其次在有很多名人的这组电影中, 根据电影预算的高低做另一个划分 (划分 2).

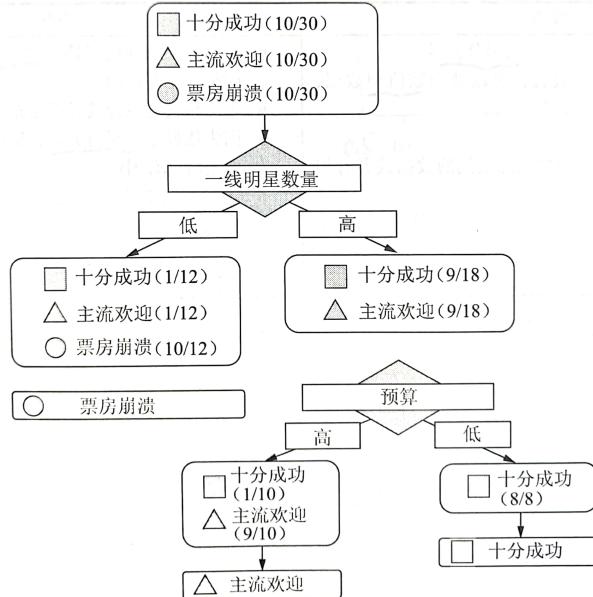


如果需要, 我们也可以继续划分区域, 根据越来越具体的预算范围和名人数量, 直到每个错误的分类值都分配到自己的分区 (或是很微小的分区). 由于数据可以继续划分, 直到在一个分区内的特征没有

明显区别, 所以决策树容易对训练数据进行过度拟合, 给出过于具体的决策. 我们将在这里停止分类算法以避免这种情况发生, 因为每组中超过 80% 的案例来自于同一个类.

你可能已经注意到, 对角线可能更清晰地划分数据. 这使决策树算法使用的轴平行分割来表现知识的局限性. 每分割一次只考虑一个特征地事实, 预防了决策树会由更复杂的决策形成, 如”如果名人的数量远大于估计的预算, 那么这部电影将会是一部很成功 (critical success 类) 的电影”

我们用于预测电影未来成功的模型可以用一个简单的决策树来表示, 如下图所示. 为了评估一个剧本, 可以依照每一个决策分支, 直到预测出它是成功的或者失败的.



由于现实世界中的数据包含的不仅仅是两个特征, 所以决策树很快就会变得比上图复杂. 下面你会学习一个自动建立决策树模型的算法.

5.1.2 C5.0 决策树算法

C5.0 算法是最知名的决策树算法之一, 已成为生成决策树的行业标准, 因为它确实适用于大多数类型的问题. 与其他先进的机器学习模型 (eg.chp7 黑箱方法-神经网络与支持向量机) 相比, 通过 C5.0 算法建立的决策树一般都表现的与其他先进模型几乎一样好, 而且更容易理解和部署. 此外该算法缺点轻微, 在很大程度上也可以避免:

优点:

- 通用;
- 高度自动化的学习过程, 可以处理数值型数据, 名义特征和缺失数据.
- 只使用最重要的特征
- 可以用于只有相对较少训练案例的数据或者有相当多训练案例的数据;
- 没有数学背景也可以解释一个模型结果 (对比较小的树))
- 比其他复杂模型更有效

缺点:

- 在根据具有大量水平的特征进行划分时往往有偏
- 容易过度拟合或者不能充分拟合模型;
- 因为依赖于轴平行分割, 所以对于一些关系建立模型时会有困难;
- 训练数据中的小变化可能导致决策逻辑的较大变化;
- 大的决策树可能很难理解, 给出的决策可能看起来有点违反直觉.

下面我们来详细说明如何使用分而治之的策略.

5.1.2.1 选择最佳的分割

决策树面临的一个挑战就是需要对数据的特征进行分割, 以前面的电影分类为例, 我们分的区中主要包含来源于一个单一类的案例. 如果一组数据中只包含一个单一类, 那么这些类被认为是纯的. 有许多用来衡量纯度的方法, 它们可以用来确定分割的标准.

C5.0 使用熵度量纯度, 样本数据的熵表示分类值如何混杂在一起, 如果样本完全同质, 则为 0; 而 1 表示样本凌乱的最大数量.

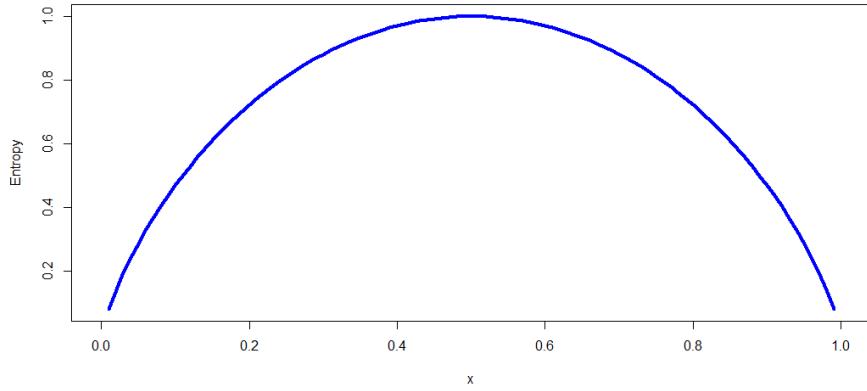
熵 (entropy):

$$Entropy(S) = \sum_{i=1}^c -p_i \log_2(p_i) \quad (5.1)$$

在熵的公式中, 对于给定的数据分割 (S), 常数 c 代表类的水平数, p_i 代表落入类中的水平 i 中的特征值的比例.

考察所有可能的两个类划分的熵. 如果我们知道在一个类中案例的比例为 x , 另一个就为 $1-x$, 使用函数 `curve()`, 就可以绘制关于 x 的所有可能值的熵的图像:

```
1 > curve(-x * log2(x) - (1 - x) * log2(1 - x),
2 +       col="blue", xlab = "x", ylab = "Entropy", lwd=4)
```



给定这种度量纯度的方式, 决策树算法仍然必须决定根据哪个特征进行分割. 根据这一点, 决策树算法使用熵值来计算由每一个可能特征的分割所引起的同质性 (均匀性) 变化, 该计算称为**信息增益**.

对于特征 F , 信息增益的计算方法是分割前的数据分区 (S_1) 的熵值减去由分割产生的数据分区 (S_2) 的熵值:

$$InfoGain(F) = Entropy(S_1) - Entropy(S_2) \quad (5.2)$$

复杂之处在于, 一次分割后, 数据被划分到多个分区中, 因此计算 $Entropy(S_2)$ 需要考虑所有分区熵值的总和. 这可以通过记录落入每一分区的比例来计算每一分区的权重:

$$Entropy(S_2) = \sum_{i=1}^n w_i Entropy(P_i) \quad (5.3)$$

信息增益越高, 根据某一特征分割后创建的分组越均衡. 如果最大信息增益等于分割前的熵值, 这意味着分割后的熵值为 0, 即决策结果是在完全同质的分组中.

前面的公式假定是名义特征, 但对于数值特征的分割, 决策树同样可以使用信息增益. 一个通常的做法是测试不痛的分割, 然后比较阈值, 将数值划分到不同的组中, 这将数值特征压缩到一个两水平的分类特征, 而且信息增益很容易计算, 选择产生最大信息增益的数字与之用于进行分割.

信息增益并不是构建决策树的唯一分割标准. 其他常用的标准有基尼系数 (Gini index), 卡方统计量 (Chi-Squared statistic), 增益比 (gain ratio)

基尼系数有一个直观的数学含义. 假设从总人口中随机抽签出两个人, 设他们的收入为 x_1 和 x_2 . 于是二人中较为富有的一个人收入为 $M = \text{Max}\{x_1, x_2\}$, 较为贫穷的一个人收入为 $m = \text{min}\{x_1, x_2\}$. 用概率论的语言, M 和 m 是随机抽签的这两个人收入的**次序统计量**. 从上述洛伦兹曲线的定义可以推导基尼系数满足

$$\text{Gini} = \frac{\langle M \rangle - \langle m \rangle}{\langle M \rangle + \langle m \rangle} = \frac{\langle |x_1 - x_2| \rangle}{2\mu}.$$

这里 $\langle \dots \rangle$ 表示取**数学期望**, 而 $\langle x_1 \rangle = \langle x_2 \rangle = \mu$ 为总体的人均收入. 用上式便于理解基尼系数的含义, 它与**变异系数** σ / μ 类似, 用来描述恒正分布的离散程度, 具有规模不变性. 例如, 把所有人的收入都乘以 2, 基尼系数是不变的. 而洛伦兹曲线的方法则更便于由收入五等分、十等分的实际调查统计数据计算出基尼系数的数值.

5.1.2.2 修剪决策树

如果决策树增长过大，将会使许多决策过于具体，模型将会过度拟合训练数据。

解决这个问题的一种方法就是一旦决策树达到一定数量的决策,或者决策节点仅含少量的案例,我们就停止树的增长,叫提前停止法/预剪枝决策法. 不足之处在于没有办法知道决策树是否会错过细微但很重要的模式,这种细微的模式只有决策树生长到足够大时才能学习到.

另一种方法是后剪枝决策法, 如果一棵决策树生长得过大, 就根据节点处的错误率使用修剪准则将决策树减小到更合适的大小. 该方法比预剪枝更有效, 因为如果没有事先生成决策树, 要确定一棵决策树生长的最佳程度是相当困难的.

C5.0 算法的优点在于自动修剪, 即他关注许多决策, 自动使用相当合理的默认值. 该算法的总体策略就是事后修剪决策树, 它先生成一个过度拟合训练数据的大决策树, 然后删除对分类误差影响不大的节点和分枝. 在某些情况下, 整个分支会被进一步向上移动或者被一些简单的决策所取代, 这两个移植分支的过程分别称为子树提升和子树替换.

5.2 例子—使用 C5.0 决策树识别高风险银行贷款

背景：通过电话和网络，将自动化的信用评分用于即时进行的信贷审批是很有可能的。本节使用 C5.0 决策树建立一个简单的信贷审批模型。我们也将看到如何调整模型的结果，从而使机构财务损失的误差最小化。

5.2.1 第一步—收集数据

使用的信贷数据集包括了 1000 个贷款的案例,一个用来表示贷款特征和贷款申请者特征的数值特征与名义特征的组合,一个类变量表示贷款是否陷入违约.

5.2.2 第二步—探索和准备数据

由于数据中大多数特征都是名义特征(变量), 所以我们将忽略 `stringasfactor` 选项, 默认为 `true`.

```
1 > credit<-read.csv("E:/机器学习/ml&r/Machine Learning with R/  
2                               chapter 5/credit.csv")  
3 > str(credit)  
4 'data.frame': 1000 obs. of 17 variables:  
5 $ checking_balance : Factor w/ 4 levels "< 0 DM", "> 200 DM", ...: 1 3 4 1 1 4 4 3  
6 $ months_loan_duration: int 6 48 12 42 24 36 24 36 12 30 ...  
7 $ credit_history : Factor w/ 5 levels "critical", "good", ...: 1 2 1 2 4 2 2 2 2  
8 $ purpose : Factor w/ 6 levels "business", "car", ...: 5 5 4 5 2 4 5 2 5 2  
9 $ amount : int 1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ...  
10 $ savings_balance : Factor w/ 5 levels "< 100 DM", "> 1000 DM", ...: 5 1 1 1 1 5 4  
11 $ employment_duration : Factor w/ 5 levels "< 1 year", "> 7 years", ...: 2 3 4 4 3 3 2  
12 $ percent_of_income : int 4 2 2 2 3 2 3 2 2 4 ...  
13 $ years_at_residence : int 4 2 3 4 4 4 4 2 4 2 ...  
14 $ age : int 67 22 49 45 53 35 53 35 61 28 ...  
15 $ other_credit : Factor w/ 3 levels "bank", "none", ...: 2 2 2 2 2 2 2 2 2 2
```

```

16 $ housing           : Factor w/ 3 levels "other","own",...: 2 2 2 1 1 1 2 3 2 2 ...
17 $ existing_loans_count: int 2 1 1 1 2 1 1 1 1 2 ...
18 $ job               : Factor w/ 4 levels "management","skilled",...: 2 2 4 2 2 4 ...
19 $ dependents        : int 1 1 2 2 2 2 1 1 1 1 ...
20 $ phone              : Factor w/ 2 levels "no","yes": 2 1 1 1 1 2 1 2 1 1 ...
21 $ default            : Factor w/ 2 levels "no","yes": 1 2 1 1 2 1 1 1 1 2 ...

```

现在看看用 `table()` 对贷款的一对特征输出的一些结果, 似乎很有可能预测出违约者. 特征 (变量)`checking_balance` 和 `saving_balance` 表示支票和储蓄账户余额, 并记录为分类变量:

```

1 > table(credit$checking_balance)
2 < 0 DM      > 200 DM 1 - 200 DM      unknown
3 274          63          269          394
4
5 > table(credit$savings_balance)
6 < 100 DM      > 1000 DM 100 - 500 DM 500 - 1000 DM      unknown
7 603          48          103          63          183

```

本数据为德国的数据, 因此以德国马克 (deutsche mark) 为单位. 较大的支票和储蓄账户余额应该与较小的贷款违约可能性相联系.

有些贷款的特征是数值型变量, 如贷款期限 (`months_loan_duration`) 和信贷申请的金额 (`amount`)

```

1 > summary(credit$months_loan_duration)
2 Min. 1st Qu. Median Mean 3rd Qu. Max.
3 4.0    12.0   18.0  20.9   24.0   72.0
4 > summary(credit$amount)
5 Min. 1st Qu. Median Mean 3rd Qu. Max.
6 250    1366   2320  3271   3972   18424

```

变量 `default` 表示贷款申请者是否未能符合约定的付款条件而陷入违约. 所有申请贷款的有 30% 陷入违约:

```

1 > table(credit$default)
2 no yes
3 700 300

```

5.2.2.1 数据准备—创建随机的训练数据集和测试数据集

我们将使用 90% 的数据作为训练数据, 10% 的数据作为测试数据, 但是此次的数据并非随机排列, 因此在划分数据之前, 通过随机排列信贷数据框来解决问题.

`order()` 函数以升序或者降序的方式对数据列表进行重新排列, 如果我们将 `order()` 函数与生成一列随机数的函数相结合, 就可以生成一个随机排序的列表. 随机数用 `runif()` 函数, 该函数在默认情况下产生 0-1 之间的随机数序列.

```

1 > set.seed(12345)
2 > credit_rand <- credit[order(runif(1000)), ]

```

以上命令创建了一个随机排序的 credit 数据框, set.seed() 函数可以在一个预定义的序列里面生成随机数, 从一个称为随机数种子的位置开始 (此处设为任意值 12345). set.seed() 函数能够确保如果要重复这里的分析, 那么可以获得相同的结果.

runif() 生成了 1000 个随机数; order() 函数返回一个数值向量, 记录了排序之后的 1000 个随机数在原来序列中的位置. 我们利用这些位置来筛选 credit 数据框中的行.

为了更好的理解该函数的作用, 注意 order(c(0.25,0.5,0.75,0.1)) 返回的序列是 (4,1,2,3), 因为最小的数 0.1 在第四个, 第二小的数 0.25 在第一个, 以此类推.

为了确定我们得到的数据除了顺序不同, 其余一样, 我们看看:

```
1 > summary(credit$amount)
2 Min. 1st Qu. Median Mean 3rd Qu. Max.
3 250 1366 2320 3271 3972 18424
4 > summary(credit_rand$amount)
5 Min. 1st Qu. Median Mean 3rd Qu. Max.
6 250 1366 2320 3271 3972 18424
7 > head(credit$amount)
8 [1] 1169 5951 2096 7882 4870 9055
9 > head(credit_rand$amount)
10 [1] 1199 2576 1103 4020 1501 1568
```

划分数据:

```
1 > credit_train <- credit_rand[1:900, ]
2 > credit_test <- credit_rand[901:1000, ]
```

如果一切顺利, 那么每个数据集中约有三分之一的人违约:

```
1 > prop.table(table(credit_train$default))
2 no yes
3 0.7022222 0.2977778
4
5 > prop.table(table(credit_test$default))
6 no yes
7 0.68 0.32
```

5.2.3 第三步-基于数据训练模型

使用 C50 包中的 C5.0 算法来训练决策树模型.

对于我们信贷审批模型的第一次迭代, 我们将使用默认的 C5.0 配置, 如下面代码所示.

示。在 `credit_train` 中, 第 17 列是类变量 `default`, 所以我们需要将它作为一个自变量从训练数据框中排除, 但把它作为用于分类的目标因子向量。

```
> credit_model <- C5.0(credit_train[-17], credit_train$default)
```

C5.0 决策树语法

大写

C5.0 决策树只针对分类型变量

应用 C50 添加包中的函数 C5.0()

创建分类器:

```
m <- C5.0(train, class, trials = 1, costs = NULL)
• train: 一个包含训练数据的数据框
• class: 包含训练数据每一行的分类的一个因子向量
• trial: 为一个可选数值, 用于控制自助法循环的次数 (默认值为 1)
• costs: 为一个可选矩阵, 用于给出与各种类型错误相对应的成本
```

该函数返回一个 `C5.0` 模型对象, 该对象能够用于预测。

进行预测:

```
p <- predict(m, test, type = "class")
• m: 由函数 C5.0() 训练的一个模型
• test: 一个包含测试数据的数据框, 该数据框和用来创建分类器的训练数据有同样的特征。
• type: 取值为 "class" 或者 "prob", 标识预测是最可能的类别值或者是原始的预测概率
```

该函数将返回一个向量, 根据参数 `type` 的取值, 该向量含有预测的类别值或者原始预测的概率值。

例子:

```
credit_model <- C5.0(credit_train, loan_default)
credit_prediction <- predict(credit_model, credit_test)
```

```
1 > library(C50)
2 > credit_model <- C5.0(credit_train[-17], credit_train$default)
```

现在, 对象 `credit_model` 包含一个 C5.0 决策树对象:

```
1 > credit_model
2
3 Call:
4 C5.0.default(x = credit_train[-17], y = credit_train$default)
5
6 Classification Tree
7 Number of samples: 900
8 Number of predictors: 16
9
10 Tree size: 67
11
12 Non-standard options: attempt to group attributes
```

文字显示了一些关于该决策树的简单情况, 包括生成决策树的函数调用, 特征数 (predictors) 和用于决策树增长的案例 (samples). 同时列出树的大小为 67, 表明有 67 个决策.

查看决策:

```
1 > summary(credit_model)
2
3 Call:
4 C5.0.default(x = credit_train[-17], y = credit_train$default)
```

```

5
6 C5.0 [Release 2.07 GPL Edition]           Mon Oct 05 16:58:22 2020
7 -----
8
9 Class specified by attribute 'outcome' \\
10
11 Read 900 cases (17 attributes) from undefined.data\textbackslash{}\textbackslash
12
13 Decision tree:

```

```

1 checking_balance = unknown: no (358/44)
2 checking_balance in {< 0 DM,> 200 DM,1 - 200 DM}:
3 ....credit_history in {perfect,very good}:
4 ....dependents > 1: yes (10/1)
5 : dependents <= 1:
6 : ....savings_balance = < 100 DM: yes (39/11)
7 :       savings_balance in {> 1000 DM,500 - 1000 DM,unknown}: no (8/1)
8 :       savings_balance = 100 - 500 DM:
9 :         ....checking_balance = < 0 DM: no (1)
10:           checking_balance in {> 200 DM,1 - 200 DM}: yes (5/1)
11
12 .....

```

前四行还是可以用语言解读的..

1. 如果支票账户余额是未知的, 则归类为不太可能违约.
2. 否则, 如果支票账户余额少于 0 马克, 或者 1-200 马克之间, 或者超过 200 马克, 或者...
3. 信用记录非常好, 甚至完美, 或者...
4. 有多个受扶养人, 就归类为很有可能违约.

括号中的数字表示符合该决策准则的案例个数以及根据该决策不正确分类的案例的数量. 如第一行的 (358/44) 表示有 358 案例符合该决策条件, 有 44 个案例被错误的归类为 no, 也即是不太可能违约. 换句话说, 就是有 44 个申请者确实违约了, 尽管模型的预测与此相反.

决策树输出之后,summary(credit_model) 输出了一个混淆矩阵, 这是一个交叉列表, 表示模型对训练数据错误分类的记录数:

```

1 Evaluation on training data (900 cases):
2
3 Decision Tree
4 -----
5 Size      Errors
6

```

```

7 66 125(13.9%) <<
8
9 (a) (b) <-classified as
10 -----
11 609 23 (a): class no
12 102 166 (b): class yes

```

900 个案例中的 125 个被错误分类, 错误率 13.9%. 其中 23 个真实值为 no 的个案被错误地归为了 yes, 而有 102 个真实值为 yes 的个案被错误地归类为 no.

众所周知, 决策树有一种过度拟合训练数据模型的倾向. 由于这个原因, 训练数据中报告的错误率可能过于乐观, 因此, 仍需要基于测试数据来评估.

5.2.4 第四步-评估模型性能

```

1 > credit_pred <- predict(credit_model, credit_test)

2 > library(gmodels)
3 > CrossTable(credit_test$default, credit_pred,
4 +             prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
5 +             dnn = c('actual default', 'predicted default'))

6 Cell Contents
7 |-----|
8 |           N |
9 |           N / Table Total |
10|-----|
11
12 Total Observations in Table: 100
13
14           | predicted default
15 actual default |       no |       yes | Row Total |
16 |-----|-----|-----|-----|
17 no |       57 |       11 |       68 |
18 |       0.570 |       0.110 |       |
19 |-----|-----|-----|-----|
20 yes |       16 |       16 |       32 |
21 |       0.160 |       0.160 |       |
22 |-----|-----|-----|-----|
23 Column Total |       73 |       27 |       100 |
24 |-----|-----|-----|-----|

```

错误率 27%.

5.2.5 第五步—提高模型性能

5.2.5.1 提高决策树的准确性

C5.0 算法对 C4.5 算法改进的一种方法就是通过加入自适应增强 (adaptive boosting) 算法. 这是需索决策树构建的一个过程, 然后这些决策树通过投票表决的方法为每个案例选择最佳分类.

由于 boosting 算法可以更广泛地应用于任机器学习算法, 所以在本书的第 11 章种有该算法的详细信息. 就目前而言, 只需要说明 boosting 算法植根于这样一种概念: 通过将很多能力较弱的学习算法组合在一起, 就可以创建一个团队, 这比任何一个单独的学习算法都强得多, 优缺互补.

C5.0() 函数可以很轻松地将 boosting 算法添加到 C5.0 决策树中. 我们只需要添加一个额外的参数 trials 表示在模型增强团队中使用的独立决策树的数量.

参数 trials 设置了一个上限, 如果该算法识别出额外的试验似乎并没有提高模型的准确性, 那么它将停止添加决策树. 我们将开始于 10 个实验 (trials=10)——一个已经成为事实标准的数字, 研究表明这能降低关于测试数据约 25% 的错误率.

```
1 > credit_boost10 <- C5.0(credit_train[-17], credit_train$default,
2 +                         trials = 10)
3
4 > credit_boost10
5
6 Call:
7 C5.0.default(x = credit_train[-17], y = credit_train$default, trials = 10)
8
9 Classification Tree
10 Number of samples: 900
11 Number of predictors: 16
12
13 Number of boosting iterations: 10
14 Average tree size: 56
15
16 Non-standard options: attempt to group attributes
```

通过 10 次迭代, 决策树变小.

输入 summary(credit_boost10) 看到所有的 10 棵决策树, 训练数据的表现:

```
1 (a) (b) <-classified as
2 ---- ----
3 626    6    (a): class no
4 25     243   (b): class yes
```

只犯了 31 个错误, 错误率 3.4%.

让我们来看看测试数据:

```
1 > credit_boost_pred10 <- predict(credit_boost10, credit_test)
2 > CrossTable(credit_test$default, credit_boost_pred10,
3 +               prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
```

```

4      dnn = c('actual default', 'predicted default'))
5
6 Cell Contents
7 |-----|
8 |           N |
9 |   N / Table Total |
10|-----|
11
12 Total Observations in Table: 100
13
14           | predicted default
15 actual default |      no |      yes | Row Total |
16 |-----|-----|-----|-----|
17   no |      60 |      8 |      68 |
18   | 0.600 | 0.080 |      |
19 |-----|-----|-----|-----|
20   yes |      15 |      17 |      32 |
21   | 0.150 | 0.170 |      |
22 |-----|-----|-----|-----|
23 Column Total |      75 |      25 |      100 |
24 |-----|-----|-----|-----|

```

23% 的错误率.

错误率下降了 4%. 另一方面, 模型在预测贷款违约方面做的不好, 有 $15/32=47\%$ 是错误的. 缺乏更大的提高可能是因为我们的训练数据集太小了.

5.2.5.2 犯一些比其他错误更严重的错误

为了防止一颗决策树犯更严重的错误,C5.0 允许我们将一个惩罚因子分配到不同类型的错误上. 这些惩罚因子设定在一个代价矩阵中, 用来指定每种错误相对于其他任何错误有多少倍的严重性. 假设我们认为一个贷款违约者使银行损失了 4 倍:

```

1 > error_cost <- matrix(c(0, 1, 4, 0), nrow = 2)
2 > error_cost
3      [,1] [,2]
4 [1,]    0    4
5 [2,]    1    0

```

行名和列名中的代码 1 表 no,2 表 yes. 行表预测, 列表实际.

```

1 > credit_cost <- C5.0(credit_train[-17], credit_train$default,
2 +                      costs = error_cost)
3 Warning message:
4 no dimnames were given for the cost matrix; the factor levels will be used

```

```

5 > credit_cost_pred <- predict(credit_cost, credit_test)
6 >
7 > CrossTable(credit_test$default, credit_cost_pred,
8 +             prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,
9 +             dnn = c('actual default', 'predicted default'))
10
11 Cell Contents
12 |-----|
13 |           N |
14 |   N / Table Total |
15 |-----|
16
17 Total Observations in Table: 100
18
19           | predicted default
20 actual default |      no |      yes | Row Total |
21 |-----|-----|-----|-----|
22      no |      42 |      26 |      68 |
23      |      0.420 |      0.260 |      |
24 |-----|-----|-----|-----|
25      yes |       6 |      26 |      32 |
26      |      0.060 |      0.260 |      |
27 |-----|-----|-----|-----|
28  Column Total |      48 |      52 |      100 |
29 |-----|-----|-----|-----|

```

与最好的增强模型相比, 这个版本的模型整体上犯了很多错误: 相对于最好的增强模型的 23%, 这里为 32%. 然而错误的类型却相差很大.

5.3 理解分类规则

分类规则代表的是 if-else 逻辑语句形式的知识, 可用来对未标记的案例指定一个分类. 未标记的案例依据前件和后件的概念而指定, 而前件和后件就构成了一个假设, 即如果这种情况发生, 那么那种情况就会发生. 前件是由特征值的特定组合构成的, 而在满足规则的条件下, 后件描述用来指定的分类值.

规则学习经常以一种类似于决策树学习的方式被使用. 与决策树一样, 规则学习也可以用来为今后的行动形成认识, 比如:

- 确定导致机械设备出现硬件故障的条件;
- 描述人群的界定特征用于客户细分;
- 发现先于股票市场上股票价格大跌或大涨的情况.

另一方面,对于某些任务,规则学习相对于决策树有明显的优势.与决策树不同的是,决策树必须从上至下地应用,而规则是单独存在的事实.根据相同数据建立的模型,规则学习的结果往往比决策树地结果更简洁,更直接,更容易理解.

你将在本章地后面看到,可以使用决策树生成规则.那么为什么还要单独研究规则学习地算法呢?因为决策树会给任务带来一组特定的偏差,而规则学习可以通过直接识别规则而避免误差.

规则学习通常应用于以名义特征为主或全部是名义特征的问题.规则学习擅长识别偶发事件,即使偶发事件只是因为特征之间非常特殊的相互作用才发生的.

5.3.1 独立而治之

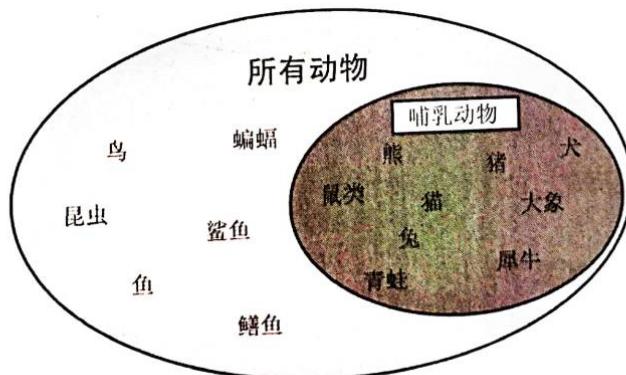
规则学习分类算法使用了一种称为独立而治之的探索法这个过程包括确定训练数据中覆盖一个案子例子集的规则,然后再从剩余的数据中分离出该分区.随着规则的增加,更多的数据子集会被分离,直到整个数据集都被覆盖,不再有案例残留.

分而治之和独立而治之之间的区别是很小的.或许,区分这两种方法的最佳方式就是考虑决策树中每个决策节点是会受到过去决策历史的影响.而规则学习不存在这样的沿袭,一旦规则算法分离出一组案例,下一组案例可能会根据完全不同的特征,以完全不同的顺序分离出.

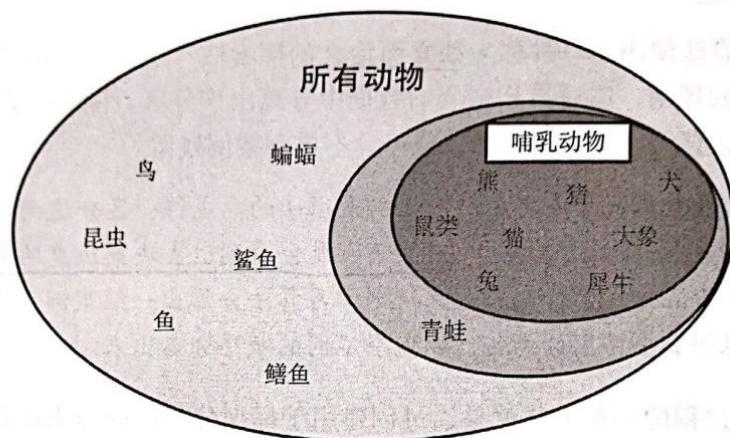
想象规则学习过程的一种方式就是通过创建用于标识分类值的越来越具体的规则来考虑向下挖掘(钻取)数据.假设任务是通过创建规则来确定一个动物是否是哺乳动物.你可以用一个大的空间来描述所有动物,如下图所示.



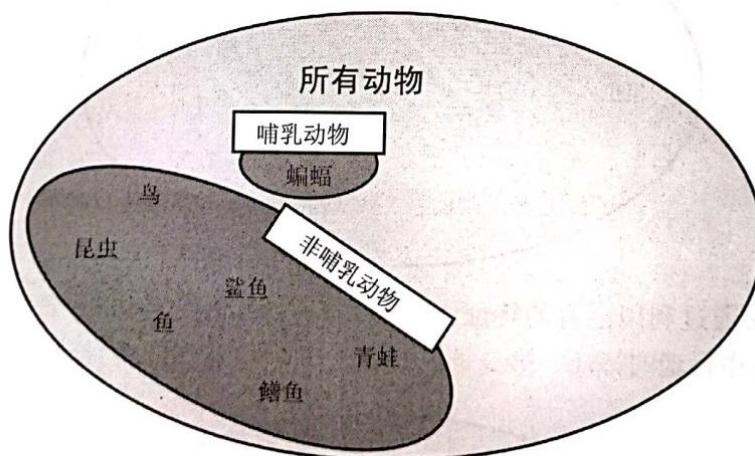
规则学习算法通过利用已有的特征来开始寻找同类群体.例如,根据衡量物种是在陆地、海洋,还是空中行走的特征,第一条规则可能表明任何陆地动物都是哺乳动物:



你是否注意到这条规则存在问题呢？如果仔细看，你可能会注意到青蛙是两栖动物，而不是哺乳动物。因此，规则需要更具体一点儿。让我们进一步细化规则，表明哺乳动物是在陆地上行走，并且有一条尾巴：



如上图所示，更具体的规则生成了一个全部是哺乳动物的子集。因此，该子集可以从其他数据中分离出来，而且可以通过定义额外的规则来确定剩余的哺乳动物蝙蝠。一个潜在的将蝙蝠从剩余的其他动物中区分出来的特征是其有皮毛。根据这个特征建立并利用这个规则，然后我们就能正确识别所有的动物：



这时，由于所有的训练案例都被分类了，所以规则学习过程就会停止。我们一共学习了3个规则：

- 在陆地上行走且有尾巴的动物是哺乳动物。
- 如果动物有皮毛，那么它就是哺乳动物。
- 否则，该动物不是哺乳动物。

前面的例子说明规则是如何逐步分离出越来越大的数据分区，最终将所有案例分类。因为数据的利用是基于先到先得的思想，所以分而治之和独立而治之算法又称为贪婪（学习）

由于规则看起来覆盖部分数据，所以独立而治之算法又称为覆盖算法，规则称为覆盖规则。下一节我们将研究一个简单的规则学习算法来学习覆盖规则在实际中是如何应用的。然后再研究一个复杂的，再解决现实问题。

5.3.2 单规则 (1R) 算法

设想一个均匀分成 10 块并涂色的轮盘, 其中 3 块红色, 3 块蓝色, 4 块白色. 只要预测和实际转到的颜色一样即获胜. 如果选白色, 你当然更易获胜, 因为这是轮盘上最常见的颜色. 很显然, 这个游戏很无聊, 但是它演示了最简单的分类器 **ZeroR**, 一个规则学习算法, 从字面上看没有规则 (因此得名). 对于每一个未标记的案例, 不用考虑它的特征值就会把它预测为最常见的类.

单规则算法 (1R 或 OneR) 通过选择一个单一的规则来提高 ZeroR 算法的性能.

优点:

- 可以生成一个单一的, 易于理解的, 人类可读的经验法则 (大拇指规则)
- 往往表现得出奇的好
- 可以作为更复杂算法的一个基准

缺点:

- 只使用了单一的特征
- 可能会过于简单

该算法运作的方式很简单. 对于每一个特征, 基于相似的特征值 1R 对数据分组. 然后对于每一个数据分组, 该算法的预测类为占大多数的类. 规则错误率的计算基于每一个特征, 犯最少错误的规则选为唯一的规则.

例子:

根据 **Travels By** (行走途径) 这一特征, 我们将数据分为 3 组: **Air** (空中)、**Land** (陆地) 和 **Sea** (海洋), 在 **Air** (空中) 组和 **Sea** (海洋) 组的动物被预测为非哺乳动物, 而 **Land** (陆地) 组的动物被预测为哺乳动物, 这样就导致了 2 个错误: 蝙蝠和青蛙. 根据 **Has Fur** (皮毛特征) 可以将动物分为两组, 有皮毛的动物被预测为哺乳动物, 而没有皮毛的动物被预测为非哺乳动物, 一共出现 3 个错误: 猪、大象和犀牛. 由于 **Travels By** (行走途径) 特征导致了更少的错误, 所以 1R 算法将会基于 **Travels By** (行走途径) 返回下面的“一个规则”:

- 如果该动物在空中行走, 那么它就不是哺乳动物。
- 如果该动物在陆地上行走, 那么它就是哺乳动物。
- 如果该动物在海洋中行走, 那么它就不是哺乳动物。

在发现了唯一的最重要的规则后, 该算法就会止于此。

很明显, 该算法对于某些任务来说可能过于简单了. 你想要一个只考虑单一症状的医疗诊断系统吗? 或者你想要一个只基于单一因素来停止或者加速车的自动驾驶系统吗? 对于这些类型的任务, 一个更复杂的规则学习算法可能会有用. 我们将在下一节中学习这种算法。

5.3.3 RIPPER 算法

早期的规则学习算法受到量问题的困扰:

1. 速度慢, 使得对于越来越多的大数据问题效率很低;

2. 对于噪声数据往往不准确.

解决这些问题的第一步是: **增量减少误差修剪算法 (incremental reduced error pruning,IREP)**, 它使用了生成复杂规则的预剪枝和后剪枝方法的组合, 并在将案例从全部收据集分离之前进行修剪. 虽然这种策略有助于提高规则学习算法的性能, 但往往还是决策树表现得更好.

随后规则算法又迈进了一步, **重复增量修剪算法 (repeated incremental pruning to produce error reduction,RIPPER)**, 它对 IREP 算法进行改进后再生规则, 它的性能与决策树相当, 甚至超越决策树.

规则学习分类算法的演变没有到此为止, 新的规则学习算法正快速的提出.

如下所示,RIPPER 规则学习算法的优点和缺点通常可与决策树比较, 其主要优点就是可能生成一个稍微精简的模型.

优点:

- 生成易于理解的, 人类可读的规则
- 对大数据和噪声数据集有效
- 通常比决策树产生的模型更简单

缺点:

- 可能会导致违反常理或者专家知识的规则;
- 处理数值型数据不够理想
- 性能有可能不如复杂的模型

RIPPER 算法是规则学习算法经过多次迭代进化而来的, 是用于规则学习的有效探索方法的一个组合. 由于该算法的复杂性, 所以其技术实现细节的讨论超出了本书范围, 但是可笼统地概括为三个步骤:

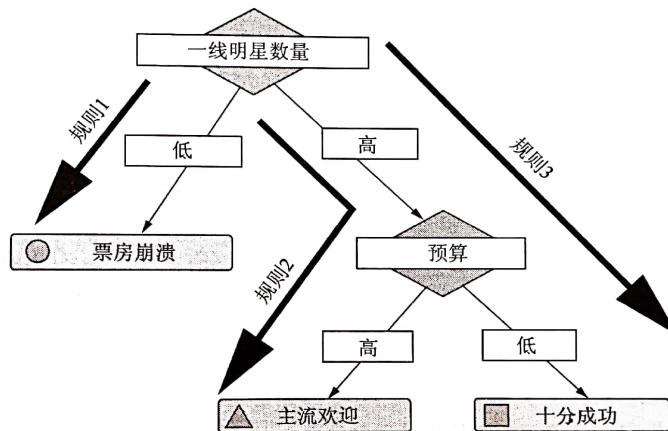
1. 生长
2. 修剪
3. 优化

生长过程利用独立而治之技术, 对规则贪婪地添加条件, 直到该规则能完全划分出一个数据子集或者没有属性可用于分割. 与决策树类似, 信息增益准则可以用来确定下一个分割属性, 直到达到一个停止准则, 然后使用各种探索法对整套规则进行优化.

RIPPER 算法的多因素规则比 1R 算法的规则更复杂, 这意味着 RIPPER 算法可以考虑多个属性.

5.3.4 来自决策树的规则

分类规则也可以直接从决策树获得。从一个叶节点开始沿着树枝回到树根，将获得一系列的决策，这些决策可以组合成一个单一的规则。下图显示了如何根据决策树构建规则来预测成功的电影。



沿着从根节点到每个叶子的路径，规则将是：票房

- 1) 如果名人的数量少，那么该电影将属于 Box Office Bust (票房崩溃) 类。
- 2) 如果名人的数量多且预算高，那么该电影将属于 Mainstream Hit (主流欢迎) 类。
- 3) 如果名人的数量多且预算低，那么该电影将属于 Critical Success (十分成功) 类。

使用决策树生成规则的主要缺点是由此产生的规则通常比那些由规则学习算法学到的规则更复杂。决策树应用分而治之策略产生的结果是有偏的，与规则学习产生的结果不同。另一方面，从决策树生成的规则有时候计算上会更有效。



当训练模型时，如果你指定 rules=TRUE，C5.0() 函数就可以利用分类规则生成一个模型。

5.4 例子—应用规则学习识别有毒的蘑菇

背景：如果有简单，清晰，一致的规则可用来识别有毒蘑菇，那么就可以拯救食物采集者的生命。由于规则学习算法的优势之一就是它们能生成易于理解的规则，所以规则学习算法似乎很合适这种分类任务。然而，只有在它们准确时才有用。

5.4.1 第一步—收集数据

该数据集包含了 23 个带菌褶的蘑菇品种的 8124 个蘑菇案例信息。在食用指南中，每种蘑菇被鉴定为“肯定可以食用”，“肯定是有毒的”，“可能有毒，不建议食用”。对于该数据集的目的而言，后两者合并为“肯定有毒”，故最终只有两个类：有毒和无毒。数据集中有 22 个蘑菇的特征。

5.4.2 第二步—探索和准备数据

由于所有特征都是名义变量，因此设置 `stringsAsFactors = TRUE`，并采用自动因子转换：

```
1 > mushrooms<-read.csv("E:/机器学习/ml&r/Machine Learning with R/chapter 5/2 mushrooms.csv",stringsAsFactors = TRUE)
```

```

1 > str(mushrooms)
2 'data.frame': 8124 obs. of 23 variables:
3 $ type                  : Factor w/ 2 levels "edible","poisonous": 2 1 1 2 1 1 1 ...
4 $ cap_shape              : Factor w/ 6 levels "bell","conical",..: 3 3 1 3 3 3 1 ...
5 $ cap_surface             : Factor w/ 4 levels "fibrous","grooves",..: 4 4 4 4 3 4 3 ...
6 $ cap_color               : Factor w/ 10 levels "brown","buff",..: 1 10 9 9 4 10 9 ...
7 $ bruises                : Factor w/ 2 levels "no","yes": 2 2 2 1 2 2 2 2 2 ...
8 $ odor                   : Factor w/ 9 levels "almond","anise",..: 8 1 2 8 7 1 1 ...
9 $ gill_attachment        : Factor w/ 2 levels "attached","free": 2 2 2 2 2 2 2 2 ...
10 $ gill_spacing            : Factor w/ 2 levels "close","crowded": 1 1 1 1 2 1 1 1 ...
11 $ gill_size               : Factor w/ 2 levels "broad","narrow": 2 1 1 2 1 1 1 1 ...
12 $ gill_color              : Factor w/ 12 levels "black","brown",..: 1 1 2 2 1 2 5 ...
13 $ stalk_shape              : Factor w/ 2 levels "enlarging","tapering": 1 1 1 1 2 1 ...
14 $ stalk_root               : Factor w/ 5 levels "bulbous","club",..: 3 2 2 3 3 2 2 ...
15 $ stalk_surface_above_ring: Factor w/ 4 levels "fibrous","scaly",..: 4 4 4 4 4 4 4 ...
16 $ stalk_surface_below_ring: Factor w/ 4 levels "fibrous","scaly",..: 4 4 4 4 4 4 4 ...
17 $ stalk_color_above_ring   : Factor w/ 9 levels "brown","buff",..: 8 8 8 8 8 8 8 8 ...
18 $ stalk_color_below_ring   : Factor w/ 9 levels "brown","buff",..: 8 8 8 8 8 8 8 8 ...
19 $ veil_type                : Factor w/ 1 level "partial": 1 1 1 1 1 1 1 1 1 ...
20 $ veil_color               : Factor w/ 4 levels "brown","orange",..: 3 3 3 3 3 3 3 ...
21 $ ring_number              : Factor w/ 3 levels "none","one","two": 2 2 2 2 2 2 2 ...
22 $ ring_type                : Factor w/ 5 levels "evanescent","flaring",..: 5 5 5 5 ...
23 $ spore_print_color        : Factor w/ 9 levels "black","brown",..: 1 2 2 1 2 1 1 ...
24 $ population               : Factor w/ 6 levels "abundant","clustered",..: 4 3 3 4 ...
25 $ habitat                  : Factor w/ 7 levels "grasses","leaves",..: 5 1 3 5 1 1 ...

```

注意其中有一行:

```

1 $ veil_type                : Factor w/ 1 level "partial": 1 1 1 1 1 1 1 1 1 ...

```

该因素变量就只有一个水平值. 数据字典列出了这个特征的两种水平:partial 和 universal, 然而我们对数据中的所有案例都分类为 partial, 很可能这个变量的有些编码是不正确的. 由于该变量的取值都一样, 不能为预测提供任何有用的信息, 那我们删除它:

```

1 > mushrooms$veil_type <- NULL

```

在进一步研究之前, 我们需要快速查看数据集中蘑菇类型这一类变量的分布. 如果分类水平的分布很不均匀 (这意味着它们严重失衡), 则有些模型如规则学习, 在预测少数类时会有困难.

```

1 > table(mushrooms$type)
2 edible  poisonous
3 4208      3916

```

为了达到这次实验的目的, 我们把蘑菇数据中的 8124 个案例看做一个所有可能的野生蘑菇完备集. 这是一个重要的假设, 因为这意味着我们不需要从训练数据中保存一些案例来达到测试的目的. 我

们没有尝试研究规则来覆盖不可预测的蘑菇类型, 我们只是试图找到能准确描绘已知蘑菇类型这一完备集的规则. 因此我们可以依据相同的数据来建立并测试模型.

5.4.3 第三步-基于数据训练模型

由于 ZeroR 忽略了所有的特征, 只是预测目标的模式, 所以它的规则可以说:”所有的蘑菇都是可使用的.” 显然不是很有用.

我们用 1R 分类器, 它能够识别对于目标类最具预测性的单一特征, 并利用该特征构建一个规则集. 我们将使用 RWeka 包里的 OneR() 函数来实现 1R 算法.

1R 分类规则语法	
应用 RWeka 添加包中的函数 OneR()	
创建分类器:	
<pre>m <- OneR(class ~ predictors, data = mydata)</pre>	
• class : 是 mydata 数据框中需要预测的那一列	
• predictors : 为一个公式, 用来指定 mydata 数据框中用来进行预测的特征	
• data : 为包含 class 和 predictors 所要求的数据的数据框	
该函数返回一个 1R 模型对象, 该对象能够用于预测。	
进行预测:	
<pre>p <- predict(m, test)</pre>	
• m : 由函数 OneR() 训练的一个模型	
• test : 一个包含测试数据的数据框, 该数据框和用来创建分类器的训练数据有同样的特征。	
该函数将返回一个含有预测的类别值的向量。	
例子:	
<pre>mushroom_classifier <- OneR(type ~ odor + cap_color, data = mushroom_train) mushroom_prediction <- predict(mushroom_classifier, mushroom_test)</pre>	

其公式语法使用运算符 `~` 表示一个目标变量和它的预测变量之间的关系. 需要学习的类变量放在左侧, 预测变量放在右侧, 用 `+` 分割. 如 $y \sim x_1 + x_2$, 如果你想在模型中包含所有变量, 可以使用专业术语”`~.`”, 如 $y \sim .$ 指定 y 和数据集中所有其他特征的关系.

```
1 > mushroom_1R <- OneR(type ~ ., data = mushrooms)
2
3 > mushroom_1R
4 odor: # 气味
5 almond -> edible # 杏仁味
6 anise -> edible # 茴香味
7 creosote -> poisonous # 木焦油
8 fishy -> poisonous # 腥味
9 foul -> poisonous # 臭味
10 musty -> poisonous # 霉味
11 none -> edible # 无气味
12 pungent -> poisonous # 刺鼻
```

```

13 | spicy    -> poisonous      # 辛辣
14 | (8004/8124 instances correct)

```

在输出的第一行, 我们看到特征 odor(气味) 被选为规则生成.

5.4.4 第四步-评估模型的性能

```

1 > summary(mushroom_1R)
2
3 === Summary ===
4
5 Correctly Classified Instances      8004          98.5229 %
6 Incorrectly Classified Instances   120           1.4771 %
7 Kappa statistic                   0.9704
8 Mean absolute error              0.0148
9 Root mean squared error          0.1215
10 Relative absolute error          2.958  %
11 Root relative squared error     24.323  %
12 Total Number of Instances        8124
13
14 === Confusion Matrix ===
15
16      a      b  <-- classified as
17 4208    0 |  a = edible
18 120  3796 |  b = poisonous

```

左下角的 120 表示有 120 种蘑菇实际上可以食用, 但被归类为了有毒; 另一方面, 没有有毒的蘑菇被错误的归类. 考虑到这个规则学习算法只使用了一个单一的特征, 我们确实做的不错.

5.4.5 第五步-提高模型性能

对于一个更复杂的规则学习算法, 我们将使用 JRip() 函数, 一个基于 Java 实现的 RIPPER 规则学习算法. 也在 RWeka 包.

```

1 > mushroom_JRip <- JRip(type ~ ., data = mushrooms)
2 > mushroom_JRip
3 JRIP rules:
4 =====
5
6 (odor = foul) => type=poisonous (2160.0/0.0)
7 (gill_size = narrow) and (gill_color = buff) => type=poisonous (1152.0/0.0)
8 (gill_size = narrow) and (odor = pungent) => type=poisonous (256.0/0.0)
9 (odor = creosote) => type=poisonous (192.0/0.0)
10 (spore_print_color = green) => type=poisonous (72.0/0.0)

```

```

11 (stalk_surface_below_ring = scaly) and (stalk_surface_above_ring = silky)
12 => type=poisonous (68.0/0.0)
13 (habitat = leaves) and (cap_color = white) => type=poisonous (8.0/0.0)
14 (stalk_color_above_ring = yellow) => type=poisonous (8.0/0.0)
15 => type=edible (4208.0/0.0)
16
17 Number of Rules : 9

```

JRip() 分类器从 mushrooms 数据中学习了 9 条规则. 理解这些规则的一种简单方法就是把它们当做类似于编程逻辑中 if-else 语句的一个列表.

前三条规则可以这样表达:

1. 如果气味是臭的, 那么该蘑菇是有毒的.
2. 如果菌褶尺寸狭小且颜色浅黄, 那么有毒
3. 如果菌褶尺寸狭小而且气味刺鼻, 那么有毒

最后第 9 条规则表示不属于上述 8 条规则的任何蘑菇案例都是可食用的. 每个规则后面的数字表示被规则覆盖的案例数和被错误分类的案例数.

RIPPER 分类规则语法	
应用 RWeka 添加包中的函数 JRip()	
创建分类器:	
m <- JRip(class ~ predictors, data = mydata)	
• class: 是 mydata 数据框中需要预测的那一列	
• predictors: 为一个 R 公式, 用来指定 mydata 数据框中用来进行预测的特征	
• data: 为包含 class 和 predictors 所要求的数据的数据框	
该函数返回一个 RIPPER 模型对象, 该对象能够用于预测。	
进行预测:	
p <- predict(m, test)	
• m: 由函数 JRip() 训练的一个模型	
• test: 一个包含测试数据的数据框, 该数据框和用来创建分类器的训练数据有同样的特征。	
该函数将返回一个含有预测的类别值的向量。	
例子:	
<pre> mushroom_classifier <- JRip(type ~ odor + cap_color, data = mushroom_train) mushroom_prediction <- predict(mushroom_classifier, mushroom_test) </pre>	

```

1 > summary(mushroom_JRip)
2
3 === Summary ===
4
5 Correctly Classified Instances      8124          100    %
6 Incorrectly Classified Instances      0            0    %

```

```

7 Kappa statistic 1
8 Mean absolute error 0
9 Root mean squared error 0
10 Relative absolute error 0 %
11 Root relative squared error 0 %
12 Total Number of Instances 8124
13
14 === Confusion Matrix ===
15
16 a b <-- classified as
17 4208 0 | a = edible
18 0 3916 | b = poisonous

```

=====
指定 rules=TRUE, 用 C5.0 分类规则生成模型:

```

1 > library(C50)
2 > mushroom_c5rules <- C5.0(type ~ odor + gill_size, data = mushrooms,
3                               rules = TRUE)
4 > summary(mushroom_c5rules)
5
6 Call:
7 C5.0(formula = type ~ odor + gill_size, data = mushrooms,
8       rules = TRUE)
9
10
11 C5.0 [Release 2.07 GPL Edition] Tue Oct 06 14:00:36 2020
12 -----
13
14 Class specified by attribute outcome'
  Read 8124 cases (3 attributes) from undefined.data

```

```

1 Rules:
2
3 Rule 1: (4328/120, lift 1.9)
4 odor in {almond, anise, none}
5 -> class edible [0.972]
6
7 Rule 2: (3796, lift 2.1)
8 odor in {creosote, fishy, foul, musty, pungent, spicy}
9 -> class poisonous [1.000]
10
11 Default class: edible

```

```
12
13
14 Evaluation on training data (8124 cases):
15
16 Rules
17 -----
18 No      Errors
19
20 2  120( 1.5%)  <<
21
22
23 (a)  (b)  <-classified as
24 -----
25 4208          (a): class edible
26 120  3796    (b): class poisonous
27
28
29 Attribute usage:
30
31 100.00% odor
32
33 Time: 0.0 secs
```

本章仅仅涉及如何应用决策树和规则的表面.

第 6 章介绍回归树和模型树的方法, 使用决策树预测数值型数据.

第 11 章将发现, 决策树的性能可以通过将他们组合在一起称为随机森林的模型而得到提高.

第 8 章将看到关联规则-分类规则的关联, 如何被用来识别交易数据的商品组.

Chapter 6

预测数值型数据—回归方法

在统计领域中,有很大一部分工作介绍用于估计数据元素之间这种数值关系的方法,其中一个研究领域称为回归分析. 这些方法可用于预测数值型数据以及量化预测结果与其预测变量之间关系的大小及强度.

你将学习:

- 用线性回归方法拟合数据方程的基本统计原则和它们如何描述数据元素之间的关系.
- 如何用 R 回归分析
- 如何使用称为回归树和模型树的混合模型,使得决策树可以用来预测数值型数据.

6.1 理解回归

回归分析主要关注确定一个唯一的因变量 (dependent variable)(需要预测的值) 和一个或者多个数值型的自变量 (independent variables)(预测变量). 我们首先假设因变量和自变量遵循一个线性关系.

回归方程使用类似类似斜截式的形式对数据建立模型. 该机器的工作就是确定 a 和 b , 从而使指定的直线最适合用来反映所提供的 x 值和 y 值之间的关系, 这可能不是完美的匹配, 所以该机器也需要一些方法来量化误差范围, 我们很快就会深入讨论这个.

回归分析通常用来对数据元素之间的复杂关系建立模型, 用来估计一种处理方法对结果的影响和推断未来. 具体案例包括:

- 根据种群和个体测得的特征, 研究他们之间如何不同 (差异性), 从而用于不同领域
- 量化事件及其相应的因果关系, 比如可应用于药物临床试验.
- 给定已知的准则, 确定可用来预测未来行为的模型, 比如用来预测保险赔偿.

回归方法也可用于假设检验, 其中包括数据是否能够表明原假设更可能是真还是假. 回归模型对关系强度和一致性的估计提供了信息用于评估结果是否是由于偶然性造成的.

回归分析是大量方法的综合体, 几乎可以应用于所有的机器学习任务.

在本章中, 我们只关注最基本的回归模型, 即那些使用回归直线的模型, 这叫做**线性回归模型 (linear regression)**. 如果只有单一一个自变量, 称为**简单线性回归 (simple linear regression)**. 否则叫**多元回归 (multiple regression)**, 这两个模型都是基于因变量连续.

对于其他类型的因变量, 即使是分类任务, 使用回归方法都是可能的. 如逻辑回归 (logistic regression) 可以用来对二元分类的结果建模. 泊松回归 (poisson regression) 可以用来对整型的计数数据建模.

相同的基本原则适用于所有的回归方法, 所以一旦你理解了线性情况下的回归方法, 你就可以研究其他的回归方法.

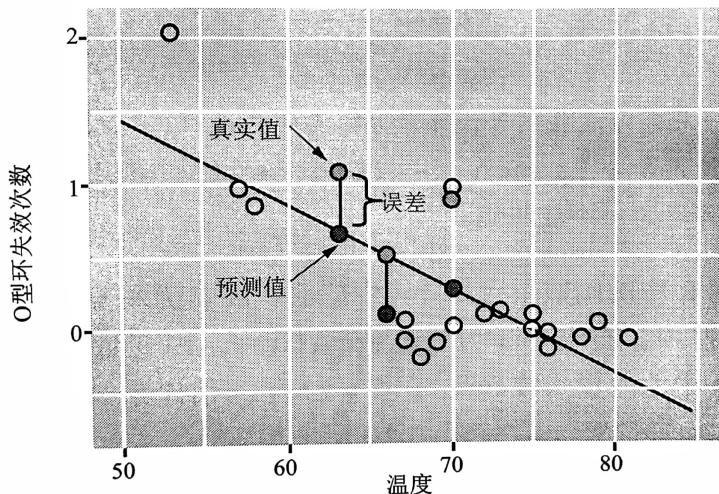
6.1.1 简单线性回归

简单线性回归定义了一个因变量和一个单一的自变量之间的关系:

$$y = \alpha + \beta x \quad (6.1)$$

6.1.2 普通最小二乘估计

为了确定 α 和 β 的最优估计值, 可使用普通最小二乘法 (ordinary least squares, OLS) 的估计方法. 在 OLS 回归中, 斜率和截距的选择要使得残差 (residual) 的平方和最小.



用数学术语, OLS 回归的目标可以表述为求下列方程最小值的任务:

$$\sum (y_i - \hat{y}_i)^2 = \sum e_i^2 \quad (6.2)$$

由普通最小二乘法得到的 \hat{b} 为

$$b = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} = \frac{Cov(x, y)}{Var(x)} \quad (6.3)$$

a 为

$$a = \bar{y} - b\bar{x} \quad (6.4)$$

用以上数据示例:

```

1 > launch<- read.csv("E:/机器学习/ml&r/Machine Learning with R/chapter 6
2   /challenger.csv")
3
4 > b<-cov(launch$temperature, launch$distress_ct)/var(launch$temperature)
5 > a<-mean(launch$distress_ct)-b*mean(launch$temperature)

```

用以上这种方式估计回归方程是不理想的, 所以 R 中理所当然地提供了可以自动估计回归方程的函数. 首先将通过学习一种用来衡量线性关系强度的方法来拓展我们对回归的理解, 然后我们将看一看线性回归如何应用于有多个自变量的数据中.

6.1.3 相关系数

Pearson 相关系数:

$$\rho_{x,y} = \text{Corr}(x, y) = \frac{\text{Cov}(x, y)}{\sigma_x \sigma_y} \quad (6.5)$$

```
> r<-cor(launch$temperature, launch$distress_ct)
```

度量两个变量之间的相关性给我们提供了一种快速了解因变量和自变量之间关系的方法, 当我们开始用大量的预测变量来定义回归模型时, 这变得越来越重要.

6.1.4 多元线性回归

优点:

- 数值型建模最常用
- 可适用于几乎所有的数据
- 提供了特征 (变量) 与结果之间关系的强度与大小的估计.

缺点:

- 对数据做出了很强的假设
- 该模型的形式必须由使用者事先指定
- 不能很好地处理缺失数据
- 只能处理数值特征, 所以分类数据需要额外的处理
- 需要一些统计知识来理解模型

多元回归方程:

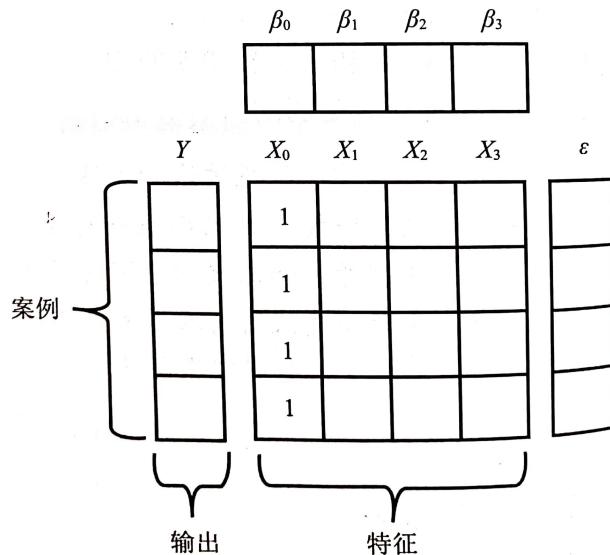
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_i x_i + \epsilon \quad (6.6)$$

$$Y = X\beta + \epsilon \quad (6.7)$$

目标是求出使得 y 的预测值与真实值之间的误差的平方和最小的 β

由最小二乘法可得向量 β 的最佳估计:

$$\hat{\beta} = (X^T X)^{-1} X^T Y \quad (6.8)$$



使用下面代码可以创建一个名为 reg 的简单回归函数, 输入参数为 y 和 x, 并返回一个估计的 β 系数矩阵.

```

1 > reg <- function(y, x) {
2 +   x <- as.matrix(x)
3 +   x <- cbind(Intercept = 1, x)
4 +   solve(t(x) %*% x) %*% t(x) %*% y
5 + }
```

注:

- 因为函数要使用来自数据框的列数据, 所以要用函数 as.matrix() 将数据强制变成矩阵.
- 函数 cbind() 用来将额外的一列添加到矩阵 x 中, 命令 Intercept = 1 指示 R 将一列新的命名为 Intercept, 并将这一列全部用数值 1 填充
- 函数 solve() 执行矩阵的逆运算
- 函数 t() 将矩阵转置
- %*% 将两个矩阵相乘

将 reg() 应用于航天飞机的数据:

```

1 > str(launch)
2 'data.frame': 23 obs. of 5 variables:
3 $ o_ring_ct : int 6 6 6 6 6 6 6 6 6 ...
4 $ distress_ct: int 0 1 0 0 0 0 0 1 1 ...
5 $ temperature: int 66 70 69 68 67 72 73 70 57 63 ...
6 $ pressure   : int 50 50 50 50 50 50 100 100 200 200 ...
7 $ launch_id  : int 1 2 3 4 5 6 7 8 9 10 ...
8
9 > reg(y = launch$distress_ct, x = launch[3])
```

```

10 [,1]
11 Intercept 4.30158730
12 temperature -0.05746032

```

这与我们之前算的 a,b 结果相同, 因此可以使用该函数建立多元回归模型. 但是这一次我们使用三列数据:

```

1 > reg(y = launch$distress_ct, x = launch[3:5])
2 [,1]
3 Intercept 3.814247216
4 temperature -0.055068768
5 pressure 0.003428843
6 launch_id -0.016734090

```

6.2 例子—应用线性回归预测医疗费用

背景: 利用分析的数据来预测这部分群体的平均医疗费用. 这些估计可以用来创建一个精算表, 根据预期的治疗费用来设定年度保费的价格高低.

6.2.1 第一步—收集数据

6.2.2 第二步—探索和准备数据

```

1 > insurance<-read.csv("E:/机器学习/ml&r/Machine Learning with R/chapter
2   6/insurance.csv",stringsAsFactors = TRUE)
3 > View(insurance)
4 > str(insurance)
5 'data.frame': 1338 obs. of 7 variables:
6 $ age      : int 19 18 28 33 32 31 46 37 37 60 ...
7 $ sex      : Factor w/ 2 levels "female","male": 1 2 2 2 2 1 1 1 2 1 ...
8 $ bmi      : num 27.9 33.8 33 22.7 28.9 ...
9 $ children: int 0 1 3 0 0 0 1 3 2 0 ...
10 $ smoker   : Factor w/ 2 levels "no","yes": 2 1 1 1 1 1 1 1 1 ...
11 $ region   : Factor w/ 4 levels "northeast","northwest",...: 4 3 3 2 2 3 3...
12 $ charges  : num 16885 1726 4449 21984 3867 ...

```

看看因变量 charges 的分布:

```

1 > summary(insurance$charges)
2 Min. 1st Qu. Median Mean 3rd Qu. Max.
3 1122      4740     9382    13270    16640    63770

```

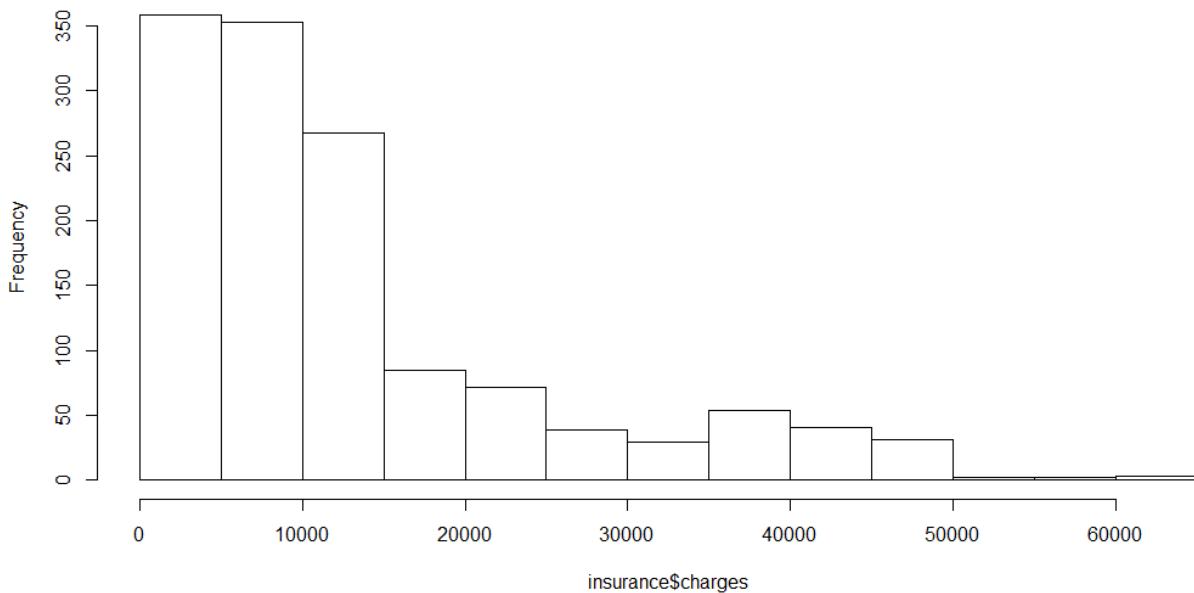
均值远大于中位数, 表明保险费用右偏:

```

1 > hist(insurance$charges)

```

Histogram of insurance\$charges



因为线性回归的假设因变量的分布是正态分布, 所以这种分布并不十分理想. 在实际应用中, 线性回归的假设往往会被违背. 如果有需要, 我们在后面能够修正该假设.

即将面临的另一个问题是, 回归模型需要每一个特征都是数值型的, 很快我们会了解 R 中线性回归函数如何处理.

sex 和 smoker 都是两个水平, region 是四个水平, 看看它的分布:

```
1 > table(insurance$region)
2
3
4
5
6
```

region	frequency
northeast	324
northwest	325
southeast	364
southwest	325

数据几乎均匀地分布在四个地理区域.

6.2.2.1 探索特征之间的关系—相关系数矩阵

```
1 > cor(insurance[c("age", "bmi", "children", "charges")])
2
3
4
5
6
```

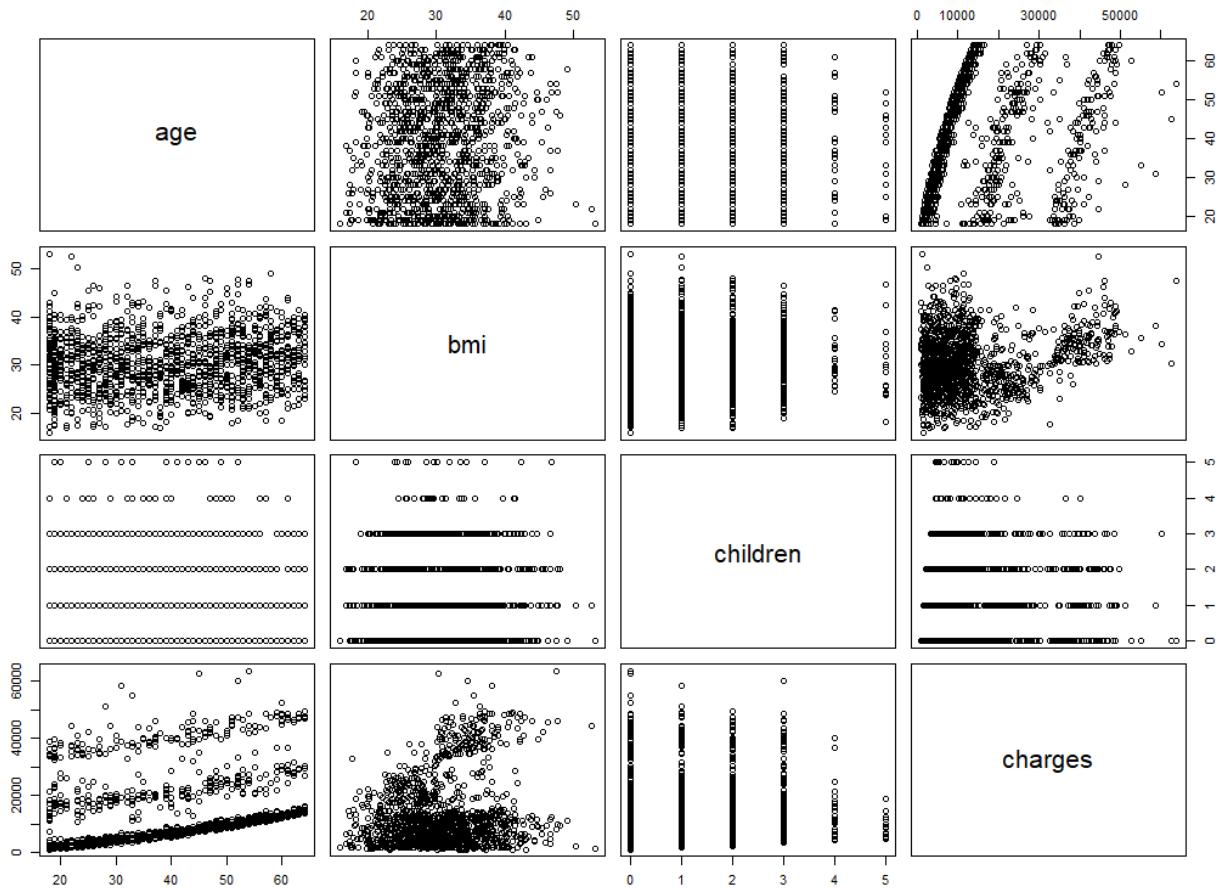
	age	bmi	children	charges
age	1.0000000	0.1092719	0.04246900	0.29900819
bmi	0.1092719	1.0000000	0.01275890	0.19834097
children	0.0424690	0.0127589	1.00000000	0.06799823
charges	0.2990082	0.1983410	0.06799823	1.00000000

该矩阵中的相关系数不是强相关的, 但还是存在一些显著的关联.

6.2.2.2 可视化特征之间的关系—散点图矩阵

用 R 默认的函数 pair() 来生成:

```
1 > pairs(insurance[c("age", "bmi", "children", "charges")])
```



尽管有些它看上去像是随机密布的点, 但有些还是呈现了某种趋势. age 和 charges 之间的关系呈现出几条相对的直线.

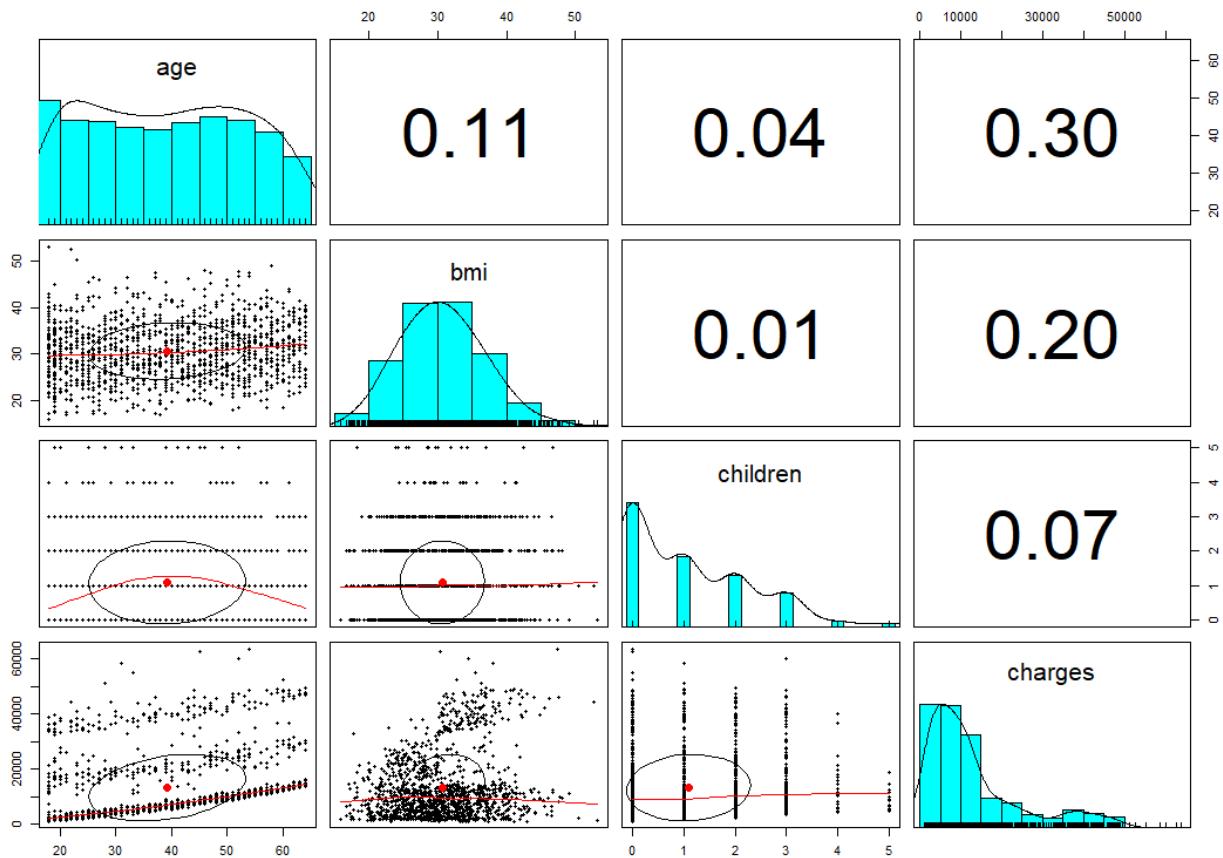
用 psych 包中的 pairs.panels() 函数创建散点图, 添加更多的信息:

```
1 > pairs.panels(insurance[c("age", "bmi", "children", "charges")])
```

在对角线上方, 散点图被相关系数矩阵取代; 在对角线上, 直方图描绘了每个特征的数值分布; 对角线下方的散点图有额外的可视化信息.

每个散点图中呈椭圆形的对象称为相关椭圆 (correlation ellipse), 它提供了一种变量之间是如何密切相关的可视化信息. 位于椭圆中心的点表示 x 轴变量的均值和 y 轴变量的均值所确定的点. 两个变量之间的相关性由椭圆的形状表示, 椭圆越被拉伸, 相关性越强. 椭圆越正, 越不相关.

散点图中绘制的曲线称为局部回归平滑 (loess smooth), 它表示 x 轴和 y 轴变量之间的一般关系. 例如: age 和 children 的曲线是倒 U, 峰值在 40 岁中年附近, 表明案例中年龄最大的人和最小的人比中年人拥有的孩子少. 因为这种趋势是非线性的, 这一发现不能单独从相关性中推断出来. 另一方面, 对于 age 和 bmi, 是一条倾斜的逐渐上升的线, 表明 bmi 随 age 增长而增加, 但我们已经从相关系数矩阵中推断出该结论.



6.2.3 第三步-基于数据训练模型

用 `lm()` 函数拟合线性回归模型.

无需指定回归模型的截距项, 默认假定存在.

多元回归模型语法
应用 stats 添加包中的函数 <code>lm()</code>
建立模型:
<pre>m <- lm(dv ~ iv, data = mydata)</pre> <ul style="list-style-type: none"> • <code>dv</code>: 是 <code>mydata</code> 数据框中需要建模的因变量 • <code>iv</code>: 为一个 R 公式, 用来指定 <code>mydata</code> 数据框中被用于模型的自变量 • <code>data</code>: 为包含变量 <code>dv</code> 和变量 <code>iv</code> 的数据框
该函数返回一个回归模型对象, 该对象能够用于预测。自变量之间的交互作用可以通过运算符 “ <code>*</code> ” 来给出。
进行预测:
<pre>p <- predict(m, test)</pre> <ul style="list-style-type: none"> • <code>m</code>: 由函数 <code>lm()</code> 训练的一个模型 • <code>test</code>: 一个包含测试数据的数据框, 该数据框和用来建立模型的训练数据有同样的特征。
该函数将返回一个含有预测值的向量。
例子:
<pre>ins_model <- lm(charges ~ age + children + sex + smoke, rdata = insurance) ins_pred <- predict(ins_model, insurance_test)</pre>

```
1 > ins_model <- lm(charges ~ age + children + bmi + sex + smoker + region,
2 +                               data = insurance)
```

上面的命令与下面相同:

```
1 ins_model <- lm(charges ~ ., data = insurance)
```

看看 β 系数:

```
1 > ins_model
2
3 Call:
4 lm(formula = charges ~ ., data = insurance)
5
6 Coefficients:
7 (Intercept)          age          sexmale          bmi
8      -11938.5       256.9       -131.3       339.2
9 children      smokeryes regionnorthwest regionsoutheast
10      475.5       23848.5      -353.0      -1035.0
11 regionsouthwest
12      -960.1
```

你可能会注意到, 虽然在我们模型公式中仅设定了 6 个变量, 但是除截距外输出了 8 个系数. 是因为 `lm()` 函数自动将一种称为**虚拟编码 (dummy coding)** 的技术应用于模型所包含的每一个因子类型的变量中.

虚拟编码允许名义特征通过为一个特征的每一类创建一个二元变量来将其处理为数值型变量, 即如果观测值属于某一类, 那就设定为 1, 否则为 0. 例如: `sex` 变量有两类: `male` 和 `female`, 这将分为两个二进制变量, R 中将其命名为 `sexmale` 和 `sexfemale`. 对于观测值 `sex=male`, 那么 `sexmale=1, sexfemale=0`; 对于观测值 `sex=female`, 那么 `sexmale=0, sexfemale=1`. 相同的编码适用于有三个类别甚至更多类别的变量如 `region`.

当添加一个虚拟编码的变量到回归模型中, 一个类别总是被排除在外作为参照类别. 估计的系数就是相对于参照类别设定的.

因此相对于女性, 男性每年医疗费少 \$131.10; 吸烟者平均多花 \$23848.5; 此外模型中三个地区系数为负, 意味着东北地区倾向于具有最高的平均医疗费.

默认 R 用 `factor` 变量中第一个水平作为参照组,, 当然你也可以用 `relevel()` 来手动指定参照组.

6.2.4 第四步-评估模型的性能

```
1 > summary(ins_model)
2
3 Call:
4 lm(formula = charges ~ ., data = insurance)
5
```

```

6 Residuals:
7   Min      1Q   Median      3Q      Max
8 -11304.9 -2848.1 - 982.1  1393.9  29992.8
9
10 Coefficients:
11             Estimate Std. Error t value Pr(>|t| )
12 (Intercept) -11938.5    987.8 -12.086 < 2e-16 ***
13 age          256.9     11.9   21.587 < 2e-16 ***
14 sexmale      -131.3    332.9 -0.394 0.693348
15 bmi          339.2     28.6   11.860 < 2e-16 ***
16 children     475.5     137.8   3.451 0.000577 ***
17 smokeryes   23848.5    413.1  57.723 < 2e-16 ***
18 regionnorthwest -353.0    476.3 -0.741 0.458769
19 regionsoutheast -1035.0    478.7 -2.162 0.030782 *
20 regionsouthwest -960.0    477.9 -2.009 0.044765 *
21 ---
22 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
23
24 Residual standard error: 6062 on 1329 degrees of freedom
25 Multiple R-squared:  0.7509,    Adjusted R-squared:  0.7494
26 F-statistic: 500.8 on 8 and 1329 DF,  p-value: < 2.2e-16

```

1. residuals(残差): 部分提供了预测误差的主要统计量; 另一方面, 误差的 50% 落在第一分位数和第三分位数之间, 故大部分预测值在超过真实值 2850 和低于真实值 1400 之间.
2. *** (星号): 表示模型中每个特征的预测能力. 星号越多, 与因变量的相关性越大.
3. Adjusted R-squared(多元 R 方值)(也称为判定系数): 提供了一种度量模型性能的方式, 即从整体上模型能多大程度解释因变量的值. 它类似于相关系数, 因为它的值越接近 1, 模型解释数据的性能就越好. 由于 R 方值为 0.749, 故近 75% 的因变量的变化程度可以由模型解释. 因为特征越多, 模型的变化程度越大, 所以调整 R 方值通过惩罚具有很多自变量的模型来修正 R 方值, 用它来比较具有不同数目的解释变量的模型的性能是很有用的.

6.2.5 第五步—提高模型性能

回归模型和其他机器学习方法的关键区别在于回归通常会让使用者来选择特征和设定模型.

6.2.5.1 模型的设定—添加非线性关系

此处的非线性是指 y 关于 x 的非线性, 但 y 关于 β 还是线性的

$$y = \alpha + \beta_1 x + \beta_2 x^2 \quad (6.9)$$

```

1 > insurance$age2<-insurance$age^2

```

6.2.5.2 转换—将一个数值型变量转换为一个二进制指标

假设我们有一种预感, 一个特征的影响不是累积的, 而是当特征的取值达到一个给定的阈值后产生影响. 例如, 对于在正常体重范围的个人来说,BMI 对医疗费用的影响可能为 0, 但是对于肥胖者 (i.e.bmi 不低于 30) 来说, 它可能与较高的费用密切相关.

我们可以通过建立一个二进制指标变量来建立这种关系, 如果 bmi 大于等于 30, 设为 1, 否则为 0. 该二元特征的 β 估计表示 bmi 大于等于 30 的个人相对于 bmi 小于 30 的个人对医疗费用的平均净影响.

要创建一个特征, 可以用 `ifelse()`, 该函数用于对向量中的每一个元素测试一个指定的条件, 并根据条件是 `true` 还是 `false` 返回 1 或 0:

```
1 > insurance$bmi30 <- ifelse(insurance$bmi >= 30, 1, 0)
```

如果你在决定是否要包含一个变量时遇到困难, 一种常见的做法就是包含它并检验显著性水平. 如果该变量在统计上并不显著, 那么就有证据支持在将来排除该变量.

6.2.5.3 模型的设定—加入相互作用的影响

到目前为止, 我们只考虑了每个特征对结果的单独影响 (贡献), 如果某些特征对因变量有综合影响, 那该怎么办呢?

当两个特征存在共同影响时, 这称为相互作用 (interaction). 如果怀疑两个变量相互作用, 那么可以通过在模型中添加它们的相互作用来检验这一假设. 为了体现肥胖指标 (bmi30) 和吸烟指标 (smoker) 的相互作用, 可以这样形式的表达: `charges~bmi30*smoker`.

运算符 `*` 是一个简写, 用来指示 R 对如下进行建模:

`charges~bim30+smokeryes+bmi30:smokeryes`

冒号表示两者的相互作用. 注意, 该表达式还自动包括 `bmi30` 和 `smoker` 变量及其相互作用.

6.2.5.4 全部放在一起—一个改进的回归模型

```
1 > ins_model2 <- lm(charges ~ age + age2 + children + bmi + sex +
2 +                               bmi30*smoker + region, data = insurance)
3 >
4 > summary(ins_model2)
5
6 Call:
7 lm(formula = charges ~ age + age2 + children + bmi + sex + bmi30 *
8 smoker + region, data = insurance)
9
10 Residuals:
11      Min       1Q   Median       3Q      Max
12 -17296.4 -1656.0 -1263.3 -722.1 24160.2
13
14 Coefficients:
```

```

15 |                               Estimate Std. Error t value Pr(>|t| )
16 | (Intercept)      134.2509  1362.7511  0.099 0.921539
17 | age            -32.6851   59.8242 -0.546 0.584915
18 | age2           3.7316   0.7463  5.000 6.50e-07 ***
19 | children        678.5612  105.8831  6.409 2.04e-10 ***
20 | bmi            120.0196  34.2660  3.503 0.000476 ***
21 | sexmale        -496.8245 244.3659 -2.033 0.042240 *
22 | bmi30          -1000.1403 422.8402 -2.365 0.018159 *
23 | smokeryes      13404.6866 439.9491 30.469 < 2e-16 ***
24 | regionnorthwest -279.2038 349.2746 -0.799 0.424212
25 | regionsoutheast -828.5467 351.6352 -2.356 0.018604 *
26 | regionsouthwest -1222.6437 350.5285 -3.488 0.000503 ***
27 | bmi30:smokeryes 19810.7533 604.6567 32.764 < 2e-16 ***
28 | ---
29 | Signif. codes:  0  '***'  0.001  '**'  0.01  '*'  0.05  '.'  0.1  ' '  1
30 |
31 | Residual standard error: 4445 on 1326 degrees of freedom
32 | Multiple R-squared:  0.8664,    Adjusted R-squared:  0.8653
33 | F-statistic: 781.7 on 11 and 1326 DF,  p-value: < 2.2e-16

```

6.3 理解回归树和模型树

在第五章我们学习了决策树. 通过对树的生长算法做小幅调整, 这些树可以同样应用于数值预测. 本节中, 我们将只考虑决策树用于数值预测, 不同于将其用于分类.

决策树用于数值预测可分为两类. 第一类为回归树 (**regression tree**), 回归树并没有使用线性回归的方法, 而是基于到达叶节点的案例的均值做出预测. 第二类称为模型树 (**model trees**), 模型树和回归树以大致相同的方式生长, 但是在每个叶节点, 根据到达该节点的案例建立多元线性回归模型.

6.3.0.1 将回归加入到决策树

相对于更常见的回归方法, 回归树和模型树的优点与缺点都列在下表:

优点:

- 将决策树的优点与对数值型数据建模的能力相结合
- 能自动选择特征, 允许该方法与大量特征一起使用
- 不需要使用者事先指定模型
- 拟合某些类型的数据可能会比线性回归好得多
- 不要求用统计的知识来解释模型

缺点:

- 不如线性回归常用
- 需要大量的训练数据
- 难以确定单个特征对于结果的总体净影响
- 可能比回归模型更难解释

虽然传统的回归方法通常是数值预测的首选,但是在有些情况下,数值决策树提供了明显的优势,如,决策树可能更适合于具有许多特征或者特征和结果之间具有许多复杂或非线性关系的任务,这些情形给回归带来了挑战. 回归建模关于数值型数据是如何分布的假设往往被现实世界的数据所违背,但决策树就不存在这种情况.

用于数值预测的决策树的建立方式与用于分类的决策树建立方式大致相同. 从根节点开始,按照特征使用分而治之的策略对数据进行划分,在进行一次分割之后,将会导致结果最大化地均匀增长.

在分类决策树中,一致性(均匀性)是由熵值来度量的;

对于数值决策树,一致性(均匀性)可以通过统计量(如方差,标准差,平均绝对偏差)来度量.

一个常见的分割标准称为标准偏差减少(Standard Deviation Reduction, SDR),它由下面地公式定义:

$$SDR = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i) \quad (6.10)$$

在这个公式中,函数 $sd(T)$ 指的是集合 T 中的值的标准差,而 T_1, T_2, \dots, T_n 是由对于一个特征的一次分割产生的值的集合; $|T|$ 项指的是集合 T 中观测的数量. 从本质上讲,公式度量的是从原始值的标准差减去分割后加权的标准差的减少量.

例如:一颗决策树需要决定对二元特征 A 进行分割还是对二元特征 B 进行分割.

```
1 > tee <- c(1, 1, 1, 2, 2, 3, 4, 5, 5, 6, 6, 7, 7, 7, 7)      # 原始数据
2 > at1 <- c(1, 1, 1, 2, 2, 3, 4, 5, 5)                      # 特征A的T_1划分
3 > at2 <- c(6, 6, 7, 7, 7, 7)                      # 特征A的T_2划分
4 > bt1 <- c(1, 1, 1, 2, 2, 3, 4)                      # 特征B的T_1划分
5 > bt2 <- c(5, 5, 6, 6, 7, 7, 7)                      # 特征B的T_2划分
```

让我们来比较 A 和 B 的 SDR:

```
1 > sdr_a <- sd(tee) - (length(at1) / length(tee) * sd(at1) + length(at2) /
2   length(tee) * sd(at2))
3 > sdr_b <- sd(tee) - (length(bt1) / length(tee) * sd(bt1) + length(bt2) /
4   length(tee) * sd(bt2))
5 > sdr_a
6 [1] 1.202815
7 > sdr_b
8 [1] 1.392751
```

由于对于特征 B,标准差减少得更多,所以决策树首先使用特征 B,它产生了比特征 A 略多的一致性(均匀性)集合.

假设决策数在这里停止生长, 只使用了这一分割, 那么回归树的工作就完成了. 它可以为新的案例进行预测, 取决于它们是落入 T_1 组还是 T_2 组. 如果案例最后在入 T_1 组, 那么模型将预测 $\text{mean}(\text{bt1})=2$, 否则将预测 $\text{mean}(\text{bt2})=6.25$.

相比之下, 模型树将多走一步. 使用落入组 bt1 的 7 个训练案例和落入组 bt2 的 8 个训练案例, 模型树可以建立一个结果相对于特征 A 的线性回归模型 (在回归模型中, 特征 B 没有任何帮助, 因为所有位于叶节点的个案与 B 有相同的值). 然后, 模型树可以使用两个线性模型中的任何一个为新案例做出预测.

6.4 例子—用回归树和模型树估计葡萄酒的质量

在这个案例学习中, 将使用回归树和模型树来创建一个能模仿葡萄酒专家评级的系统.

6.4.1 第一步—收集数据

白葡萄酒数据包含了 4898 个葡萄酒案例的 11 种化学特性的信息. 这些样本会由不少于 3 个鉴定师组成的小组以盲品的方式评级, 质量尺度从 0(很差) 到 10(极好). 如果鉴定者对于评级没有达成一致意见, 那么就会使用中间值.

Cortez 的研究评估了 3 种机器学习方法多元回归, 人工神经网络和支持向量机对葡萄酒数据建立模型的能力. 研究发现, 支持向量机提供了比线性回归模型显著更好的结果. 然而, 与回归不同的是, 支持向量机模型很难解释. 使用回归树和模型树, 我们或许能够改善回归的结果, 同时还能拥有一个容易理解的模型.

6.4.2 第二步—探索和准备数据

```
1 > wine<-read.csv("E:/机器学习/ml&r/Machine Learning with R/chapter 6/  
2 whitewines.csv")
```

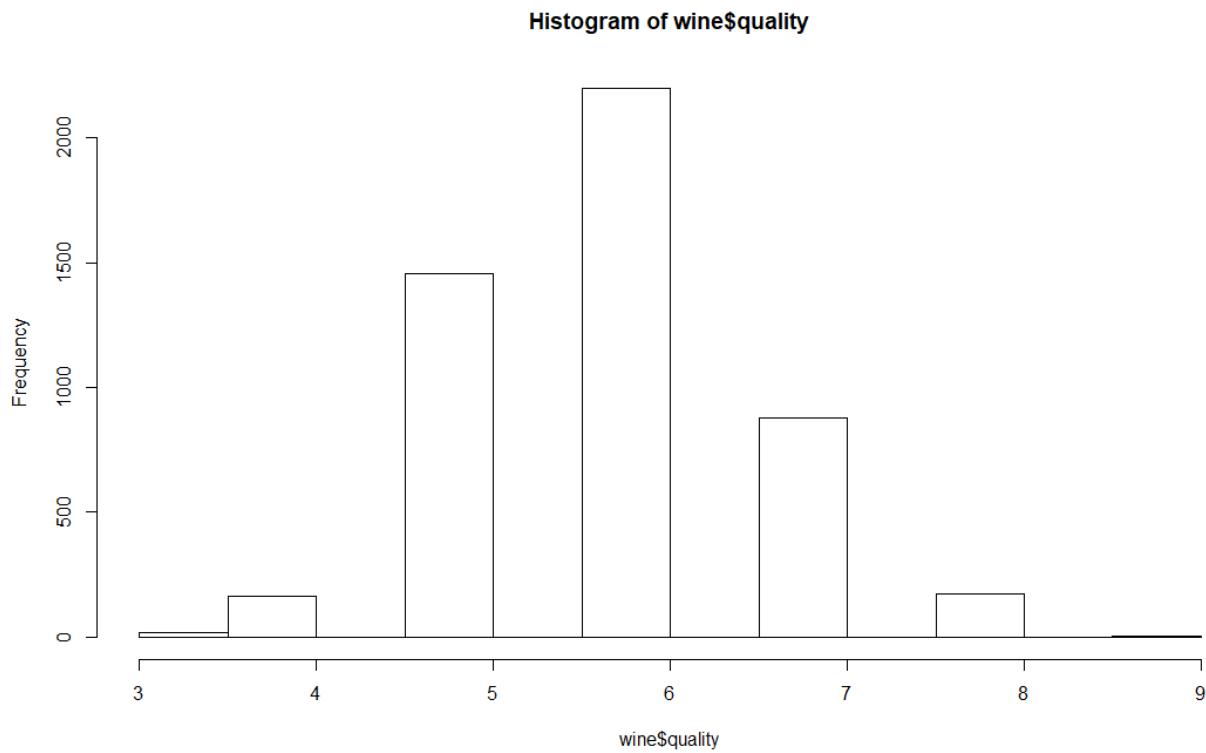
十一个特征和一个结果变量 quality:

```
1 > str(wine)  
2 'data.frame': 4898 obs. of 12 variables:  
3 $ fixed.acidity      : num  6.7 5.7 5.9 5.3 6.4 7 7.9 6.6 7 6.5 ...  
4 $ volatile.acidity    : num  0.62 0.22 0.19 0.47 0.29 0.14 0.12 0.38 0.16 0.37 ...  
5 $ citric.acid        : num  0.24 0.2 0.26 0.1 0.21 0.41 0.49 0.28 0.3 0.33 ...  
6 $ residual.sugar     : num  1.1 16 7.4 1.3 9.65 0.9 5.2 2.8 2.6 3.9 ...  
7 $ chlorides          : num  0.039 0.044 0.034 0.036 0.041 0.037 0.049 0.043 0.043 ...  
8 $ free.sulfur.dioxide: num  6 41 33 11 36 22 33 17 34 40 ...  
9 $ total.sulfur.dioxide: num  62 113 123 74 119 95 152 67 90 130 ...  
10 $ density            : num  0.993 0.999 0.995 0.991 0.993 ...  
11 $ pH                 : num  3.41 3.22 3.49 3.48 2.99 3.25 3.18 3.21 2.88 3.28 ...  
12 $ sulphates          : num  0.32 0.46 0.42 0.54 0.34 0.43 0.47 0.47 0.47 0.39 ...  
13 $ alcohol            : num  10.4 8.9 10.1 11.2 10.9 ...  
14 $ quality             : int  5 6 6 4 6 6 6 6 6 7 ...
```

相比于其他类型的机器学习模型, 决策树的优点之一就是它们可以处理多种类型的数据而无需进行预处理. 这意味着不需要将特征规范化或者标准化.

为了增加模型评价的信息, 还是需要花点精力来研究结果变量分布:

```
1 > hist(wine$quality)
```



似乎遵循一个相当于正太的钟形分布, 大约以数值 6 为中心. 直观上看大部分葡萄酒质量为平均质量, 少数葡萄酒特别差或者特别好.

但是研究 summary(wine) 的输出同样有益于发现异常值或者其他潜在数据问题.

```
1 > summary(wine)
2 fixed.acidity      volatile.acidity      citric.acid      residual.sugar
3 Min.    : 3.800      Min.    :0.0800      Min.    :0.0000      Min.    : 0.600
4 1st Qu.: 6.300      1st Qu.:0.2100      1st Qu.:0.2700      1st Qu.: 1.700
5 Median  : 6.800      Median  :0.2600      Median  :0.3200      Median  : 5.200
6 Mean    : 6.855      Mean    :0.2782      Mean    :0.3342      Mean    : 6.391
7 3rd Qu.: 7.300      3rd Qu.:0.3200      3rd Qu.:0.3900      3rd Qu.: 9.900
8 Max.    :14.200      Max.    :1.1000      Max.    :1.6600      Max.    :65.800
9 chlorides      free.sulfur.dioxide      total.sulfur.dioxide      density
10 Min.    :0.00900      Min.    : 2.00      Min.    : 9.0      Min.    :0.9871
11 1st Qu.:0.03600      1st Qu.:23.00      1st Qu.:108.0      1st Qu.:0.9917
12 Median  :0.04300      Median  :34.00      Median  :134.0      Median  :0.9937
13 Mean    :0.04577      Mean    :35.31      Mean    :138.4      Mean    :0.9940
14 3rd Qu.:0.05000      3rd Qu.:46.00      3rd Qu.:167.0      3rd Qu.:0.9961
15 Max.    :0.34600      Max.    :289.00      Max.    :440.0      Max.    :1.0390
```

```

16 pH          sulphates      alcohol      quality
17 Min. :2.720  Min. :0.2200  Min. : 8.00  Min. :3.000
18 1st Qu.:3.090 1st Qu.:0.4100 1st Qu.: 9.50 1st Qu.:5.000
19 Median :3.180 Median :0.4700 Median :10.40 Median :6.000
20 Mean   :3.188 Mean   :0.4898 Mean   :10.51 Mean   :5.878
21 3rd Qu.:3.280 3rd Qu.:0.5500 3rd Qu.:11.40 3rd Qu.:6.000
22 Max.   :3.820  Max.   :1.0800  Max.   :14.20  Max.   :9.000

```

将数据分为训练数据集和测试数据集, 源数据已随机3/4 用于训练.

```

1 > wine_train <- wine[1:3750, ]
2 > wine_test <- wine[3751:4898, ]

```

我们将数据分为训练数据集和测试数据集, 由于 wine(葡萄酒) 数据已经处理成随机的顺序, 所以可以将数据分割成两个连续行的集合.

6.4.3 第三步-基于数据训练模型

虽然几乎决策树的所有实现都可以用来进行回归树建模, 但是 rpart(递归划分) 包提供了像 CArT(分类回归树) 团队中所描述的最可靠的回归树实现. 作为 CART 的经典 R 实现, rpart 添加包同样有着充分的帮助文档, 有用于可视化和评估 rpart 模型的多个函数支持.

回归树语法

应用 rpart 添加包中的函数 rpart()

建立模型:

```
m <- rpart(dv ~ iv, data = mydata )
```

- dv: 是 mydata 数据框中需要建模的因变量
- iv: 为一个 R 公式, 用来指定 mydata 数据框中被用于模型的自变量
- data: 为包含变量 dv 和变量 iv 的数据框

该函数返回一个回归树模型对象, 该对象能够用于预测。

进行预测:

```
p <- predict(m, test, type = "vector")
```

- m: 由函数 rpart() 训练的一个模型
- test: 一个包含测试数据的数据框, 该数据框和用来建立模型的训练数据有同样的特征。
- type: 给定返回的预测值的类型, 取值为 "vector" (预测数值型数据), 或者 "class" (预测类别), 或者 "prob" (预测类别的概率)。

该函数的返回值取决于 type 参数, 它是一个含有预测值的向量。

例子:

```
wine_model <- rpart(quality ~ alcohol + sulfates, data = wine_train)

wine_predictions <- predict(wine_model), wine_test)
```

```

1 > library(rpart)
2 > m.rpart <- rpart(quality ~ ., data = wine_train)
3 > m.rpart
4 n= 3750
5

```

```

6  node), split, n, deviance, yval
7  * denotes terminal node
8
9 1) root 3750 2945.53200 5.870933
10 2) alcohol< 10.85 2372 1418.86100 5.604975
11 4) volatile.acidity>=0.2275 1611 821.30730 5.432030
12 8) volatile.acidity>=0.3025 688 278.97670 5.255814 *
13 9) volatile.acidity< 0.3025 923 505.04230 5.563380 *
14 5) volatile.acidity< 0.2275 761 447.36400 5.971091 *
15 3) alcohol>=10.85 1378 1070.08200 6.328737
16 6) free.sulfur.dioxide< 10.5 84 95.55952 5.369048 *
17 7) free.sulfur.dioxide>=10.5 1294 892.13600 6.391036
18 14) alcohol< 11.76667 629 430.11130 6.173291
19 28) volatile.acidity>=0.465 11 10.72727 4.545455 *
20 29) volatile.acidity< 0.465 618 389.71680 6.202265 *
21 15) alcohol>=11.76667 665 403.99400 6.596992 *

```

对于决策树中的每个节点, 到达决策点的案例数量都列出来了. 例如, 所有的 3750 个案例从根节点开始, 其中有 2372 个案例的 $\text{alcohol} < 10.85$, 1378 个案例的 $\text{alcohol} \geq 10.85$. 因为 alcohol 是决策树中第一个使用的变量, 所以它是葡萄酒质量中唯一最重要的指标.

用 * 表示的节点是终端或者叶节点, 这意味着它们会产生预测 $yval$. 如, 节点 5 有一个 5.971091 的 $yval$. 当该决策树用来预测时, 对任意一个葡萄酒案例, 如果其 $\text{alcohol} < 10.85$ 且 $\text{volatile.acidity} < 0.2275$, 那么它的质量值将预测为 5.97.

6.4.3.1 可视化决策树

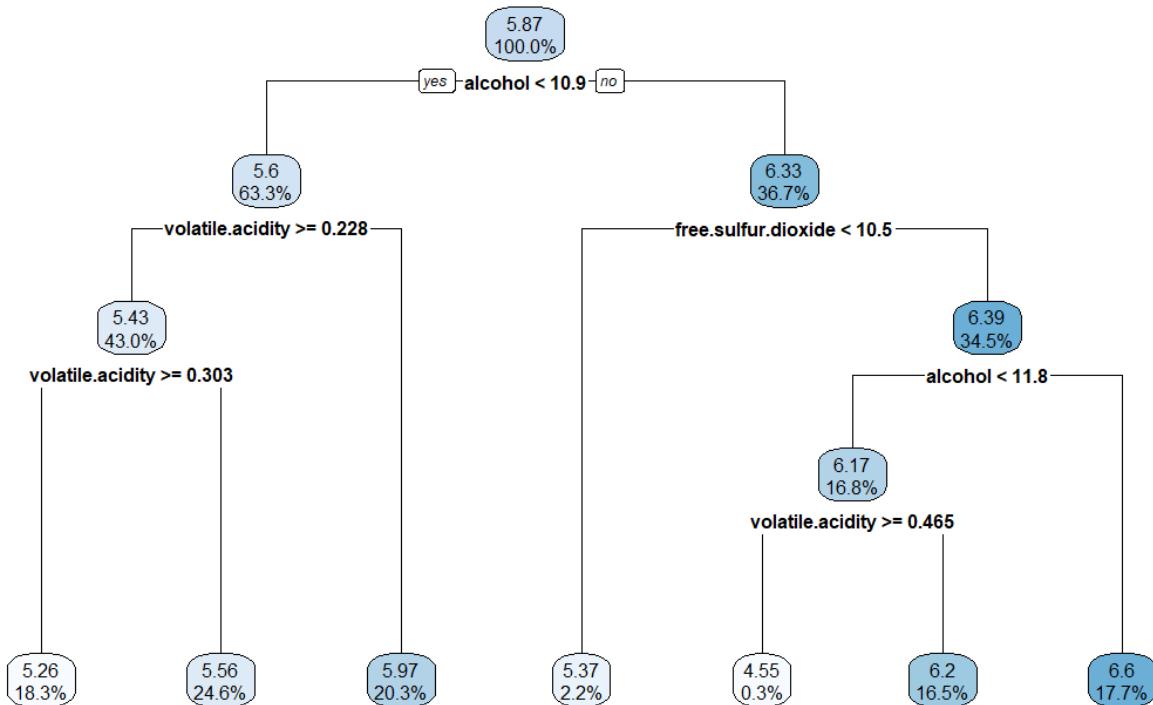
用 `rpart.plot` 包提供了一个易于使用的函数来生成具有出版质量的决策树:

```

1 > library(rpart.plot)
2 > rpart.plot(m.rpart, digits = 3)

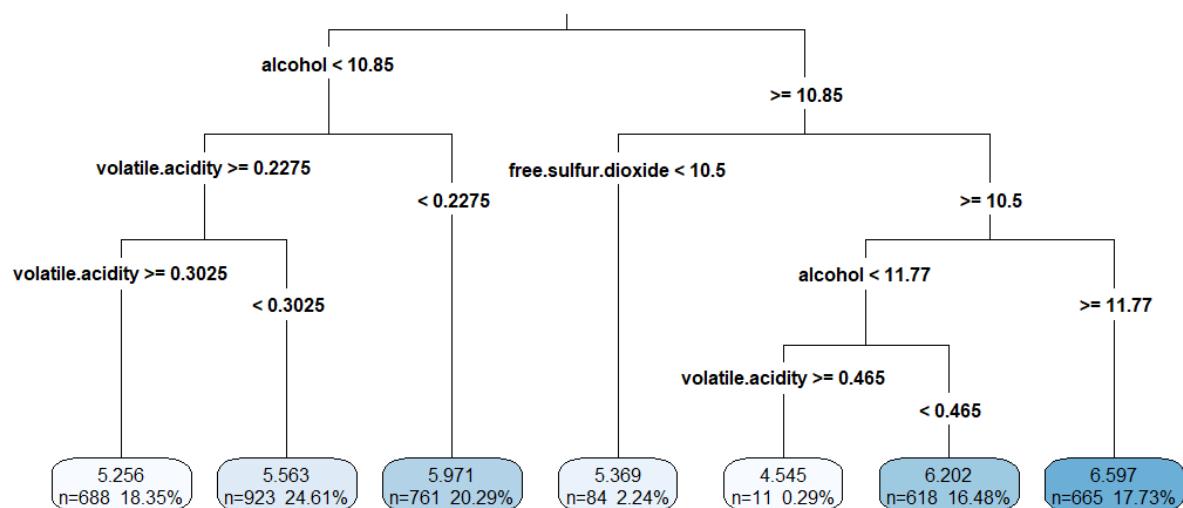
```

`digits` 控制数字位数.



增加参数 `fallen.leaves` 强制叶节点与图的底部保持一致, 参数 `type` 和 `extra` 影响决策和节点被标记的方式.

```
1 > rpart.plot(m.rpart, digits = 4, fallen.leaves = TRUE, type = 3, extra = 101)
```



6.4.4 第四步—评估模型的性能

为了使用基于测试数据的回归树模型进行预测, 用 `predict()` 函数. 在默认情况下, 该函数返回结果变量的数值估计, 我们将它的返回值保存在一个名为 `p.rpart` 的向量中.

```
1 > p.rpart <- predict(m.rpart, wine_test)
2
3 > summary(p.rpart)
4 Min. 1st Qu. Median Mean 3rd Qu. Max.
5 4.545 5.563 5.971 5.893 6.202 6.597
6 > summary(wine_test$quality)
7 Min. 1st Qu. Median Mean 3rd Qu. Max.
8 3.000 5.000 6.000 5.901 6.000 9.000
```

我们预测的主要统计量表明了一个潜在的问题, 预测值于真实值相比落在一个更窄的范围内. 这一发现表明, 该模型不能正确识别极端的情形, 尤其是最好的和最差的.

另一方面, 在第一四分位数和第三四分位数之间, 我们可能做的不错.

预测的质量 (quality) 值和真实的质量 (quality) 之间的相关性:

```
1 > cor(p.rpart, wine_test$quality)
2 [1] 0.5369525
```

相关系数 0.54 肯定是可以接受的. 然而相关系数只是度量了预测值与真实值的相关性有多强, 而不是度量预测值离真实值有多远的方法.

6.4.4.1 用平均绝对误差度量性能

一般来说, 另一种思考模型性能的方法就是考虑它的预测值离真实值有多远, 这种度量方法称为平均绝对误差 (mean absolute error,MAE)

MAE: n 表示预测值的数量, e_i 表示第 i 个预测值的误差

$$MAE = \frac{1}{n} \sum_{i=1}^n |e_i| \quad (6.11)$$

本质上, 这个方程得到的误差绝对值的均值. 由于误差仅仅是预测值与真实值之间的差值, 所以可以创建一个简单的 `MAE()` 函数:

```
1 > MAE <- function(actual, predicted) {
2     +     mean(abs(actual - predicted))
3     + }
4 >
5 > MAE(p.rpart, wine_test$quality)
6 [1] 0.5872652
```

就平均而言, 这意味着模型的预测值与真实的质量分数之间的差值大约为 0.59. 基于质量尺度是从 0~10, 这似乎表明我们的模型做得相当好.

另一方面, 本身我们的酒大多数不好也不差, 约为 5-6, 因此一个什么也没做仅仅预测了均值的分类器可能同样会做的相当好.

```
1 > mean(wine_train$quality) # result = 5.87
2 [1] 5.870933
3 > mean_abserror<-MAE(5.87, wine_test$quality)
4 > mean_abserror
5 [1] 0.6722474
```

如果我们对每一个葡萄酒案例预测的值为 5.87, 那么我们将只有大约 0.67 的平均绝对误差.

作为比较,Cortez 报告了神经网络模型的 MAE 为 0.58, 支持向量机的 MAE 为 0.45.

6.4.5 第五步-提高模型的性能

我们尝试构建模型树. 模型树可以通过回归模型取代叶节点来改善回归树. 这通常会导致比回归树更精确的结果, 而回归树在叶结点进行预测只使用了单一的值.

目前模型树最先进的算法为 RWeka 包中的 M5' 算法, 函数 M5P().

模型树语法

应用 RWeka 添加包中的函数 M5P()

建立模型:

```
m <- M5P(dv ~ iv, data = mydata )
```

- dv: 是 mydata 数据框中需要建模的因变量
- iv: 为一个 R 公式, 用来指定 mydata 数据框中被用于模型的自变量
- data: 为包含变量 dv 和变量 iv 的数据框

该函数返回一个模型树对象, 该对象能够用于预测。

进行预测:

```
p <- predict(m, test)
```

- m: 由函数 M5P() 训练的一个模型
- test: 一个包含测试数据的数据框, 该数据框和用来建立模型的训练数据有同样的特征

该函数返回一个含有预测值的数值向量。

例子:

```
wine_model <- M5P(quality ~ alcohol + sulfates, data = wine_train)
wine_predictions <- predict(wine_model, wine_test)
```

```
1 > library(RWeka)
2 > m.m5p <- M5P(quality ~ ., data = wine_train)
3
4 > m.m5p
```

输出的树很大. 与前面的回归树很相似, 但是一个重要的区别是, 节点不是以一个数值预测终止的, 而是以一个线性模型终止的, 这里表示为 LM1 和 LM2.

```
1 LM num: 163
2 quality =
3 1.2221 * volatile.acidity
4 - 0.1667 * citric.acid
```

```
5 + 0.0175 * chlorides
6 - 2.0814 * free.sulfur.dioxide
7 - 0.0022 * total.sulfur.dioxide
8 + 0.0001 * density
9 - 35.1322 * pH
10 + 0.2 * sulphates
11 + 0.1468 * alcohol
12 + 40.6639
13
14 Number of Rules : 163
```

这个结果是错误的

线性模型本身娴熟在输出的后面, 如 LM163 如上所示, 这些值完全可以向我们在本章前面建立的多元回归模型一样解释, 每一个数字都是相关的特征对于预测的葡萄酒质量的净影响 (效应).

.....

Chapter 7

黑箱方法—神经网络和支持向量机

黑箱 (black box) 过程, 就是输出转换为输入的机制通过一个模糊化处理. 在机器学习下, 黑箱是因为潜在的模型基于复杂的数学系统, 而且结果难以解释.

你将会学到:

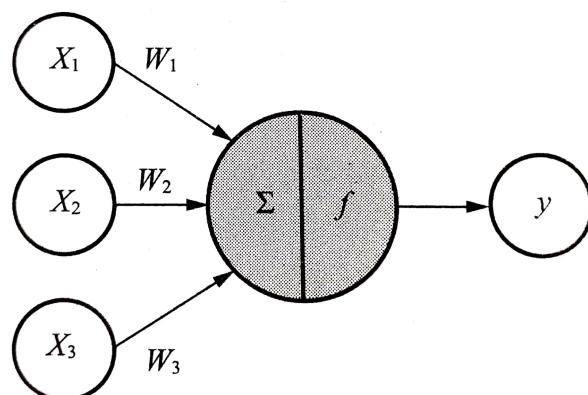
- 为了模拟任意函数 (功能), 神经网络借用了人们理解人脑所应用的一些概念.
- 支持向量机使用多维曲面来定义特征和结果的关系;
- 模型很容易地用到现实世界的问题中.

7.1 理解神经网络

人工神经网络 (artificial neural network, ANN) 对一组输入信号和一组输出信号之间的关系进行建模, 使用的模型来源于人类大脑对来自感觉输入的刺激是如何反应的理解. 就像大脑使用一个称为神经元 (neuron) 的相互连接的细胞网络来创建一个巨大的并行处理器一样, 人工神经网络使用仍神经元或者节点 (node) 的网络来解决学习问题.

广义上讲, 人工神经网络是可以应用于几乎所有学习任务的多功能学习方法: 分类, 数值预测, 甚至无监督的模式识别.

7.1.1 从生物神经元到人工神经元



一个单一的人工神经元模型, 可以用非常类似于生物模型的术语来解释. 如图, 一个有向图定义了树突接受的输入信号 (变量 x) 和输出信号 (变量 y) 之间的关系. 与生物神经元一样, 每一个树突的信号都根据其重要性被加权 (w 值), 输入信号由细胞体求和, 然后该信号根据一个用 f 表示的激活函数 (**activation function**) 来传递.

一个典型的有 n 个输入树突的神经元可以用下面公式表示. 权重 w 可以控制 n 个输入 x 中每个输入对输入信号之和所做的贡献大小. 激活函数 $f(x)$ 使用净总和, 结果信号 $y(x)$ 就是输出轴突.

$$y(x) = f\left(\sum_{i=1}^n w_i x_i\right) \quad (7.1)$$

就像使用积木那样, 神经网络应用这种方式定义的神经元来构建复杂的数据模型. 虽然有很多种不同的神经网络, 但是每一种都可以由下面的特征来定义.

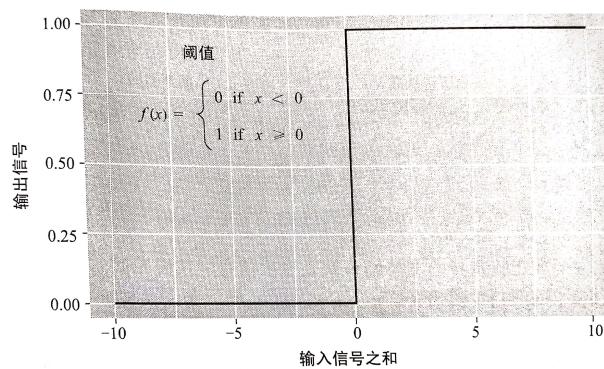
- **激活函数 (activation function):** 将神经元的净输入信号转换成单一的输出信号, 以便进一步在网络中传播.
- **网络拓扑 (network topology)(或结构):** 描述了模型中神经元的数量以及层数和它们连接的方式.
- **训练算法 (training algorithm):** 指定如何设置连接权重, 以便抑制或者增加神经元在输入信号中的比重.

7.1.2 激活函数

激活函数是人工神经元处理信息并将信息传递到整个网络的机制. 正如人工神经元是以生物中的神经元为模型, 激活函数也是以生物神经元的机制为模型.

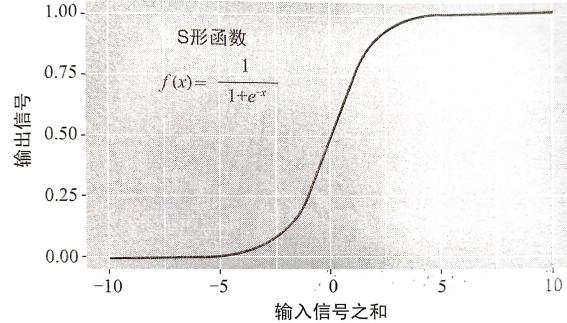
在生物世界中, 激活函数可以想象为一个过程, 涉及对总的输入信号求和, 和确定其是否满足激活阈值. 如果满足, 神经元传递信号; 否则, 它不执行任何操作. 在人工神经网络术语中, 这称为**阈值激活函数 (threshold activation function)**, 因为它近在以恶指定的输入阈值达到之后, 才产生一个输出信号.

下图显示了一个典型的阈值函数. 在这种情况下, 当输入信号的总和至少为 0 时, 神经元才击破阈值. 因为它的形状, 有时它也称为**单位跳跃激活函数 (unit step activation function)**.

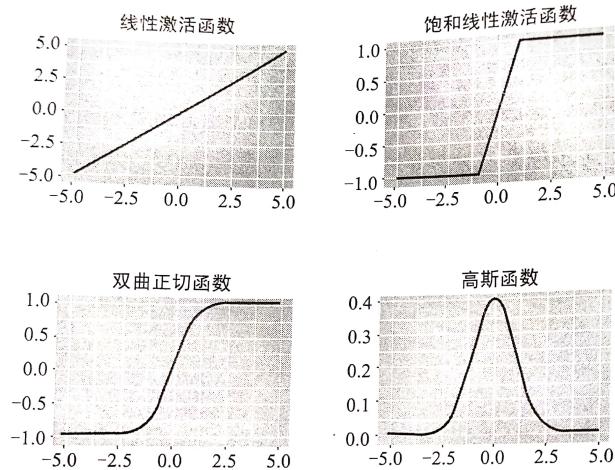


虽然该函数其与生物学的相似之处很有趣, 但是它很少用于人工神经网络中. 从生物化学的局限中解脱出来, 根据它们解释令人满意的数学特征的能力和对数据之间的关系建立模型的能力来选择人工神经网络激活函数.

或许最常用的方法是 **S 形激活函数 (sigmoid activation function)**(特别是逻辑 S 形, the logistic sigmoid), 其中 e 是自然对数的底. 尽管它与阈值激活函数有类似的步骤或者 S 的形状, 但是输出信号不再是二元的, 输出值可以是落在 0~1 的任何地方. 此外, 该 S 形激活函数是可微的, 这意味着它很有可能计算出遍及整个输入范围的导数. 正如你后面要学习的, 该特征对于创建高效的人工神经网络优化算法是至关重要的.



虽然该 S 形激活函数也许是最常用的激活函数, 并且通常在默认情况下使用, 但是有些神经网络允许选择其他的激活函数. 如图:



构成这些激活函数之间的差异的主要细节就是输出信号的范围不同. 激活函数的选择与具体的神经网络有关, 它可能更适合拟合某些类型的数据, 允许构建专门的神经网络. 如, 线性激活函数产生非常类似于线性回归模型的神经网络, 而高斯激活函数产生称为径向基函数 (**Radial Basis Function, RBF**) 网络模型.

重要的是要认识到, 对于许多激活函数, 影响输出信号的输入值范围是相对较窄的. 例如, 在 S 形激活函数的情况下, 对于一个低于 -5 或者超过 +5 的输入信号, 输出信号始终为 0 或 1. 这种方式的信号压缩会导致一个饱和信号位于非常动态化的输入的最高端或者最低端. 因为本质上这是将输入值压缩到一个较小的输出范围, 所以这样的激活函数 (像 S 形) 有时候称为压缩函数 (**squashing function**).

压缩问题的解决办法是转换所有的神经网络输入, 使特征值落在 0 附近的小范围内. 通常情况下, 这可以通过标准化或者规范化完成. 通过限制输入值, 激活函数将对整个范围采取行动, 从而预防取值尺度很大的特征 (例如收入) 支配取值尺度较小的特征 (例如, 家庭的孩子个数). 另一个好处是, 这些模型也可能更快地训练, 因为这些算法可以通过输入值的可操作范围更快地迭代.

虽然理论上一个神经网络可以经过多次迭代来调整它的权重以适应非常动态化的特征,但是在极端情况下,许多算法将在此发生之前就已经停止迭代. 如果你的模型预测是没有意义的,请检查你是否已经正确标准化了输入数据.

7.1.3 网络拓扑

神经网络的学习能力来源于它的**拓扑结构 (topology)**, 或者相互连接的神经元的模式与结构. 虽然有无数的网络结构形式,但是它们可以通过三个关键特征来区分:

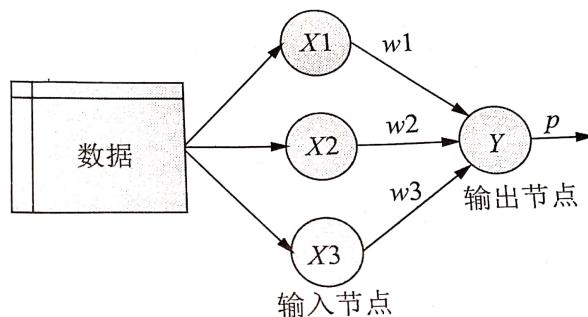
- 层的数目;
- 网络中信息是否允许向后传播;
- 网络中每一层内的节点数.

拓扑结构决定了可以通过网络学习进行学习任务的复杂性. 一般来说,更大更复杂的网络能够识别更微妙的模式及更复杂的决策边界. 然而, 神经网络的效能不仅是一个网络规模的函数,也取决于其构成元素的组织方式.

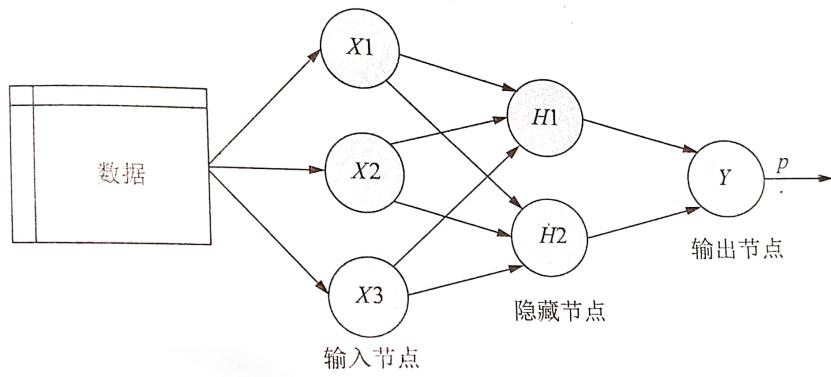
7.1.3.1 层的数目

为了定义拓扑,需要一个术语来区分位于网络不同位置的人工神经元. 下图显示了一个非常简单网络的拓扑结构. 称为**输入节点 (input node)**的一组神经元直接从输入的数据接收未经处理的信号,然后每个输入节点负责处理数据集中一个单一的特征,该特征的值将由节点的激活函数进行转换. 从输入节点产生的信号由**输出节点 (output node)**接收,输出节点使用他自己的激活函数来生成最终的预测(这里记为 P).

输入节点和输出节点安排在称为**层 (layer)**的组中. 因为输入节点处理正确接收的输入数据,所以该网络只有一组连接权重(这里记为 w_1, w_2, w_3). 因此它称为**单层网络 (single-layer network)**. 单层网络可以用于基本的模式分类,特别是可用于能够线性分割的模式,但大多数的学习任务需要更复杂的网络.



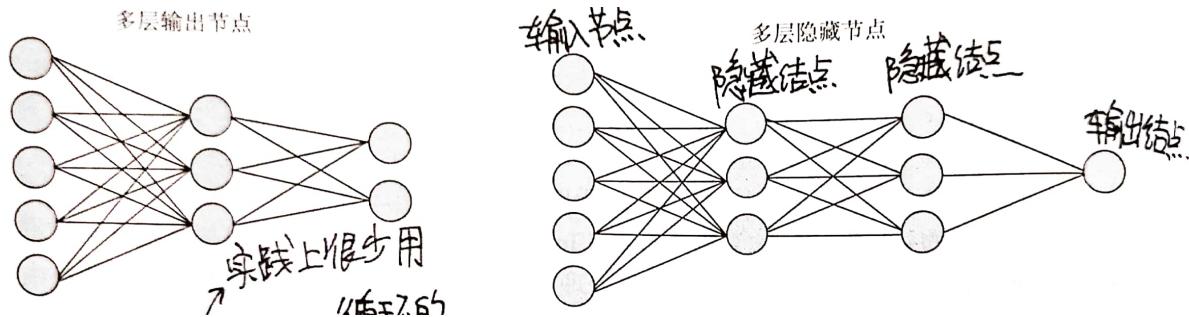
一种明显的创建更复杂网络的方法是通过添加额外的层. 就像这里描绘的, **多层网络 (multilayer network)**添加了一个或者更多的**隐藏层 (hidden layer)**,它们在信号到达输出节点之前处理来自输入节点的信号. 大多数错层网络被**完全连接 (fully connected)**,这意味着前一层中的每个节点都连接到下一层中的每个节点,但这不是必须的.



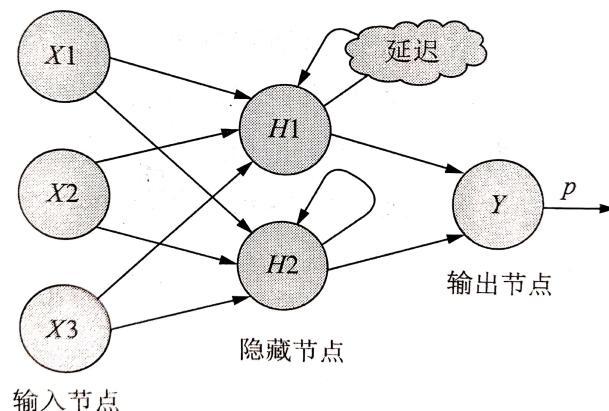
7.1.3.2 信息传播的方向

在前面的例子中, 箭头用来指示信号只在一个方向上传播. 如果网络中的输入信号在一个方向上从一个节点到另一个节点连续地传送, 直到到达输出层, 那么这样的网络称为前馈网络 (feedforward network).

虽然信息流有限制, 但是前馈网络提供了大量令人吃惊地灵活性. 例如, 层数和每一组的节点数都可以改变, 多个结果可以同时进行建模, 或者可以应用多个隐藏层 (这种做法有时成为深度学习 (deep learning)).



相比之下, 递归网络 (recurrent network) (或者反馈网络, feedback network) 允许信号使用循环在两个方向上传播. 这个性质更贴近地模拟了生物神经网络的工作原理, 它使得极其复杂的模式可以被学习. 增加一个短期记忆 (下图中标记为延迟 (delay)) 会给递归网络增加巨大的功效. 值得注意的是, 这包含了能够理解经过一段时间的事件序列的能力. 因此, 递归网络可用于股市预测 (stock market prediction), 语言理解 (speech comprehension) 和天气预报 (weather forecasting)



尽管递归网络很有潜力, 但是它们在很大程度上仍然是理论上的, 在实践中很少使用. 另一方面,

前馈网络已经广泛地应用于现实世界地问题中. 实际上, **多层前馈网络** (又称**多层感知器 (multilayer perception, MLP)**) 是人工神经网络拓扑结构的事实标准. 如果有人正在拟合一个神经网络而无需额外说明, 那么他们最有可能指的是多层前馈网络.

7.1.3.3 每一层的节点数

除了层数和信息传播方向的变化外, 神经网络同样可以改变每一层的节点数, 从而导致复杂性发生改变. 输入节点的个数由输入数据特征的数量预先确定. 类似地, 输出节点的个数由需要进行建模的结果或者结果中的分类水平预先确定. 然而, 隐藏节点的个数留给使用者在训练模型之前确定.

不幸的是, 没有可信的规则来确定隐藏层中神经元的个数. 合适的数目取决于输入节点的个数, 训练数据的数量, 噪声数据的数量, 以及许多其他因素之间的学习任务的复杂性.

一般情况下, 更复杂的网络拓扑结构具有更多数目的网络连接, 允许更复杂问题的学习. 较多数量的神经元将产生反映训练数据更严格的模型, 但有过度拟合的风险, 而且它可能不能充分地推广到未来的数据. 此外, 大型神经网络计算量也很大, 而且训练缓慢.

最好的做法就是基于验证数据集, 使用较少的节点产生适用 (足够) 的性能. 在大多数情况下, 即使只有少量的隐藏节点 (往往少到屈指可数), 但是神经网络却可以提供惊人巨大的学习能力.

已经证明, 具有至少一个充分多神经元隐藏层的神经网络是一种**通用函数逼近器 (universal function approximator)**. 从本质上讲, 这意味着这样的一个网络可以用来以任意精度逼近有界区间上的任意连续函数.

7.1.4 用后向传播训练神经网络

网络拓扑结构是一块空白石板, 通过它本身并没有学到任何东西. 就像一个刚出生的孩子, 它必须用经验进行训练. 当神经网络处理输入数据时, 神经元之间的连接被加强或者减弱, 类似于一个婴儿在体验外界环境时, 他大脑的发育过程. 网络的连接权重反映了观察到的随时间变化的模式.

通过调整连接权重训练神经网络模型的计算量非常大. 因此, 尽管人工神经网络之前已经被研究了几十年, 但是很少将它们应用到真实世界的学习任务中, 直到 20 世纪 80 年代中后期, 一种有效的训练人工神经网络的方法被发现. 该算法使用了一种后向传播误差的策略, 简称为**后向传播 (backpropagation)**.

虽然相对于许多其他的机器学习算法, 后向传播算法还是出了名地慢, 但是该方法使得人们对人工神经网络地兴趣再度升起. 所以, 现在使用后向传播算法地多层前馈网络在数据挖掘领域是常见的. 这类模型具有如下的优点和缺点:

优点:

- 适用于分类和数值预测问题
- 属于最精确的建模方法
- 对数据的基本关系几乎不需要作出假设

缺点:

- 计算量大, 训练缓慢, 特别是在网络拓扑结构复杂的情况下
- 很容易过度拟合或者不充分拟合训练数据
- 如果不是不可能, 复杂黑箱模型的结果很难解释.

在其最一般的形式中, 后向传播算法通过两个过程的多次循环进行迭代. 该算法的每一次迭代称为一个新纪元 (**epoch**). 因为网络不包含先验的 (a priori)(已有的) 知识, 所以通常在开始之前随机设定权重. 然后, 算法通过过程循环, 直到达到一个停止准则. 该循环包括:

- 在前向阶段 (**forward phase**) 中, 神经元在从输入层到输出层的序列被激活, 沿途应用每一个神经元的权重和激活函数, 一但到达最后一层, 就产生一个输出信号.
- 在后向阶段 (**backward phase**) 中, 由前向阶段产生的网络输出信号与训练数据中的真实目标值进行比较, 网络的输入信号与真实目标值之间产生的误差在网络中向后传播, 从而来修正神经元之间的连接权重, 并减少将来产生的误差.

随着时间的推移, 网络使用向后发送的信息来减少网络的总误差. 然而, 还有一个问题: 一位内每个神经元的输入和输出之间的关系很复杂, 所以该算法如何确定一个权重需要改变多少 (或者是否需要改变呢)?

回答这个问题涉及一个称为梯度下降法 (**gradient descent**) 的即使. 从概念上讲, 它的运作方式类似于一个被困在丛林的探险者如何找到一条通向水源的路线. 通过研究地形, 在具有最大向下斜坡的方向不断走, 很有可能最终到达低谷, 而这很可能是一条河床.

在一个类似的过程中, 后向传播算法利用每一个神经元的激活函数的导数来确定每一个输入权重方向上的梯度-因此, 有一个可微的激活函数很重要, 梯度将会因为权重的改变而表明误差是如何急剧减少或增加的. 该算法将试图通过一个称为学习率 (**learning rate**) 的量来改变权重以使得误差最大化地减小. 学习率越大, 算法试图下降的梯度就越快, 这可以减少训练冒着风险越过山谷的时间.

7.2 用人工神经网络对混凝土的强度进行建模

在工程领域, 对建筑材料的性能有精确的估计至关重要.

7.2.1 第一步-收集数据

混凝土数据集包含了 1030 个混凝土案例, 8 个描述混合物成分的特征.

7.2.2 第二步-探索和准备数据

```
1 > concrete<-read.csv("E:/机器学习/ml&r/Machine Learning with R/chapter 7/  
2 concrete.csv")  
3  
4 > str(concrete)  
5 'data.frame': 1030 obs. of 9 variables:
```

```

6 $ cement      : num  141 169 250 266 155 ...
7 $ slag        : num  212 42.2 0 114 183.4 ...
8 $ ash         : num  0 124.3 95.7 0 0 ...
9 $ water       : num  204 158 187 228 193 ...
10 $ superplastic: num  0 10.8 5.5 0 9.1 0 0 6.4 0 9 ...
11 $ coarseagg   : num  972 1081 957 932 1047 ...
12 $ fineagg     : num  748 796 861 670 697 ...
13 $ age          : int  28 14 28 28 28 90 7 56 28 28 ...
14 $ strength     : num  29.9 23.5 29.2 45.9 18.3 ...

```

数据框中的 9 个变量对应于数据集中的 8 个特征和 1 个结果, 虽然已经产生一个很明显的问题. 神经网络的运行最好是输入数据缩放到 0 附近的狭窄范围内.

通常, 解决这个问题的方法是用规范化或者标准化函数来重新调整数据. 如果数据服从一个钟形曲线 (如第二章描述的正态分布), 那么使用 r 内置的 scale() 函数才可能是有意义的. 另一方面, 如果数据服从均匀分布或者严重非正态分布, 那么将其标准化到一个 0~1 范围可能更合适. 在这种情况下, 我们将使用后者.

在第三章中, 我们自定义的 normalize() 函数为:

```

1 > normalize <- function(x) {
2 +   return((x - min(x)) / (max(x) - min(x)))
3 + }

```

用 lapply() 使得以上函数作用域数据框的每一列:

```

1 > concrete_norm <- as.data.frame(lapply(concrete, normalize))

```

数据对比:

```

1 > summary(concrete_norm$strength)
2 Min. 1st Qu. Median Mean 3rd Qu. Max.
3 0.0000 0.2664 0.4001 0.4172 0.5457 1.0000
4
5 > summary(concrete$strength)
6 Min. 1st Qu. Median Mean 3rd Qu. Max.
7 2.33 23.71 34.45 35.82 46.13 82.60

```

训练模型之前应用于数据的任何变换, 之后需要应用反变换, 以便将数据转换回原始得测量单位. 为了便于重新调整, 保存原始数据或者至少保存原始数据得主要统计量是明智的.

原 csv 文件已经随机排列, 只要按照 I-Cheng Yeh 原始发表的范例, 将数据划分 3:1 的训练数据集和测试数据集:

```

1 > concrete_train <- concrete_norm[1:773, ]
2 > concrete_test <- concrete_norm[774:1030, ]

```

我们将用训练集创建神经网络, 用测试集评估模型推广到未来的结果多好, 因为很容易过度拟合, 所以这个步骤很重要.

7.2.3 第三步-基于数据训练模型

为了对混凝土中使用的原料和最终产品的强度之间的关系建立模型, 将使用多层前馈神经网络. 用 neuralnet 包, 它提供了一个标准的易于使用的网络实现, 还提供了一个函数用来绘制网络拓扑结构.

神经网络语法

应用 neuralnet 添加包中的 neuralnet() 函数

建立模型:

```
m <- neuralnet(target ~ predictors, data = mydata, hidden = 1)
```

- target: 是数据框 mydata 中需要建模的输出变量
- predictors: 是给出数据框 mydata 中用于预测的特征的一个 R 公式
- data: 给出包含变量 target 和 predictors 的数据框
- hidden: 给出隐藏层中神经元的数目 (默认为 1)

进行预测:

```
p <- compute(m, test)
```

- m: 函数 neuralnet() 所训练的模型
- test: 包含测试数据的数据框, 它具有和用于训练模型的训练数据相同的特征

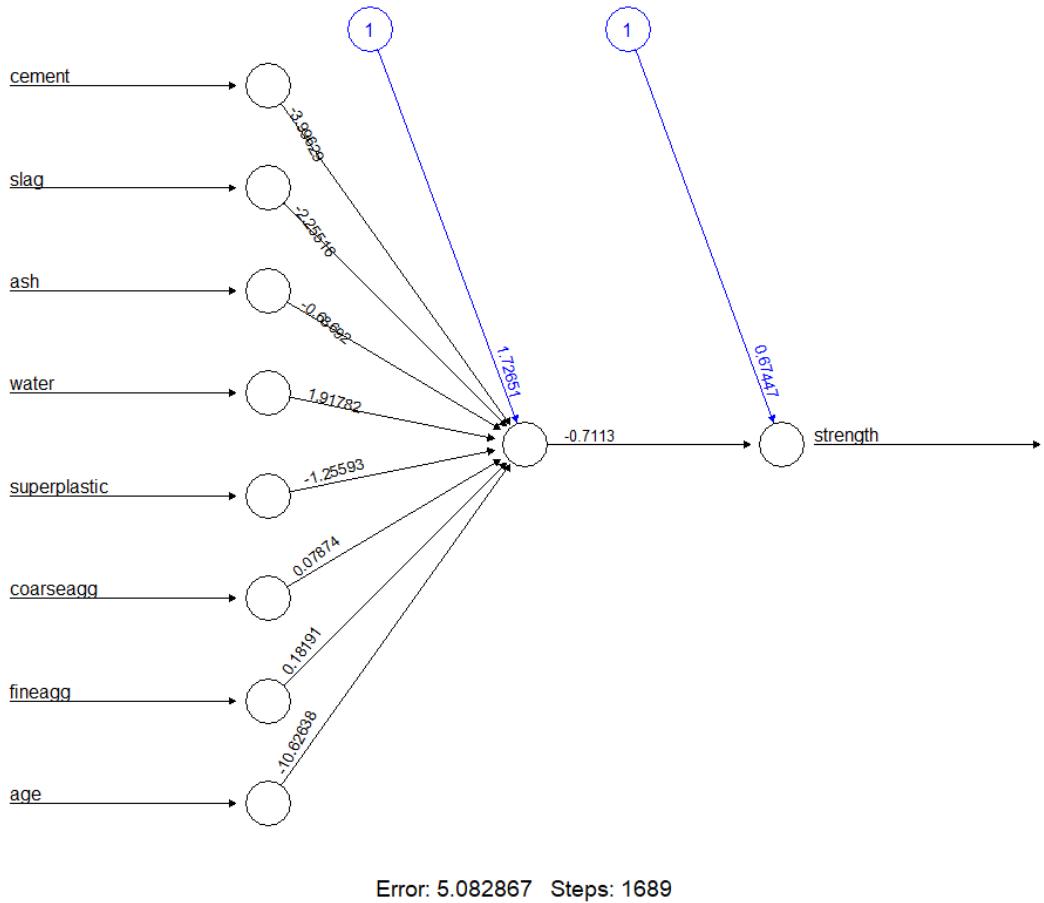
该函数返回一个两元素的列表: \$neurons, 用于保存神经网络每一层的神经元; \$net.result, 用于保存模型的预测值。

例子:

```
concrete_model <- neuralnet(strength ~ cement + slag + ash, data = concrete)
model_results <- compute(concrete_model, concrete_data)$strength_predictions <-
model_results$net.result
```

```
1 > library(neuralnet)
2 > concrete_model <- neuralnet(formula = strength ~ cement + slag +
3 +                               ash + water + superplastic +
4 +                               coarseagg + fineagg + age,
5 +                               data = concrete_train)
6 > plot(concrete_model)
```

在这个简单的模型中, 对于 8 个特征中的都有一个输入节点, 后面跟着一个单一的隐藏节点和一个单一的预测混凝土强度的输出节点. 每一个链接的权重, 也都被, 描绘出来, 偏差项也被描绘出来 (通常带有 1 的节点表示). 该图还报告了训练的步数和一个称为误差平方和 (sum of squared error, SSE) 的度量指标. 当评估模型的性能时, 这些指标很有用.



7.2.4 第四步—评估模型性能

为了评估模型的性能, 可以基于测试数据集使用 `compute()` 函数生成预测;

注意, `compute()` 函数的运行原理与我们之前一直使用的 `predict()` 函数有些不同. 它返回一个带有两个分量的列表:`$neurons`, 用来存储网络中每一层的神经元; `$net.results` 用来存储预测值. 我们想要后者.

注意这是数值预测问题, 不是分类问题, 所以不能用混淆矩阵来检查模型的准确性.

```

1 > model_results <- compute(concrete_model, concrete_test[1:8])
2
3 > predicted_strength <- model_results$net.result
4 > cor(predicted_strength, concrete_test$strength)
5      [,1]
6 [1,] 0.8059085

```

如果结果不同, 莫方. 因为神经网络开始于随机的权重, 所以模型之间的预测值可以有所不同.

0.8 的相关性, 非常不错了.

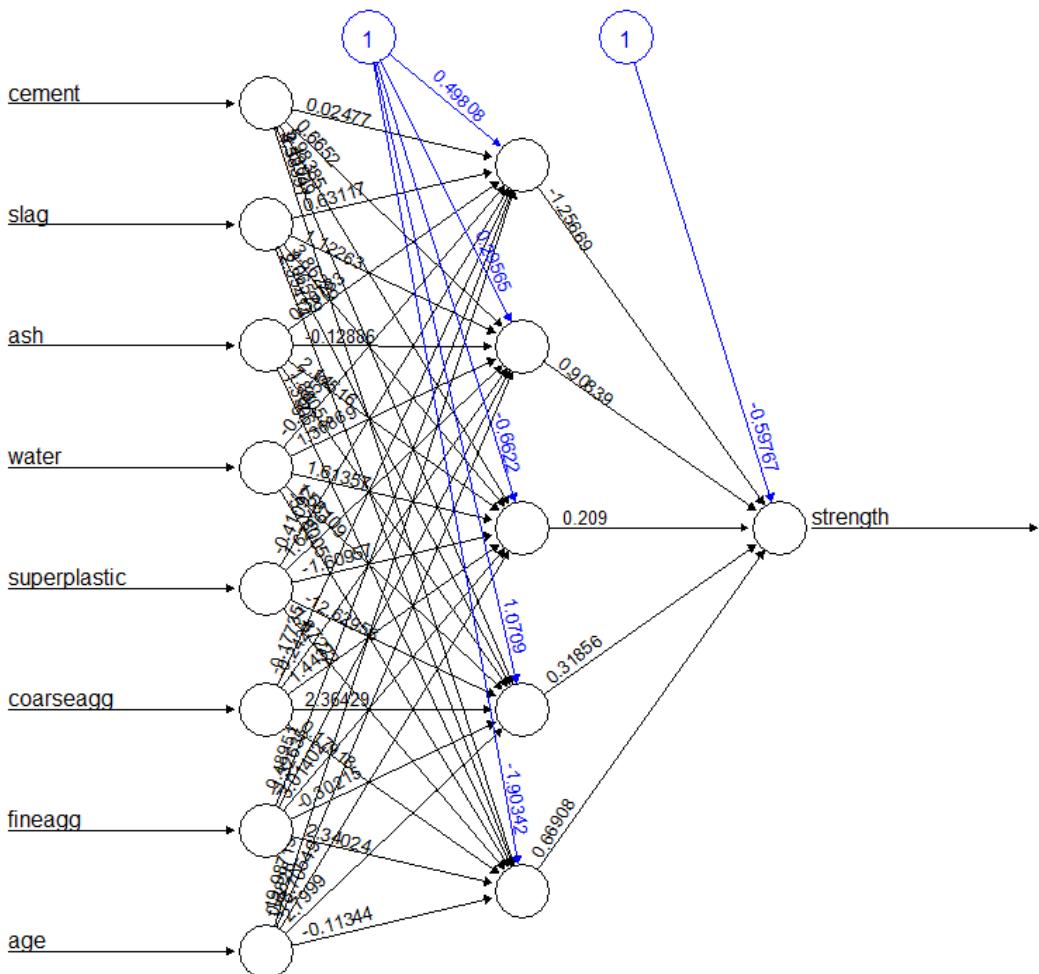
考虑到只使用了一个隐藏节点, 因此很有可能提高我们模型的性能. 让我们试着建立一个更好的模型.

7.2.5 第五步—提高模型的性能

```

1 > concrete_model2 <- neuralnet(strength ~ cement + slag +
2 + ash + water + superplastic +
3 + coarseagg + fineagg + age,
4 + data = concrete_train, hidden = 5)
5 >
6 > plot(concrete_model2)

```



注意所报告的误差 (依然是通过 SSE 度量) 已经从之前模型的 5 点几减少为 1.77; 此外, 训练步数从 3222 步上升为 7230 步.

```

1 > model_results2 <- compute(concrete_model2, concrete_test[1:8])
2 > predicted_strength2 <- model_results2$net.result
3 > cor(predicted_strength2, concrete_test$strength)
4 [1,]
5 [1,] 0.9222841

```

相关系数也从 0.8 上升到 0.9, 这是一个相当大的改进.

7.3 理解支持向量机

支持向量机 (support vector machine, SVM) 可以想象成一个平面, 该平面定义了各个数据点之间的界限, 而这些数据点代表根据它们的特征值绘制在多维空间中的案例. 支持向量机的目标是创建一个平面边界, 称为超平面 (hyperplane), 使得任何一边的数据划分都是相当均匀的. 通过这种方式, 支持向量机学习结合了第三章中呈现的基于实例的近邻学习和第六章中描述的线性回归建模两个方面, 这种结合是相当强大的, 允许支持向量机对非常复杂的关系进行建模.

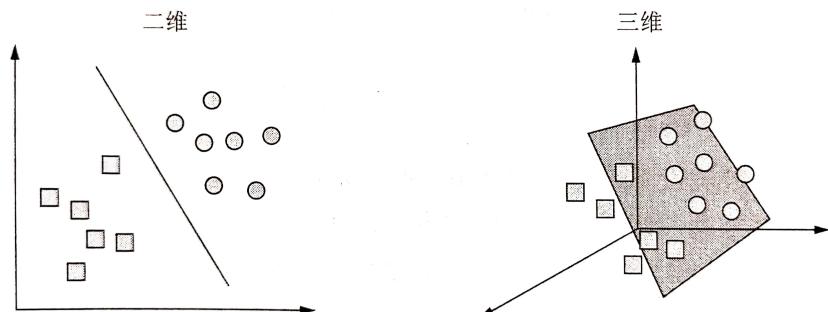
支持向量机几乎可以适用于所有的学习任务, 包括分类和数值预测两个方面. 许多算法成功的关键都是来自模式识别. 著名的应用包括:

- 在生物信息学领域, 识别癌症或者其他遗传疾病的微阵列基因表达数据的分类.
- 文本分类, 比如根据主题鉴定用于某个文档或者组织文档的语言.
- 罕见却重要的事件检测, 如内燃机故障, 安全漏洞, 或者地震.

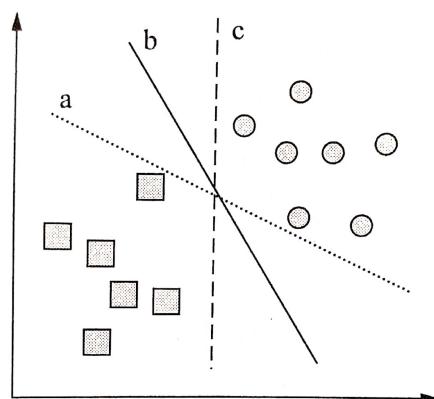
当支持向量机用于二元分类时, 它最容易理解, 这就是为什么该方法已经被习惯应用的原因. 因此, 在剩下的部分, 我们将只专注于支持向量机分类器. 当支持向量机适用于其他学习任务如数值预测时, 这里的学习原则同样适用.

7.3.1 用超平面分类

由于圆形和正方形可以由一条直线或者一个平面进行划分, 所以它们是线性可分的 (linear separable). 支持向量机可以扩展到数据不是线性可分的问题.



支持向量机算法的任务即使确定一条用于分割两个类别的线.

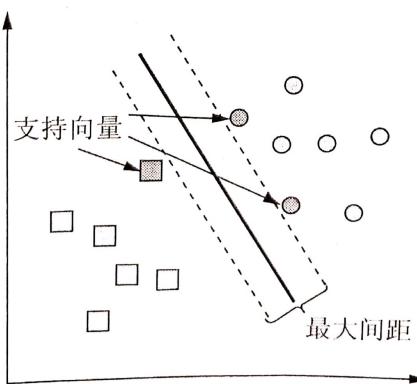


分割线不止一种选择, 支持向量机怎么选呢?

7.3.2 寻找最大间隔

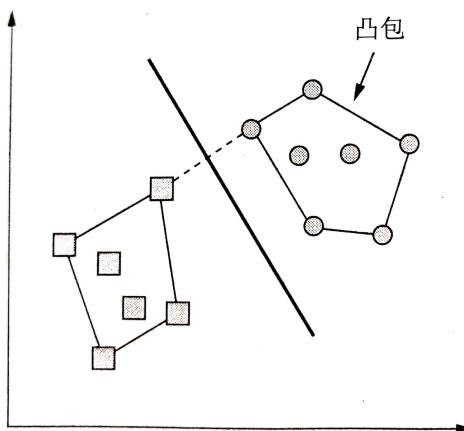
回答上面的问题涉及寻找创建两个类之间最大间隔的最大间隔超平面 (maximum margin hyperplane, MMH), 尽管分隔圆形和正方形的 3 条线中的任意一条都将对所有的数据点进行正确分类, 但很有可能产生最大间隔的那条线将能够最好的推广到将来的数据. 这是因为在边界附近点位置的微小变化可能会导致这些点中的某些点碰巧落在线之外.

支持向量 (support vector)(由下图中的箭头所示) 是每个类中最接近最大间隔超平面的点, 而且每类必须至少有一个支持向量, 但也可能有多个. 单独使用支持向量, 就可以定义最大间隔超平面, 这是支持向量机的一个重要特征. 支持向量机提供了一种非常简洁 (紧凑) 的方式来存储分类模型, 即使特征的个数非常多.



7.3.2.1 线性可分的数据情况

在类是线性可分的假设下, 如何找到最大间隔是最容易理解的. 在这种情况下, 最大间隔超平面要尽可能地远离两组数据点的外边界. 这些外边界称为凸包 (convex hull). 然后, 最大间隔超平面就是两个凸包之间最短距离直线的垂直平分线. 使用一种称为二次优化 (quadratic optimization) 的技术, 复杂的计算机算法就能够通过这种方式找到最大间隔. 另一种等价的替代方法涉及通过每一个可能的超平面的空间搜索, 从而找到一组将数据划分成同类组的两个平行平面, 但这两个平面本身却要尽可能地远离.



要理解这个搜索过程, 需要通过一个超平面来确切定义上面叙述的过程. 在 n 维空间中, 使用下面

的方程:

$$\vec{w} \cdot \vec{x} + b = 0 \quad (7.2)$$

利用这个公式, 该过程的目标就是找到一组指定两个超平面的权重, 如下所示:

$$\vec{w} \cdot \vec{x} + b \geq +1 \quad (7.3)$$

$$\vec{w} \cdot \vec{x} + b \leq -1 \quad (7.4)$$

同样, 我们将要求这两个指定的超平面使得第一类中所有的点落在第一个超平面的上方, 另一类中所有的点落在第二个超平面的下方. 只要数据是可分的, 这样做就是可能的.

向量几何定义这两个平面之间的距离为:

$$\frac{2}{\|\vec{w}\|} \quad (7.5)$$

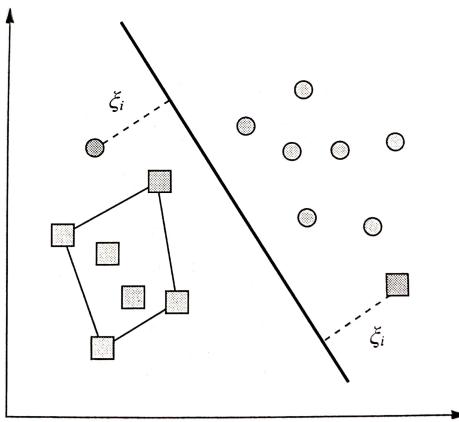
这里, $\|\vec{w}\|$ 表示欧几里得范数 (Euclidean norm), 因此要最大化距离, 我们需要最小化 $\|\vec{w}\|$. 为了便于找到解决方案, 这类任务通常重新表述为一组约束:

$$\min \frac{1}{2} \|\vec{w}\|^2, \text{ s.t. } y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \forall \vec{x}_i \quad (7.6)$$

这种想法就是使得每一个数据点 y_i 正确分类的条件下, 最小化前面的公式. 注意, y 表示分类值 (转换为 +1. 或 -1).

7.3.2.2 非线性可分的数据情况

当研究过支持向量背后的理论后, 你可能想知道: 在数据不是线性可分的情况下, 会发生什么? 这个问题的解决方案使用了一个松弛变量 (slack variable), 这样就创建了一个软间隔, 允许一些点落在在线不正确的一边. 下图显示了两个点落在了与松弛项 (ξ_i) 相对应的线的错误边:



用成本值 (C) 表示所有违反约束的点, 而且该算法试图使总成本最小, 而不是寻找最大间隔. 因此可以修正优化问题:

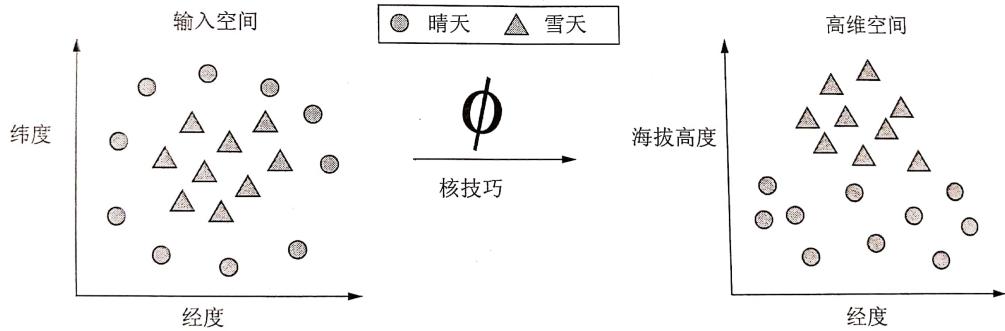
$$\min \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^n \xi_i, \text{ s.t. } y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 - \xi_i, \forall \vec{x}_i, \xi_i \geq 0 \quad (7.7)$$

需要理解的重要一点是增加的成本参数 C , 修改这个值将调整对于落在超平面错误一边的案例的惩罚. 成本参数越大, 努力实现 100% 分离的优化就会越困难. 另一方面, 较小的成本参数将把重点放在更宽泛的整体边缘. 为了创建更好的概括未来数据的模型, 在这两者之间取得平衡是非常重要的.

7.3.3 对非线性空间使用核函数

现实世界中变量之间的关系很多都是非线性的. 以上的方法并不是处理非线性问题的唯一方式. 支持向量机的一个关键特征就是它们能够使用一种称为核技巧 (kernel trick) 的处理方式将问题映射到一个更高维的空间中. 如果这样做, 非线性关系可能会突然看起来是完全线性的.

例如:



具有非线性核的支持向量通过对数据添加额外的维度, 以便以这种方式创建分离.

优点:

- 可用于分类或者数值预测问题
- 不会过多的受到噪声数据的影响, 而且不容易出现过度拟合.
- 可能比神经网络更容易使用, 特别是由于几个得到很好支持的支持向量机算法存在.
- 准确度高

缺点:

- 寻找最好的模型需要测试不同的核函数和模型参数的组合
- 训练缓慢, 尤其是输入数据具有大量的特征或者案例
- 导致一个复杂的黑箱模型

一般情况下, 核函数是下面的形式. 这里, 该函数用 $\Phi(x)$, 是一个将数据转换到另一个空间的映射:

$$K(\vec{x}_i, \vec{x}_j) = \Phi(\vec{x}_i) \cdot \Phi(\vec{x}_j) \quad (7.8)$$

最常用的几个核函数如下, 几乎所有的支持向量机软件包都将包括这些核:

线性核函数 (linear kernel):

$$K(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j \quad (7.9)$$

d 次多项式核函数 (polynomial kernel):

$$K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i \cdot \vec{x}_j + 1)^d \quad (7.10)$$

S 形核函数 (sigmoid kernel):

$$K(\vec{x}_i, \vec{x}_j) = \tanh(k\vec{x}_i \cdot \vec{x}_j - \delta) \quad (7.11)$$

高斯 RBF 核函数 (Gaussian RBF kernel):

$$K(\vec{x}_i, \vec{x}_j) = e^{-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{2\sigma^2}} \quad (7.12)$$

类似于 RBF 神经网络, RBF 核函数对于许多类型的数据都运行的很好, 且被认为是用于许多学习任务的一个合理开始

对于特定的任务, 没有可以依赖的规则用于匹配的核函数. 在很大程度上拟合取决于要学习的概念以及训练数据的量和特征之间的关系. 通常情况下, 一点点的试验和误差需要基于验证数据集和评估多个支持向量机. 即, 在许多情况下, 核函数的选择是任意的, 因为性能可能只有轻微的变化.

7.4 用支持向量机进行光学字符识别

支持向量机非常适合处理图像数据带来的挑战, 他们能够学习复杂的图按而不需要对噪声过度敏感.

在本节中, 我们将研究一个模型, 该模型类似于那些往往与桌面文档扫描仪捆绑在一起的光学字符识别 (optical character recognition,OCR) 软件的核心模型. 此类软件的目的是通过将印刷或者手写文本转换成一种电子形式, 保存在数据库中用来处理纸质文件.

7.4.1 第一步—收集数据

当光学字符识别软件第一次处理文件时, 它将文件划分成一个矩阵, 从而网格中的每一个单元包含一个单一的图像字符 (glyph), 这是一种适用于字母, 符号或者数字的好方式. 接着, 对每一个单元, 该软件将试图对一组它能识别的所有字符进行图像字符匹配. 最后, 单个字符将重新在一起组合成词, 这可以用文档语言中的字典来有选择地进行拼写检查.

这个练习中, 假设我们已经开发了将文件分割成矩形区域, 每一个区域包含一个单一字符的算法; 还假设文件中只包含英文字母字符. 因此, 我们将模拟一个过程, 涉及对从 A~Z 的 26 个字母中的一个进行图像字符匹配.

7.4.2 第二步—探索和准备数据

当图像字符被扫描到计算机中, 它们将转换成像素, 并且有 16 个统计属性.

```
1 > letters<-read.csv("E:/机器学习/ml&r/Machine Learning with R/chapter 7/  
2 letterdata.csv")  
3 > str(letters)  
4 'data.frame': 20000 obs. of 17 variables:  
5 $ letter: Factor w/ 26 levels "A","B","C","D",...: 20 9 4 14 7 19 2 1 10 13 ...  
6 $ xbox : int 2 5 4 7 2 4 4 1 2 11 ...  
7 $ ybox : int 8 12 11 11 1 11 2 1 2 15 ...
```

```

8 $ width : int 3 3 6 6 3 5 5 3 4 13 ...
9 $ height: int 5 7 8 6 1 8 4 2 4 9 ...
10 $ onpix : int 1 2 6 3 1 3 4 1 2 7 ...
11 $ xbar : int 8 10 10 5 8 8 8 8 10 13 ...
12 $ ybar : int 13 5 6 9 6 8 7 2 6 2 ...
13 $ x2bar : int 0 5 2 4 6 6 6 2 2 6 ...
14 $ y2bar : int 6 4 6 6 6 9 6 2 6 2 ...
15 $ xybar : int 6 13 10 4 6 5 7 8 12 12 ...
16 $ x2ybar: int 10 3 3 4 5 6 6 2 4 1 ...
17 $ xy2bar: int 8 9 7 10 9 6 6 8 8 9 ...
18 $ xedge : int 0 2 3 6 1 0 2 1 1 8 ...
19 $ xedgey: int 8 8 7 10 7 8 8 6 6 1 ...
20 $ yedge : int 0 4 3 2 5 9 7 2 1 1 ...
21 $ yedgex: int 8 10 9 8 10 7 10 7 7 8 ...

```

回想一下, 支持向量机学习算法需要所有的特征都是数值型的, 而且每一个特征需要缩小到一个相当小的区间. 现在我们的数据都是整数, 而且也不用标准化/规范化, 因为用来拟合支持向量机模型的 R 添加包会自动帮助我们对数据进行重新调整.

数据已经随机化, 用前 80% 建立模型, 后 20% 进行测试.

```

1 > letters_train <- letters[1:16000, ]
2 > letters_test <- letters[16001:20000, ]

```

7.4.3 第三步—基于数据训练模型

关于 R 中拟合支持向量机模型时, 如果你熟悉 LIBSVM, e1071 包提供了一个用 C++ 编写的开源支持向量机程序. 如果你会 SVMlight 算法, 那么 klaR 包提供了 R 中该支持向量机的函数实现. 如果你是从头开始, 用 kernlab 包的支持向量机函数. 这个包的优点是本就在 R 中开发, 容易设置, 无任何隐藏的幕后结构. 或许更重要的是,kernlab 包可以与 caret 包一起使用, 这就允许支持向量机模型可以使用各种自动化方法进行训练的评估 (11 章).

默认情况下,ksvm() 函数使用高斯 RBF 核函数, 但也提供其他一些选项.

我们调用 ksvm() 函数, 并使用 vanilladot 选项指定线性核函数 (即 vanilla)

支持向量机语法

应用 kernlab 添加包中的 ksvm() 函数

建立模型:

```
m <- ksvm(target ~ predictors, data = mydata, kernel = "rbfdot", c = 1)
```

- **target**: 是数据框 `mydata` 中需要建模的输出变量
- **predictors**: 是给出数据框 `mydata` 中用于预测的特征的一个 R 公式
- **data**: 给出包含变量 `target` 和 `predictors` 的数据框
- **kernel**: 给出一个非线性映射 (mapping), 例如 "rbfdot" (径向基函数), "polydot" (多项式函数), "tanhdot" (双曲正切函数), "vanilladot" (线性函数)
- **c**: 用于给出违法约束条件时的惩罚, 即对于 "软边界" 的惩罚的大小。较大的 `c` 值将导致较窄的边界。
该函数返回一个可以用于预测的 SVM 对象。

进行预测:

```
p <- predict(m, test, type = "response")
```

- **m**: 函数 `ksvm()` 所训练的模型
- **test**: 包含测试数据的数据框, 它具有和用于训练模型的训练数据相同的特征
- **type**: 用于指定预测的类型为 "response" (预测类别), 或者 "probabilities" (预测概率, 每一列对应一个类水平值)

根据 `type` 参数的设定, 该函数返回一个包含预测类别 (或者概率) 的向量 (或者矩阵)。两元素的列表: `$neurons`, 用于保存神经网络每一层的神经元; `$net.result`, 用于保存模型的预测值。

例子:

```
letter_classifier <- ksvm(letter ~ ., data = letters_train, kernel = "vanilladot")
letter_prediction <- predict(letter_classifier, letters_test)
```

```
1 > library(kernlab)
2 > letter_classifier <- ksvm(letter ~ ., data = letters_train,
3 +                               kernel = "vanilladot")
4 Setting default kernel parameters
5 > letter_classifier
6 Support Vector Machine object of class "ksvm"
7
8 SV type: C-svc (classification)
9 parameter : cost C = 1
10
11 Linear (vanilla) kernel function.
12
13 Number of Support Vectors : 7037
14
15 Objective Function Value : -14.1746 -20.0072 -23.5628 -6.2009 -7.5524
16 -32.7694 -49.9786 -18.1824 -62.1111 -32.7284 -16.2209 -32.2837
17 -28.9777 -51.2195 -13.276 -35.6217 -30.8612 -16.5256 ...
18 Training error : 0.130062
```

这些信息几乎没有告诉我们关于模型在现实世界中的运行好坏程度。

7.4.4 第四步—评估模型的性能

用 predict 函数基于测试数据集使用字母分类模型进行预测:

```
1 > letter_predictions <- predict(letter_classifier, letters_test)
```

没有指定 type 参数, 所以默认使用 type="response"(返回预测类别), 这样就返回了一个向量.

用 head() 看看前几个预测结果:

```
1 > head(letter_predictions)
2 [1] U N V X N H
3 Levels: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

将测试数据集中的预测值和真实值进行比较:

```
1 > table(letter_predictions, letters_test$letter)
2
3 letter_predictions
4   A   B   C   D   E   F   G   H   I   J   K   L   M   N
5 A 144   0   0   0   0   0   0   0   0   1   0   0   1   2
6 B   0 121   0   5   2   0   1   2   0   0   1   0   1   0
7 C   0   0 120   0   4   0  10   2   2   0   1   3   0   0
8 D   2   2   0 156   0   1   3  10   4   3   4   3   0   5
9 E   0   0   5   0 127   3   1   1   0   0   3   4   0   0
10 F   0   0   0   0 138   2   2   6   0   0   0   0   0   0
11 G   1   1   2   1   9   2 123   2   0   0   1   2   1   0
12 H   0   0   0   1   0   1   0 102   0   2   3   2   3   4
13 I   0   1   0   0   0   1   0   0 141   8   0   0   0   0
14 J   0   1   0   0   0   1   0   2   5 128   0   0   0   0
15 K   1   1   9   0   0   0   2   5   0   0 118   0   0   2
16 L   0   0   0   0   2   0   1   1   0   0   0 133   0   0
17 M   0   0   1   1   0   0   1   1   0   0   0   0 135   4
18 N   0   0   0   0   0   1   0   1   0   0   0   0   0 145
19 O   1   0   2   1   0   0   1   2   0   1   0   0   0   1
20 P   0   0   0   1   0   2   1   0   0   0   0   0   0   0
21 Q   0   0   0   0   0   0   8   2   0   0   0   3   0   0
22 R   0   7   0   0   1   0   3   8   0   0 13   0   0   1
23 S   1   1   0   0   1   0   3   0   1   1   0   1   0   0
24 T   0   0   0   0   3   2   0   0   0   0   1   0   0   0
25 U   1   0   3   1   0   0   0   2   0   0   0   0   0   0
26 V   0   0   0   0   0   1   3   4   0   0   0   0   1   2
27 W   0   0   0   0   0   0   1   0   0   0   0   0   0   2
28 X   0   1   0   0   2   0   0   1   3   0   1   6   0   0
29 Y   3   0   0   0   0   0   0   1   0   0   0   0   0   0
30 Z   2   0   0   0   1   0   0   0   3   4   0   0   0   0
```

```

31
32 letter_predictions
33   O   P   Q   R   S   T   U   V   W   X   Y   Z
34 A   2   0   5   0   1   1   1   0   1   0   0   1
35 B   0   2   2   3   5   0   0   2   0   1   0   0
36 C   2   0   0   0   0   0   0   0   0   0   0   0
37 D   5   3   1   4   0   0   0   0   0   3   3   1
38 E   0   0   2   0   10  0   0   0   0   2   0   3
39 F   0   16  0   0   3   0   0   1   0   1   2   0
40 G   1   2   8   2   4   3   0   0   0   1   0   0
41 H   20  0   2   3   0   3   0   2   0   0   1   0
42 I   0   1   0   0   3   0   0   0   0   5   1   1
43 J   1   1   3   0   2   0   0   0   0   1   0   6
44 K   0   1   0   7   0   1   3   0   0   5   0   0
45 L   0   0   1   0   5   0   0   0   0   0   0   1
46 M   0   0   0   0   0   0   3   0   8   0   0   0
47 N   0   0   0   3   0   0   1   0   2   0   0   0
48 O   99  3   3   0   0   0   3   0   0   0   0   0
49 P   2   130 0   0   0   0   0   0   0   0   1   0
50 Q   3   1   124 0   5   0   0   0   0   0   2   0
51 R   1   1   0   138 0   1   0   1   0   0   0   0
52 S   0   0   14  0   101 3   0   0   0   2   0   10
53 T   0   0   0   0   3   133 1   0   0   0   2   2
54 U   1   0   0   0   0   0   152 0   0   1   1   0
55 V   1   0   3   1   0   0   0   126 1   0   4   0
56 W   0   0   0   0   0   0   4   4   127 0   0   0
57 X   1   0   0   0   1   0   0   0   0   137 1   1
58 Y   0   7   0   0   0   3   0   0   0   0   127 0
59 Z   0   0   0   0   18  3   0   0   0   0   0   132

```

对角线的值 144,121,120,... 表示预测值与真实值相匹配的总记录数. 同样, 错误的数目也列出来了.

我们通过计算整体的准确度来简化我们的评估, 即只考虑预测的字母是正确与否, 并忽略错误的类型.

下面的命令返回一个元素为 true 或 false 值的向量, 表示是否相匹配:

```

1 > agreement <- letter_predictions == letters_test$letter
2 > table(agreement)
3
4 agreement
5 FALSE  TRUE
6 643 3357
7 > prop.table(table(agreement))
8
9 agreement

```

```
8 FALSE      TRUE
9 0.16075  0.83925
```

准确率约为 84%.

7.4.5 第五步—提高模型的性能

通过使用一个更复杂的核函数, 刻有将数据映射到一个更高维的空间, 并且有可能获得一个较好的模型拟合度.

我们用 `ksvm()` 函数训练一个基于 RBF 的支持向量机, 一个流行的惯例是从高斯 RBF 核函数开始, 因为他已经被证明对于许多类型的数据都能运行的很好.

```
1 > letter_classifier_rbf <- ksvm(letter ~ ., data = letters_train,
2   kernel = "rbfdot")
3 > letter_predictions_rbf <- predict(letter_classifier_rbf, letters_test)
```

最后与我们的线性支持向量机的准确度进行比较:

```
1 > agreement_rbf <- letter_predictions_rbf == letters_test$letter
2 > table(agreement_rbf)
3 agreement_rbf
4 FALSE      TRUE
5 275  3725
6 > prop.table(table(agreement_rbf))
7 agreement_rbf
8 FALSE      TRUE
9 0.06875  0.93125
```

准确度提高到了 93%.

如果这种性能水平对于光学字符识别程序仍不能满意, 那么可以测试其他的核函数或者通过改变成本约束函数 C 来修正决策边界的宽度.

Chapter 8

探寻模式—基于关联规则的购物篮分析

本章介绍用来判别事务型数据种商品之间关联的机器学习方法. 称为 (市场) 购物篮分析 (**market basket analysis**). 你将会学到:

- 使用简单的统计性能指标, 发现大型数据集中有用的关联方法
- 如何管理事务型数据处理的特性
- 利用关联规则对真实世界的数据进行市场购物篮分析所需要的完整步骤.

8.1 理解关联规则

市场购物篮分析的结构是一组指定的商品之间关系模式的**关联规则** (**association rule**). 如

$$\{\text{花生酱, 果冻}\} \rightarrow \{\text{面包}\}$$

大括号内的一件商品或者多件商品的组合表示它们构成一个集合, 更确切地说, 就是出现在具有某种规律性的数据中的**项集** (**itemset**). 关联规则是根据项集的子集研究得到的, 例如上述规则就是由集合 $\{\text{花生酱, 果冻, 面包}\}$ 确定的.

关联规则与前面所学的分类算法和数值预测算法不同, 它不能进行预测, 但可以用于无监督的知识发现, 所以不需要训练算法, 也不需要提前标记数据. 基于数据集就可以简单地运行程序, 希望探寻到令人感兴趣的关联. 不利的一面是除了用从定性的角度来衡量它们的可用性之外 (通常是某种形式的目测法来对它们进行评估), 没有一个简单的方法来客观地衡量一个规则学习算法的性能.

关联规则最常用于市场购物篮分析, 其他潜在应用包括:

- 癌症数据分析, 搜寻 DNA 和蛋白质序列的有趣且频繁出现的模式
- 查找发生在与信用卡欺诈或者保险应用相结合的购物或者医疗津贴的模式
- 确认导致客户放弃他们移动电话服务或者升级它们有线电视服务套餐行为组合.

8.1.1 用于关联规则学习的 **Apriori** 算法

事务型数据一般来说是非常庞大的, 增加难度的是潜在的项集书量随着特征的数量呈指数增长. 给定 k 项, 可以出现在集合中, 或者不出现, 约有 2^k 个可能的项集必须用于规则的搜索.

一个灵敏的规则学习算法利用了这样一个事实: 在现实中, 许多潜在的商品组合极少, 如果有就是在实践中发现, 而不是一个接着一个地评估这些项集中的每个元素. 通过忽略罕见 (并且因此可能不太重要) 的组合, 就可以限制规则的搜索范围, 从而更容易管理项集的规模.

为了减少需要搜索的项集数, 确定启发式算法的大部分工作已经完成. 或许用规则高效的搜索大型数据库的最广泛使用的方法就称为 **Apriori 算法**.

优点:

- 非常适合处理极其大量的事务型数据
- 规则中的结果很容易理解
- 对于“数据挖掘”和发现数据库中意外的知识很有用

缺点:

- 对于小的数据集不是很有用
- 需要努力地将对数据地洞察和常识区分开
- 容易从随机模式得出虚假的结论

Apriori 算法采用一个简单的先验信念作为准则/指引来减少关联规则地搜索空间: 一个频繁项集地所有子集必须也是频繁的. 此启发式称为 **Apriori 性质 (Apriori property)**.

8.1.1.1 度量规则兴趣度—支持度和置信度

关联规则是否令人感兴趣的, 取决于两个统计量: 支持度和置信度. 通过为这些度量的每一个量提供最小的阈值并应用 Apriori 原则, 很容易大幅地限制报告的规则数.

一个项集或者规则度量法地支持度是指其在数据中出现的频率. 任何项集的支持度都可以计算, 甚至是一个单元素的项集.

定义项集 X 的支持度的函数可以定义为:

$$support(X) = \frac{count(X)}{N} \quad (8.1)$$

其中, N 表示数据库中的交易次数, $count(X)$ 表示项集 X 出现在交易中的次数.

规则的置信度 (**confidence**) 是指该规则的预测能力或者准确度的度量

定义为同时包含项集 X 和项集 Y 的支持度与只包含项集 X 的支持度的商:

$$confidence(X \rightarrow Y) = \frac{support(X, Y)}{support(X)} \quad (8.2)$$

本质上讲, 置信度表示交易中项或者项集 X 的出现导致项或者项集 Y 出现的比例. 请记住, X 导致 Y 的置信度与 Y 导致 X 的置信度是不一样的.

如果 $X \rightarrow Y$ 同时具有高支持度和置信度, 那么称这样的规则为强规则 (**strong rule**). 发现更多强规则的一种方法就是检查礼品店中的每一个可能的商品组合, 测量支持度和置信度, 并只报告那些满足某种兴趣水平的规则. 然而这种策略对于大数据集, 一般不可行.

8.1.1.2 用 Apriori 原则建立规则

Apriori 原则指一个频繁项集地所有子集必须也是频繁的. 如果 $\{A, B\}$ 是频繁的, 那么 $\{A\}$ 和 $\{B\}$ 都必须是频繁的. 支持度表示一个项集出现在数据中的频率. 如果知道 $\{A\}$ 不满足所期望的支持度阈值, 那么就没有理由考虑 $\{A, B\}$ 或者任何包含 $\{A\}$ 的项集.

Apriori 算法利用这个逻辑在实际评估它们之间排除潜在的关联规则, 创建规则的实际过程分为两个阶段:

- 识别所有满足最小支持度阈值的项集
- 根据满足最小置信度阈值的这些项集来创建规则.

第一阶段发生在多次迭代中, 每次连续迭代都需要评估存储一组越来越大项集的支持度. 如, 迭代 1 需要评估一组 1 项的项集 (1 项集), 迭代 2 评估 2 项集, 以此类推. 每个迭代 i 的结果是一组所有满足最小支持度阈值的 i 项集.

由迭代 i 得到的所有项集结合在一起以便生成候选项集用于在迭代 $i+1$ 中进行评估. 但是 Apriori 原则甚至可以在下一轮开始之前消除一些其中的项集, 就是消除包含不是频繁的项集的项集.

如果迭代最后没有生成新的项集, 那么算法停止. 此时 Apriori 算法的第二阶段将会开始. 给定一组频繁项集, 根据所有可能的子集产生关联规则. 如 $\{A, B\}$ 将产生候选规则 $\{A\} \rightarrow \{B\}$ 和 $\{B\} \rightarrow \{A\}$, 这些规则将根据最小置信度阈值评估, 不满足将被消除.

8.2 例子—用关联规则确定经常一起购买的食品杂货

我们将根据一家杂货店的事务型数据进行一侧市场购物篮分析. 该技术可以运用于许多不同类型的问题, 从电影推荐, 约会地点, 到寻找药物之间相互作用的危险.

8.2.1 第一步—收集数据

本数据集包含 9835 次交易, 约每天 327 次交易 (在 12 小时的工作日内, 大约每小时交易 30 次). 鉴于零售商的大小适中, 假定他们不是非常管制寻找只应用于特定品牌的商品的规则. 所以所有的品牌名称可以从购买数据中去除. 食品杂货的数量减少到易于管理的 169 个类型.

8.2.2 第二步—探索和准备数据

事务型数据不同于之前我们用的数据, 形式更自由, 一行表示一次交易. 从本质上讲, example 之间的特征可能是不同的.

8.2.2.1 数据准备—为交易数据创建一个稀疏矩阵

采用一个稀疏矩阵 (sparse matrix) 的数据结构 (第四章我们用过来处理文本数据). 稀疏矩阵的每一行表示一次交易, 用列 (即特征) 表示可能出现在消费者购物篮中的每一件商品, 因为我们的杂货店数据有 169 类不同的商品, 所以我们的稀疏矩阵将包含 169 列.

为什么不像我们在大多数分析中所做的那样，将其存储为一个数据框呢？其原因就是，一旦增加额外的交易和商品，传统的数据结构很快就会变得过大而导致内存不足。即使这里使用的事务型数据集相对较小，但是矩阵包含了将近 170 万个单元（元素），其中大部分单元（元素）为零（因此命名为“稀疏”矩阵）。因为存储的所有这些零值没有益处，所以稀疏矩阵实际上在内存中没有存储完整的矩阵，只是存储了由一个商品所占用的单元（元素），这使得该结构的内存效率比一个大小相当的矩阵或者数据框的内存效率更高。

为了根据事务型数据创建稀疏矩阵的数据结构，使用由关联规则 (arules) 包提供的函数

```
1 > library(arules)
2 > library(Matrix)
3 > groceries<-read.transactions("E:/机器学习/ml&r/Machine Learning with R/
4 chapter 8/groceries.csv",sep=",")
```

参数 `sep=","` 指定输入到文件中的项之间用逗号隔开。

```
1 > summary(groceries)
2 transactions as itemMatrix in sparse format with
3 9835 rows (elements/itemsets/transactions) and
4 169 columns (items) and a density of 0.02609146
5 #以上两行是稀疏矩阵概要，9835次交易，169类不同商品。
6 #被买了则矩阵单元格为1，否则为0
7 #密度density0.026指的是非零矩阵单元格的比例。
8
9 #下面列出的是事务型数据中最常购买的商品
10 most frequent items:
11 whole milk other vegetables          rolls/buns          soda
12      2513                1903                1809                1715
13 yogurt          (Other)
14      1372                34055
15
16 #下面是一组关于交易规模的统计
17 #总共2159次交易只有一件商品，而有一次交易包含了32类商品。
18 element (itemset/transaction) length distribution:
19 sizes
20   1    2    3    4    5    6    7    8    9    10   11   12   13   14
21 2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77
22
23   15   16   17   18   19   20   21   22   23   24   26   27   28   29
24   55   46   29   14   14    9   11    4    6    1    1    1    1    3
25
26 32
```

```

27  1
28
29  Min. 1st Qu. Median     Mean 3rd Qu.     Max.
30 1.000  2.000  3.000  4.409  6.000  32.000
31
32 includes extended item information - examples:
33 labels
34 1 abrasive cleaner
35 2 artif. sweetener
36 3 baby cosmetics

```

arules() 包含了一些用于检查交易数据的有用功能. 用 inspect() 与向量运算的组合, 可以查看稀疏矩阵的内容:

```

1 > inspect(groceries[1:5])
2 items
3 [1] {citrus fruit,
4      margarine,
5      ready soups,
6      semi-finished bread}
7 [2] {coffee,
8      tropical fruit,
9      yogurt}
10 [3] {whole milk}
11 [4] {cream cheese,
12      meat spreads,
13      pip fruit,
14      yogurt}
15 [5] {condensed milk,
16      long life bakery product,
17      other vegetables,
18      whole milk}

```

若要研究一件特定商品 (即一列数据), 可以使用 row,column 矩阵概念, 同时与 itemFrequency() 一起用, 可以看到包含该商品的交易比例:

```

1 > itemFrequency(groceries[, 1:3])
2 abrasive cleaner artif. sweetener baby cosmetics
3 0.0035587189      0.0032536858      0.0006100661

```

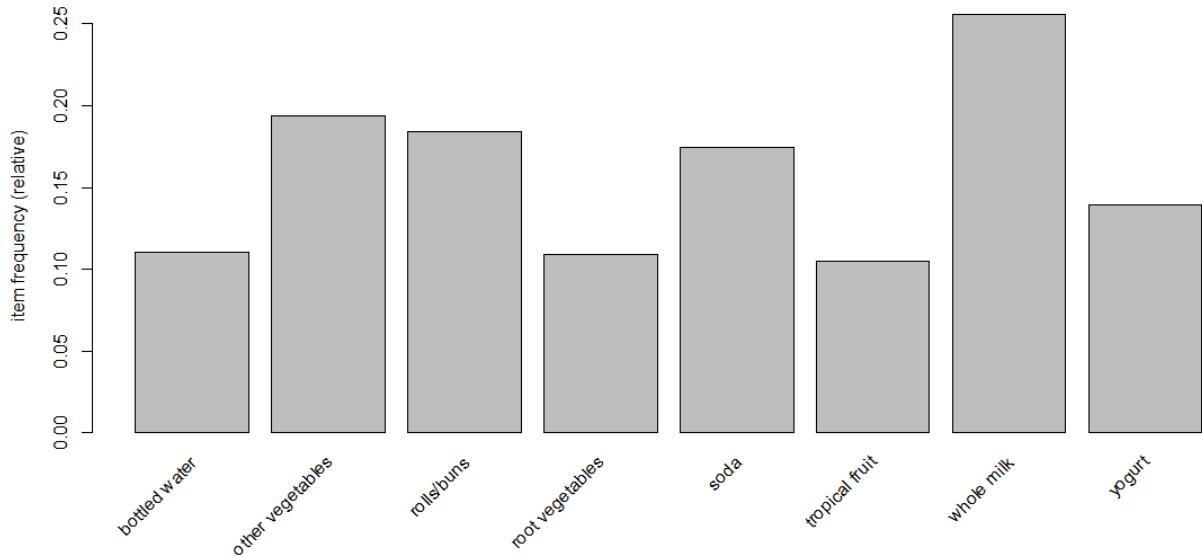
注意稀疏矩阵中商品所在列是按字母表的顺序排序的.

8.2.2.2 可视化商品的支持度—商品的频率图

用 itemFrequencyPlot() 生成一个包含指定商品的交易比例的柱状图. 如果你希望获得那些出现在最小交易比例中的商品, 可以增加参数 support.

```
1 > itemFrequencyPlot(groceries, support = 0.1)
```

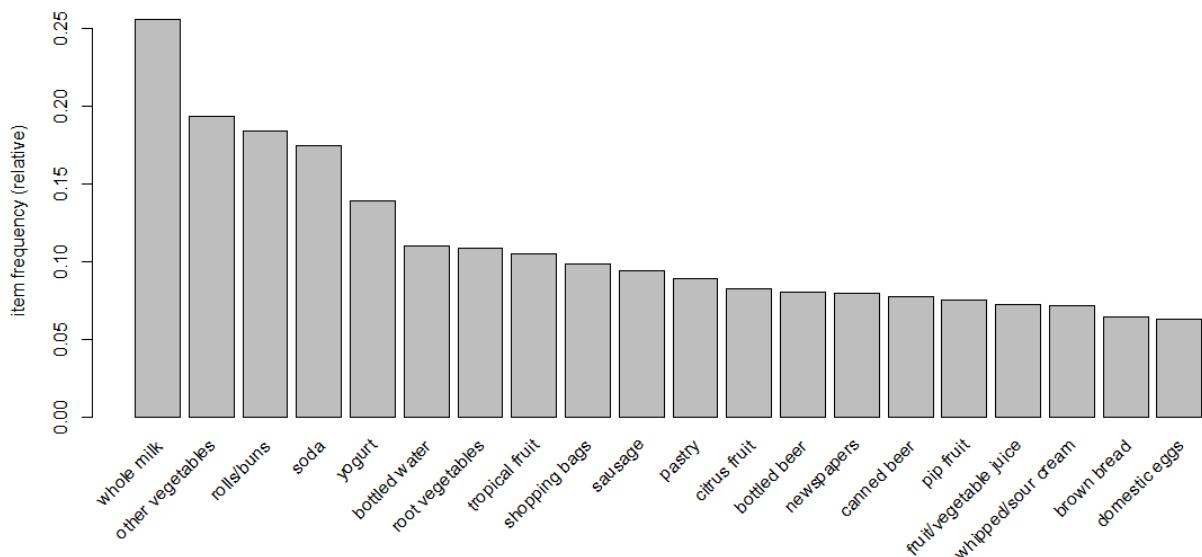
下图显示了 groceries 数据中支持度至少为 10% 的 8 类商品.



如果更愿意限制图中商品的数量, 可以用 topN 参数:

```
1 > itemFrequencyPlot(groceries, topN = 20)
```

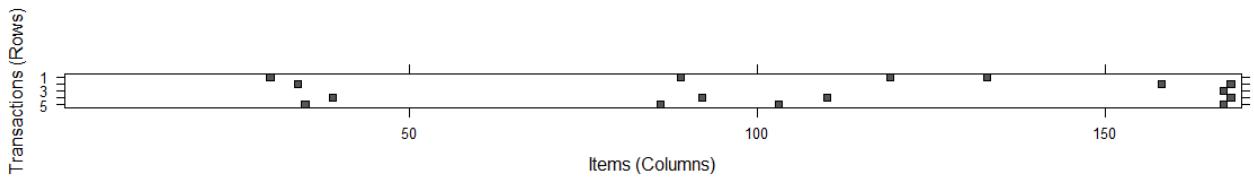
前 20 类商品:



8.2.2.3 可视化交易数据—绘制稀疏矩阵

可视化整个稀疏矩阵用 `image()`

```
1 > image(groceries[1:5])
```



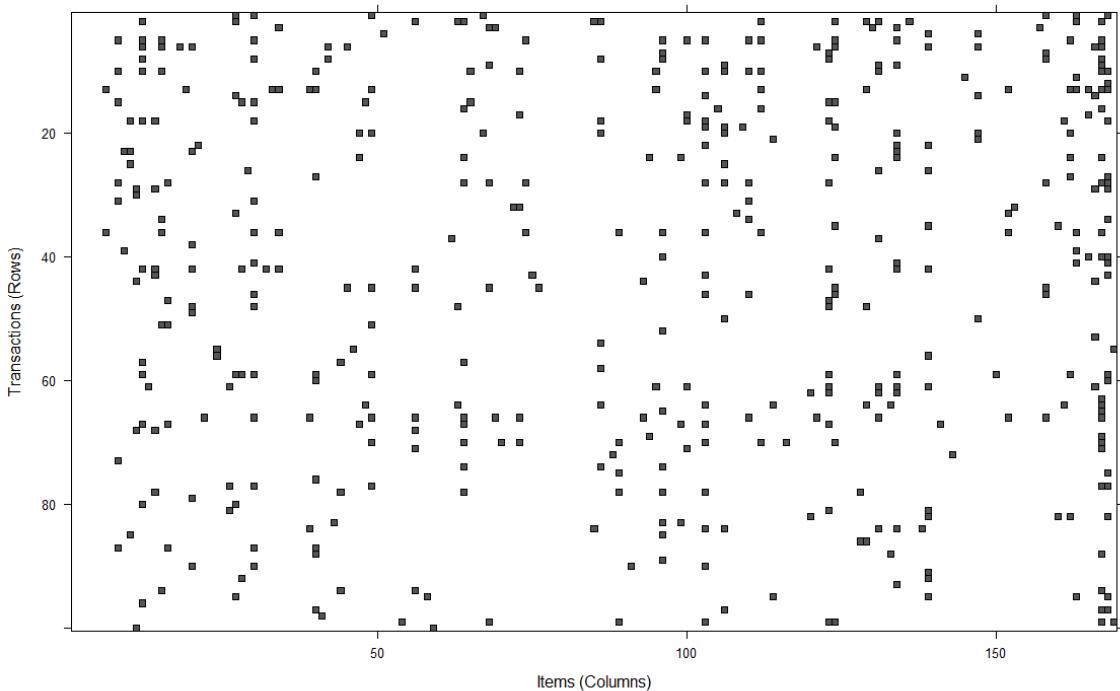
生成的图描绘了 5 行 169 列的矩阵, 黑色表示被购买.

这种可视化是用于探索数据的一种很有用的工具. 一方面可能有助于识别潜在数据问题. 列从上往下被填充可能表明这些商品在每一次交易中都被购买了—但如果一个零售商的名称或者他们的标识号以外的包含在一个数据集中, 或许会产生问题.

另一方面, 图中的模式可能有助于解释交易或者商品的有趣的部分, 特别是当数据以有趣的方式排序. 如按日期排序, 黑色圆点可能揭示购买数量或类型受节日季节的影响. 如果商品也被分类, 那么这种类型的可视化可能会特别有效.

请记住这种可视化对于超大型交易数据是没有用的, 因为单元太小而无法识别. 不过通过与 `sample()` 结合, 可以看到稀疏矩阵中一组随机抽样的交易:

```
1 > image(sample(groceries, 100))
```



8.2.3 第三步-基于数据训练模型

我们将使用 arules 包中的 Apriori 算法实现:

关联规则语法

应用 arules 添加包中的函数 apriori()

找出关联规则:

```
myrules <- apriori( data = mydata,
                      parameter = list(support = 0.1,
                                        confidence = 0.8,
                                        minlen = 1) )
```

- data: 含有交易数据的稀疏矩阵
- support: 给出要求的最低规则支持度
- confidence: 给出要求的最低规则置信度
- minlen: 给出要求的规则最低项数

该函数将返回一个满足最低准则要求的规则对象。

检验关联规则:

```
inspect(myrules)
```

- myrules 是由函数 apriori() 函数给出的一组关联规则。

它将输出关联规则到屏幕。可以对 myrules 应用向量运算来选择查看一个或者多个特定的规则。

例子:

```
concrete_model <- neuralnet(strength ~ cement + slag + ash, data = concrete)
model_results <- compute(concrete_model, concrete_data)
strength_predictions <- model_results$net.result
```

虽然运行 apriori() 函数很简单, 但是当找到支持度和置信度参数来产生合理数量的关联规则时, 有时候可能需要进行大量的试验与误差评估。如果你将这些参数设置过高, 那么你可能会发现没有规则或者规则过于普通而不是非常有用。另一方面, 阈值太低可能会导致规则的数量庞大, 或者更糟糕的是, 该算法可能需要很长时间运行或者在学习阶段耗尽内存。

若试图使用默认设置:support=0.1 和 confidence=0.8, 最终的得不到任何规则:

```
1 > library(arules)
2 >
3 > apriori(groceries)
4 Apriori
5
6 Parameter specification:
7 confidence minval smax arem  aval originalSupport maxtime support
8 0.8      0.1      1 none FALSE           TRUE      5      0.1
9 minlen maxlen target  ext
10 1      10  rules TRUE
11
12 Algorithmic control:
13 filter tree heap memopt load sort verbose
14 0.1 TRUE TRUE FALSE TRUE      2      TRUE
15
```

```

16 Absolute minimum support count: 983
17
18 set item appearances ...[0 item(s)] done [0.00s].
19 set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
20 sorting and recoding items ... [8 item(s)] done [0.00s].
21 creating transaction tree ... done [0.00s].
22 checking subsets of size 1 2 done [0.00s].
23 writing ... [0 rule(s)] done [0.00s].
24 creating S4 object ... done [0.00s].
25 set of 0 rules

```

$9835 \times 0.1 = 983.5$, 而在我们数据中, 只有八类商品出现得比较频繁. 要事先想好需要的最小交易数量. 如认为如果一种商品一天购买了两次 (约有 60 种), 那么值得考虑看看 $60/9835 = 0.006$

设定最小置信度涉及一个巧妙地平衡, 一方面如果置信度太低, 就可能会被大量不可靠的规则淹没, 另一方面如果设置太高, 那么我们将会被显而易见或者不可避免地规则所限制.

我们将置信度阈值 0.25 开始, 意味着为了将规则包含在结果当中, 此时规则地正确率至少为 25%. 这将消除最不可靠的规则, 同时为我们有针对性的营销修正行为提供了一些空间.

同时设定 minlen=2 有助于消除包含少于两类商品的规则. 这可以防止仅仅是由于某商品被频繁购买而创建的无趣规则.

```

1 > groceryrules <- apriori(groceries, parameter = list(support =
2 +                               0.006, confidence = 0.25, minlen = 2))
3 Apriori
4
5 Parameter specification:
6 confidence minval smax arem  aval originalSupport maxtime support
7 0.25      0.1      1 none FALSE           TRUE      5    0.006
8 minlen maxlen target  ext
9 2        10    rules TRUE
10
11 Algorithmic control:
12 filter tree heap memopt load sort verbose
13 0.1 TRUE TRUE FALSE TRUE    2    TRUE
14
15 Absolute minimum support count: 59
16
17 set item appearances ...[0 item(s)] done [0.00s].
18 set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
19 sorting and recoding items ... [109 item(s)] done [0.00s].
20 creating transaction tree ... done [0.00s].
21 checking subsets of size 1 2 3 4 done [0.00s].
22 writing ... [463 rule(s)] done [0.00s].

```

```

23 creating S4 object ... done [0.00s].
24 > groceryrules
25 set of 463 rules

```

463 个关联规则, 为了确定是否是有用的, 我们必须深入挖掘.

8.2.4 第四步-评估模型性能

```

1 > summary(groceryrules)
2 set of 463 rules
3
4 #长度分布告诉我们按商品的每一类计数的规则有多少
5 rule length distribution (lhs + rhs):sizes
6 2   3   4
7 150 297 16
8 #在我们的规则集中, 有150个规则只包含了两类商品
9
10 Min. 1st Qu. Median      Mean 3rd Qu.      Max.
11 2.000 2.000 3.000 2.711 3.000 4.000
12
13 summary of quality measures:
14 support           confidence           coverage           lift
15 Min. :0.006101   Min. :0.2500   Min. :0.009964   Min. :0.9932
16 1st Qu.:0.007117 1st Qu.:0.2971  1st Qu.:0.018709  1st Qu.:1.6229
17 Median :0.008744 Median :0.3554   Median :0.024809  Median :1.9332
18 Mean   :0.011539 Mean   :0.3786   Mean   :0.032608  Mean   :2.0351
19 3rd Qu.:0.012303 3rd Qu.:0.4495   3rd Qu.:0.035892  3rd Qu.:2.3565
20 Max.   :0.074835 Max.   :0.6600   Max.   :0.255516  Max.   :3.9565
21
22 count
23 Min. : 60.0
24 1st Qu.: 70.0
25 Median : 86.0
26 Mean   :113.5
27 3rd Qu.:121.0
28 Max.   :736.0
29
30 mining info:
31      data ntransactions support confidence
groceries      9835      0.006      0.25

```

summary of quality measures 中的提升度 (lift) 是新引入的一个度量标准. 假设你知道另一类商品已经被购买了提升度就是用来度量一类商品相对于它的一般购买率, 此时被购买的可能性有多大. 它的定义如下所示

提升度 (lift)

$$lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{support(X)} \quad (8.3)$$

此处, $lift(X \rightarrow Y)$ 与 $lift(Y \rightarrow X)$ 相同.

用 `inspect()` 看看规则

```
1 > inspect(groceryrules[1:3])
2   lhs                  rhs                  support      confidence
3 [1] {potted plants} => {whole milk}      0.006914082 0.4000000
4 [2] {pasta}            => {whole milk}      0.006100661 0.4054054
5 [3] {herbs}            => {root vegetables} 0.007015760 0.4312500
6   coverage    lift    count
7 [1] 0.01728521 1.565460 68
8 [2] 0.01504830 1.586614 60
9 [3] 0.01626843 3.956477 69
```

`lhs` 表示规则的前项 (条件项或者左项)(left-hand side, LHS), 表示为了触发规则需要满足的条件; `rhs` 表示规则的后项 (结果项或者右项)(right-hand side, RHS), 表示满足条件后的预期结果.

第一条规则表示如果一个人买了 `potted plants`, 那么它还会买 `whole milk`. 其支持度 0.007, 置信度 0.400, 我们可以确定该规则涵盖了 0.7% 的交易, 而且涉及 `potted plant` 购买的正确率为 40%. 提升度告诉我们假定一个人买了 `potted plants`, 它相对于一般人会购买 `whole milk` 的概率有多大. 因为 25.6% 的人买了 `whole milk`(支持度), 而购买 `potted plants` 的顾客有 40% 买了 `whole milk`(置信度), 所以提升度 $0.4/0.256=1.56$

注意标有 `support` 的列表表示规则的支持度, 不是 `lhs` 或者 `rhs` 的支持度

第一条看起来没啥逻辑, 我们怎么样才能使得这一事实是有意义的呢? 一种常见的做法是获取学习关联规则的结果, 并把它们分成三类:

- 可行的规则
- 平凡的规则
- 令人费解的规则

显然市场购物篮分析的目标就是要找到可行的 (actionable) 规则或者能够明确的有益启示的观测.

平凡的 (trivial) 规则包括那些过于明显以至于不值一提的规则.

如果商品之间的规则过于不明确以至于搞清楚如何使用这些信息采取行动需要更多的研究, 那么这样的规则就是令人费解的 (inexplicable).

下一节我们将采取方法对所学的规则进行排序和分配以提高我们的工作效用.

8.2.5 第五步-提高模型性能

8.2.5.1 对关联规则集合排序

根据市场购物篮分析的目标, 最有用的规则或许是那些具有最高支持度, 置信度和提升度的规则. `arules` 包有 `sort()` 函数可以对规则列表重新排序, 从而那些具有最高或者最低质量度量的规则将会排在第一位.

例如, 使用下面的命令, 根据提升度研究最好的 5 个规则:

```
1 > inspect(sort(groceryrules, by = "lift")[1:5])
2 lhs          rhs          support confidence coverage lift count
3 [1] {herbs}    =>{root vegetables} 0.007015 0.43125 0.01626 3.9564 69
4 [2] {berries}  =>{whipped/sour cream} 0.009049 0.27217 0.03324 3.7968 89
5 [3] {other vegetables,
6 tropical fruit,
7 whole milk}   =>{root vegetables} 0.007015 0.41071 0.01708 3.7680 69
8 [4] {beef,
9 othervegetables}=>{root vegetables} 0.007930 0.40206 0.01972 3.6886 78
10 [5] {other vegetables,
11 tropical fruit} =>{pip fruit}      0.009456 0.26345 0.03589 3.4826 93
```

在默认情况下, 排序的顺序是递减的, 这意味着最大值排在第一位, 为了反转可以添加参数 decreasing=FALSE

8.2.5.2 提取关联规则的子集

假设给定上述规则, 营销团队对于有可能创建一个广告来促销正在处于旺季的 berries 感到非常激动. 调查 berries 是否经常与其他商品一起购买.

subset() 函数提供了一种用来寻找交易, 商品 (项) 或者规则子集的方法. 要用该函数在所有规则中寻找那些包含 berries 的规则:

```
1 > berryrules <- subset(groceryrules, items %in% "berries")
2 > inspect(berryrules)
3 lhs          rhs          support      confidence coverage
4 [1] {berries} => {whipped/sour cream} 0.009049314 0.2721713 0.0332486
5 [2] {berries} => {yogurt}           0.010574479 0.3180428 0.0332486
6 [3] {berries} => {other vegetables} 0.010269446 0.3088685 0.0332486
7 [4] {berries} => {whole milk}        0.011794611 0.3547401 0.0332486
8 lift      count
9 [1] 3.796886 89
10 [2] 2.279848 104
11 [3] 1.596280 101
12 [4] 1.388328 116
```

涉及 berries 的规则有 4 个, 其中有 2 个似乎令人感兴趣, 足以称为可行的规则.

%in% 判断前面一个向量内的元素是否在后面一个向量中, 返回布尔值。

```
1 a <- c(1,3,13,1443,43,43,4,34,3,4,3)
2 b <- c(1,13,11,1313,434,1)
3 a %in% b
4 [1] TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
5
6 a[a %in% b]
7 [1] 1 13
8
9 a[!(a %in% b)]
10 [1] 3 1443 43 43 4 34 3 4 3
```

%in% 意味着至少有一项在定义的列表中可以找到。

%pin% 用于部分匹配, 只要商品名称里有寻找的词, 就认为是正确对象。

%ain% 用于完全匹配, 需要所有列出的项都存在。

8.2.5.3 将关联规则保存到文件或者数据框

若要分享市场购物篮分析结果, 可以 write() 保存到 csv

```
1 > write(groceryrules, file = "groceryrules.csv",
2 +       sep = ",", quote = TRUE, row.names = FALSE)
```

若要转换成 R 中数据框:

```
1 > groceryrules_df <- as(groceryrules, "data.frame")
2 > str(groceryrules_df)
3 'data.frame': 463 obs. of 6 variables:
4 $ rules      : Factor w/ 463 levels "{baking powder} => {other vegetables}",...
5               340 302 207 206 208 341 402 21 139 140 ...
6 $ support    : num  0.00691 0.0061 0.00702 0.00773 0.00773 ...
7 $ confidence: num  0.4 0.405 0.431 0.475 0.475 ...
8 $ coverage   : num  0.0173 0.015 0.0163 0.0163 0.0163 ...
9 $ lift       : num  1.57 1.59 3.96 2.45 1.86 ...
10 $ count      : int  68 60 69 76 76 69 70 67 63 88 ...
```

8.3 总结

下一章我们学习另一种无监督的学习算法, 该算法与关联规则一样, 目的也是探寻数据中的模式, 但其与探寻关联规则的内部特征模式不同, 它更关注找到案例之间的联系。

Chapter 9

寻找数据的分组—k 均值聚类

在聚类时, 标签可能反映了属于同一组内的个体之间的一些潜在的相似模式.

本章介绍了用于处理聚类任务的机器学习方法, 包括找到数据的自然分组. 正如你看到的, 聚类的过程与刚刚描述的观察研究的过程是非常相似的.

你将学习:

- 不同于我们先前研究的发呢类任务的聚类任务方法以及聚类如何定义分组.
- k 均值的基本方法, 它是一种经典的并且容易理解的聚类算法.
- 如何将聚类应用到现实世界的市场细分任务, 例如青少年社交媒体用户市场细分.

9.1 理解聚类

聚类是一种无监督的机器学习任务, 可以自动将数据划分成类 (cluster), 或者具有类似的分组. 因此, 聚类分组不需要提前被告知所划分的组应该是什么样的. 因为我们甚至可能都不知道我们在寻找什么, 所以聚类是用于知识发现而不是预测. 它提供了一种内部发现自然分组的深刻洞察.

聚类的原则是一个组内的记录彼此必须非常相似, 而与该组之外的记录截然不同.

你可能会发现如下一些应用会采用聚类的方法:

- 用客户细分为具有相同的人口地理特征或者相同购买模式的组别, 从而应用于有针对性的营销活动, 或者根据组别来对购买行为进行详细分析.
- 通过识别使用模式不同于已知组别来检测异常行为, 比如侦测未经授权入侵计算机网络的行为.
- 将大量具有相似值的特征划分成较小的具有同质类特征的组中, 从而简化特大数数据集.

聚类非常有用, 无论何时, 各种数据集都可以用较小的多个数据组来表示. 而且, 通过聚类生成的有意义的和可行的数据内部结构降低了数据的复杂性, 并且提供了关于关系模式的深刻见解.

9.1.1 聚类—一种机器学习任务

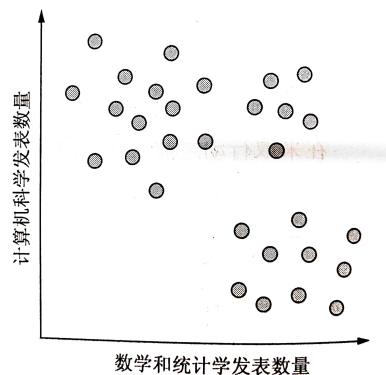
聚类与我们目前为止已经研究过的分类, 数值预测和模式检测任务稍有不同. 在我们研究过的这些案例中, 每一个案例的结果都是一个特征与结果或者特征与其他特征相关的模型. 这些模型可以识别数据内部的模式.

相比之下,聚类创建了新的数据,给无标签案例一个类标签,并完全根据数据的内部关系进行推断.由于这个原因,有时你将会看到聚类称为无监督分类 (unsupervised classification),因为从某种意义上讲,这就是对无标签案例进行分类.

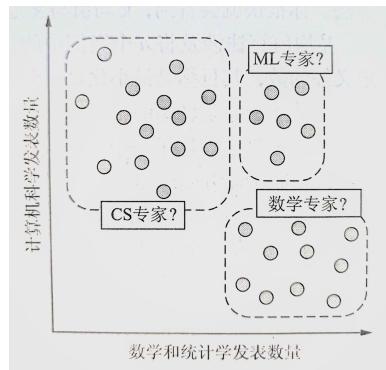
美中不足的是,从无监督分类器获得的分类标签没有内在意义.聚类将告诉你样本中的哪些类别是密切相关的,但应该用一个可行的且有意义的类标签是由你来决定.

实例:

一个会议,给成员分组以安排座位,关于每一位与会者在计算机科学相关杂志上发表论文数量以及在数学或者统计相关杂志上发表论文的数量.



猜测:(设定假定类标签)



聚类算法的过程和这个通过视觉查看散点图所做的有些类似.通过使用一个度量样本相关程度有多紧密的度量,就可以将它们分配到同质的组中.

如果你一开始从无标签的数据入手,那么你就可以使用聚类来创建分类标签然后可以应用有监督学习算法(如决策树)来寻找这些类中最重要的预测指标,这称为半监督学习 (semi-supervised learning)

9.1.2 k 均值聚类算法

k 均值算法可能是最常用的聚类算法.

优点:

- 使用简单的原则来确定可以用非统计术语解释的类
- 具有高度的灵活性, 并且通过简单的调整, 它就可以进行修正以克服它几乎所有的缺点
- 可以将数据划分成有用的类, 他是相当有效且运行良好.

缺点:

- 相对于更多近期的聚类算法, 它不够复杂
- 因为它使用了一个随机的元素, 所以它不能保证找到最佳的类
- 需要一个合理的猜测: 数据有多少个自然类

k 均值和 k 邻近有很多共同点.

k 均值算法涉及将 n 个案例中的每一个案例分配到 k 个类中, 其中 k 是一个提前定义好的数, 其目标是最小化每一个类内不的差异, 最大化类之间的差异.

除非 k 和 n 是极小的, 否则遍及案例所有可能的组合, 计算最优的聚类是不可行的. 取而代之, 该算法使用了一个可以找到局部最优解的启发式过程. 简单来说, 这意味着它从类分配的最初猜测开始, 然后对分配稍加修正以查看改变是否提升了类内部的同质性.

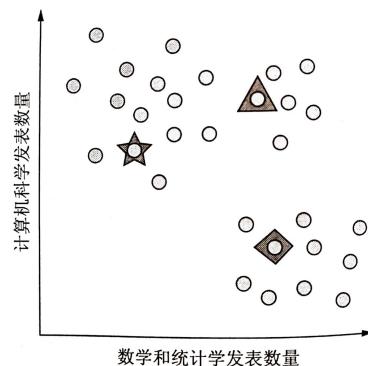
该算法本质包括两个阶段: 首先该算法将案例分配到初始的 k 个类中. 其次, 根据落入当前类的案例调整类的边界来更新分配. 重复更新和分配过程多次, 直到做出的改变不会再提升类的优度. 至此, 该过程停止, 聚类最终确定.

由于 k 均值的启发式性质, 你可能只需对初始条件做出轻微改变, 就能以略有不同的最终结果结束. 如果结果相差很大, 这可能表示存在问题. 例如, 该数据可能不存在自然分组或者已经选定的 k 值很差. 为此, 尝试不止一次的聚类分析来测试研究结果的稳健性或许是一种很好的想法.

9.1.2.1 使用距离来分配和更新类

就像 k 邻近一样, k 均值将特征的值作为一个多维特征空间中的坐标. 对于该会议, 只有两个特征, 因此画一个二维散点图.

k 均值算法首先在特征空间中选择 k 个点作为类的中心. 这些中心是促使剩余的案例落入特征空间中的催化剂 (触发因素). 通常这些点是根据从训练数据集中选择的 k 个随机案例而确定的.

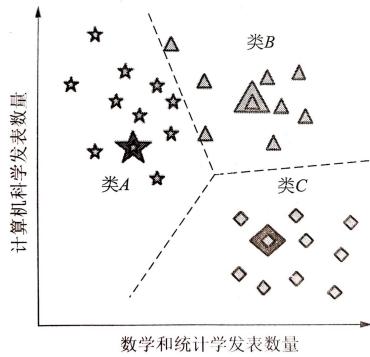


还有一些其他方法可以用来选择初始的类中心. 一种方法是选择发生在特征空间任意地方的随机值, 而不是只在数据的观测值之间进行选择; 另一种方法是完全跳过这一步, 通过将每个案例随机分配一个类中, 该算法可以直接跳过这一阶段, 立即进入更新阶段. 这里每一种方法都会对最终的结果产生一个特定的偏差, 或许你可以使用它来调整结果.

在选择了初始类中心忠厚, 其他的案例将分配到与其最相似或者根据距离函数最相近的类中心. k 邻近我们研究过距离函数. 一般, k 均值使用欧氏距离, 有时也用曼哈顿距离或者闵可夫斯基距离.

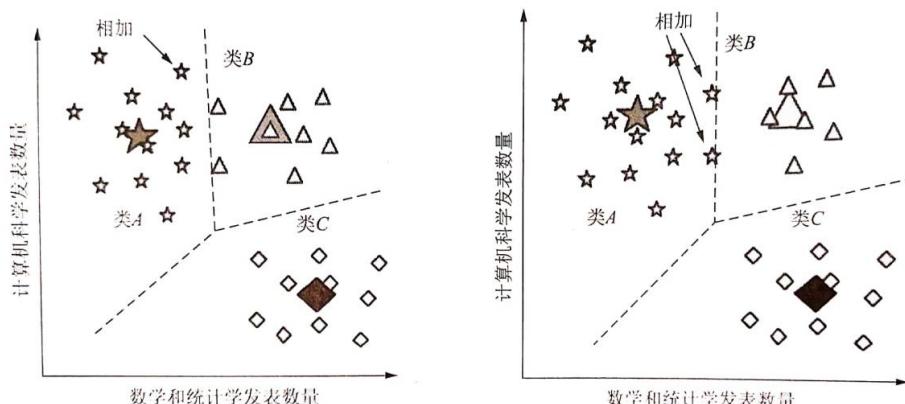
因为使用距离计算, 所以所有的数据都必须是数值型的, 而且所有的值都需要提前标准化到一个标准范围内. 第三章介绍的方法很有用.

如下图所示, 三个类中心将案例划分到标有类 A, 类 B, 类 C 三个区域. 虚线表示由类中心创建的沃洛诺伊图 (voronoi diagram) 的边界. 沃洛诺伊图给出相对于任何其他的类中心更接近于当前的类中心的区域, 3 条类边界汇合的顶点是到 3 个类中心的距离最大的点. 利用这些边界, 我们可以很容易看到由每一个初始的 k 均值种子所确定的区域:

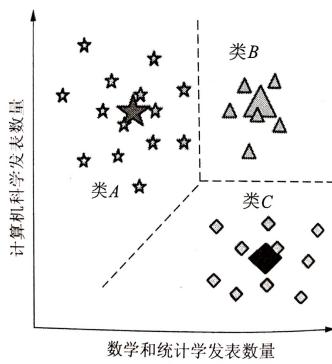


既然初始的分配阶段已经完成, 则 k 均值算法就转到更新阶段. 更新类的第一步涉及将出事的类中心转移到一个新的位置, 称为质心 (centroid), 它可以通过计算分配到当前类中的个点的均值来得到.(左下图)

因为类的边界已经根据重新定位的中心进行了调整, 所以类 A 中可以分配到来自类 B 的额外的案例, 如箭头所示. 由于这种重新分配, 所以 k 均值算法将继续进入下一个更新阶段. 在对类重新计算质心之后, 得到的图如右下所示.



在这个阶段中, 又有两个点从类 B 中重新分配到 A, 因为现在这两个点相对于 B 的质心, 更接近于类 A 的质心. 这样就导致了另一个更新阶段. 如下图:



由于没有点在此阶段被重新分配, 所以 k 均值算法停止. 到目前为止, 聚类分配最终完成.

聚类学习的最终结果可以通过以下两种方式之一来表达. 一方面你可以简单的报告每一个案例的分类情况; 另一方面, 你也可以报告在最后一次更新之后, 聚类质心的坐标. 给定两种表达方法中的任意一种, 你就能通过计算质心或者将每一个案例分配到离它最近的类种来定义类的边界.

9.1.2.2 选择适当的聚类数

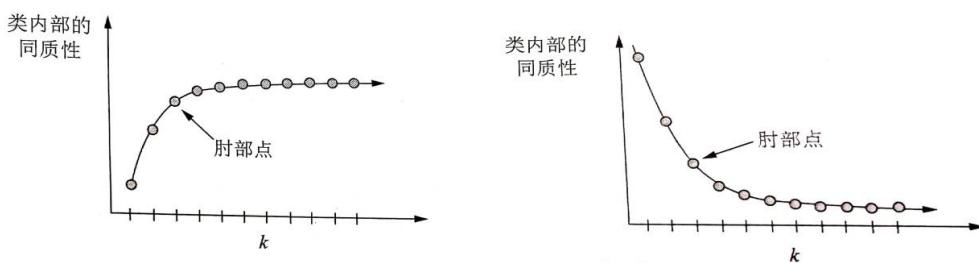
在 k 均值的介绍中, 我们了解到该算法对于随机选择的聚类中心很敏感. 事实上, 如果在前面的例子中, 我们选择了一个关于 3 个初始点的不同组合, 我们可能会发现划分数据的类与我们之前所预期的不一样.

选择类的数目需要一种微妙的平衡, 设定非常大的 k 会提升类的同质性, 但与此同时, 会有过度拟合数据的风险.

理想情况下, 你将有一些关于真是分组的先验知识 (即先验信念), 使用该信息, 你可以开始应用 k 均值算法. 有时候, 聚类数是由业务需求或者分析冬季所决定的.

如果没有先验知识, 一个经验规则建议设置 $k=\sqrt{n/2}$, 其中 n 表示数据集中的案例总数. 然而, 该经验规则可能会导致大型数据集中的聚类数比较庞大. 幸运的是, **肘部法 (elbow method)** 的技术试图度量对于不同的 k 值, 类内部的同质性或者异质性是如何变化的.

如下图, 随着额外的类增加, 类内部的同质性应该是期望上升的; 类似, 一致性也将随着越来越多的类而持续减小. 因为你可以持续地看到改进直到每个案例在一个由他自己构成的类中, 所以我们的目标不是最大化同质性或者最小化异质性, 而是找到一个 k , 使得高于该值之后地收益会发生递减. 这个 k 值称为**肘部点 (elbow point)**.



有许多用来度量类内部同质性和异质性地统计量可以与肘部法一起使用. 然而在实践中, 反复测试大量 k 不总可行. 部分原因是因为对大型数据集进行聚类相当费时, 而对数据进行重复聚类则会更加糟糕. 不管怎样, 要求获得类最优解集地应用是相当罕见的. 在大多数聚类应用中, 基于方便选择一个 k 值就足够了, 而不需要基于严格地性能要求来选择 k .

9.1.3 用 k 均值聚类探寻青少年市场细分

9.1.3.1 第一步—收集数据

使用 30000 名美国高中生地随机案例的数据. 这些数据均匀采用 4 个高中毕业年份 (2006-2009). 在数据收集时他们来自高四高三高二高一. 将爬取的数据前 500 个单词中,36 个单词被选来代表 5 大兴趣类.

9.1.3.2 第二步—探索和准备数据

```
1 > teens<-read.csv("E:/机器学习/ml&r/Machine Learning with R/chapter 9/snsdata.csv")
2 > str(teens)
3 'data.frame': 30000 obs. of 40 variables:
4 $ gradyear : int 2006 2006 2006 2006 2006 2006 2006 2006 2006 2006 ...
5 $ gender   : Factor w/ 2 levels "F","M": 2 1 2 1 NA 1 1 2 1 1 ...
6 $ age      : num 19 18.8 18.3 18.9 19 ...
7 $ friends  : int 7 0 69 0 10 142 72 17 52 39 ...
8 $ basketball : int 0 0 0 0 0 0 0 0 0 0 ...
9 $ football  : int 0 1 1 0 0 0 0 0 0 0 ...
10 $ soccer   : int 0 0 0 0 0 0 0 0 0 0 ...
11 $ softball  : int 0 0 0 0 0 0 0 1 0 0 ...
12 $ volleyball: int 0 0 0 0 0 0 0 0 0 0 ...
13 $ swimming : int 0 0 0 0 0 0 0 0 0 0 ...
14 $ cheerleading: int 0 0 0 0 0 0 0 0 0 0 ...
15 $ baseball : int 0 0 0 0 0 0 0 0 0 0 ...
16 $ tennis   : int 0 0 0 0 0 0 0 0 0 0 ...
17 $ sports   : int 0 0 0 0 0 0 0 0 0 0 ...
18 $ cute     : int 0 1 0 1 0 0 0 0 0 1 ...
19 $ sex      : int 0 0 0 0 1 1 0 2 0 0 ...
20 $ sexy     : int 0 0 0 0 0 0 0 1 0 0 ...
21 $ hot      : int 0 0 0 0 0 0 0 0 0 1 ...
22 $ kissed   : int 0 0 0 0 5 0 0 0 0 0 ...
23 $ dance    : int 1 0 0 0 1 0 0 0 0 0 ...
24 $ band    : int 0 0 2 0 1 0 1 0 0 0 ...
25 $ marching : int 0 0 0 0 0 1 1 0 0 0 ...
26 $ music   : int 0 2 1 0 3 2 0 1 0 1 ...
27 $ rock    : int 0 2 0 1 0 0 0 1 0 1 ...
28 $ god     : int 0 1 0 0 1 0 0 0 0 6 ...
29 $ church  : int 0 0 0 0 0 0 0 0 0 0 ...
30 $ jesus   : int 0 0 0 0 0 0 0 0 0 2 ...
31 $ bible   : int 0 0 0 0 0 0 0 0 0 0 ...
32 $ hair    : int 0 6 0 0 1 0 0 0 0 1 ...
33 $ dress   : int 0 4 0 0 0 1 0 0 0 0 ...
```

```

34 $ blonde      : int  0 0 0 0 0 0 0 0 0 0 ...
35 $ mall        : int  0 1 0 0 0 0 2 0 0 0 ...
36 $ shopping    : int  0 0 0 0 2 1 0 0 0 1 ...
37 $ clothes     : int  0 0 0 0 0 0 0 0 0 0 ...
38 $ hollister   : int  0 0 0 0 0 0 2 0 0 0 ...
39 $ abercrombie : int  0 0 0 0 0 0 0 0 0 0 ...
40 $ die         : int  0 0 0 0 0 0 0 0 0 0 ...
41 $ death       : int  0 0 1 0 0 0 0 0 0 0 ...
42 $ drunk       : int  0 0 0 0 1 1 0 0 0 0 ...
43 $ drugs        : int  0 0 0 0 1 0 0 0 0 0 ...

```

数据包含 30000 名青少年, 其中 4 个变量表示个人特征, 36 个变量表示兴趣.

其中性别有缺失值:

```

1 > table(teens$gender)
  F      M
2 22054  5222

```

table() 排除了 NA, 我们需要添加一个额外的参数:

```

1 > table(teens$gender, useNA = "ifany")
  F      M  <NA>
2 22054  5222  2724

```

2724 条, 9% 缺失性别数据. 其实 age 也有缺失值, 我们用 summary 命令看看缺失值的数量:

```

1 > summary(teens$age)
2 Min.  1st Qu.  Median      Mean  3rd Qu.      Max.      NA's
3 3.086   16.312   17.287   17.994   18.259   106.927      5086

```

对于 age, 5086 即 17% 有缺失值. 而且最大最小值也是不可信的, 三岁和 106 岁.

高中生合理的年龄范围是在至少 13 岁还没有超过 20 岁的学生, 任何落在此范围之外的年龄应该作缺失值处理. 为了重新对年龄编码, 我们可以使用 ifelse()

```

1 > teens$age <- ifelse(teens$age >= 13 & teens$age < 20,
2 +                     teens$age, NA)
3 > summary(teens$age)
4 Min.  1st Qu.  Median      Mean  3rd Qu.      Max.      NA's
5 13.03   16.30   17.27   17.25   18.22   20.00      5523

```

好了现在得到了更大的缺失数据问题.

9.1.3.3 数据准备—缺失值的虚拟编码

一种简单的处理缺失值的办法就是排除具有缺失值的记录. 但是要慎重. 该方法的问题在于, 即使缺失值不是太多, 你就非常快速地大部分数据.

对于分类数据像性别这样的变量, 另一种解决办法就是将缺失值作为一个单独的类别. 与此同时, 我们还应该利用虚拟编码, 这在第三章有过深入的介绍, 就是将名义性别变量转化为可以用于距离计算的数值型变量.

除了有一个水平值被拿出来作为参照组以外, 虚拟编码涉及为名义特征中的每个水平值单独创建一个取值为 1 或 0 的二元虚拟变量. 有一个水平值被排除在外, 因为它可以通过其他类水平值来推断.

```
1 > teens$female <- ifelse(teens$gender == "F" &
2 +                               is.na(teens$gender), 1, 0)
3 > teens$no_gender <- ifelse(is.na(teens$gender), 1, 0)
```

证实一下我们的工作:

```
1 > table(teens$gender, useNA = "ifany")
2
3 F      M  <NA>
4 22054  5222  2724
5 > table(teens$female, useNA = "ifany")
6
7 0      1
8 7946  22054#女性数量
9 > table(teens$no_gender, useNA = "ifany")
10
11 0      1
12 27276  2724#无性别数量
```

9.1.3.4 数据准备—插补缺失值

接下来小虎关于 age 的 5523 个缺失值. 我们使用插补法 (imputation) 的策略, 依据可能的真实值的猜测来填补缺失值.

你能想到用毕业年份来估计年龄吗? 在一个毕业生队列中, 大部分学生都是在同一年出生的. 如果我们能找出每一个队列中具有代表性的年龄, 那么我们将对那年毕业的学生的年龄有一个相当合理的估计.

我们用 mean 来找出具有代表性值:

```
1 > mean(teens$age)
2 [1] NA
```

包含缺失值, 均值无法定义. 我们要去除缺失值:

```
1 > mean(teens$age, na.rm = TRUE)
2 [1] 17.25243
```

这表明在我们数据中学生平均年龄 17. 我们真正需要的是每一个毕业年份的年龄均值. 用 aggregate() 可以为数据的子组计算统计量.

```
1 > aggregate(data = teens, age ~ gradyear, mean, na.rm = TRUE)
```

```

2 gradyear      age
3 1 2006 18.65586
4 2 2007 17.70617
5 3 2008 16.76770
6 4 2009 15.81957

```

这个数据变化证明了我们的数据是合理的.

aggregate() 的输出在一个数据框中, 它是人们可以读取的, 但是还需要额外的工作来把它合并到我们的原始数据. 作为一种替代方法, 可以使用 ave(), 该函数返回一个具有重复的组均值的向量, 使得结果在长度上等于原始向量的长度.

```

1 > ave_age <- ave(teens$age, teens$gradyear,
2 +                 FUN = function(x) mean(x, na.rm = TRUE))

```

为了将这些均值插补到缺失值中, 需要再一次使用 ifelse(), 仅当年龄缺失才调用 ave_age 的值:

```

1 > teens$age <- ifelse(is.na(teens$age), ave_age, teens$age)

```

用 summary 看看, 无缺失值了.

```

1 > summary(teens$age)
2   Min. 1st Qu. Median      Mean 3rd Qu.      Max.
3 13.03    16.28   17.24    17.24    18.21    20.00

```

9.1.4 第三步—基于数据训练模型

用 stats 包的 k 均值实现, 该包包含在 R 的默认安装中.

聚类语法
应用 stats 添加包中的函数 kmeans()
建立模型:
<pre>myclusters <- kmeans(mydata, k) • mydata: 是包含需要聚类的实例的一个矩阵或者数据框 • k: 给定需要的类的个数</pre>
该函数返回一个含有聚类结果的聚类对象。
检验聚类结果:
<ul style="list-style-type: none"> • myclusters\$cluster 是 kmeans() 函数所给出的类成员向量 • myclusters\$centers 是含有每个类组合和每一个特征的均值的一个矩阵 • myclusters\$size 给出每一个类中实例的个数
例子:
<pre>teen_clusters <- kmeans(teens, 5) teens\$cluster_id <- teen_clusters\$cluster</pre>

kmeans() 函数需要一个包含数值型数据的数据框和一个用来指类的期望数目的参数. 建模的过程很简单, 麻烦的是选择数据和类的正确组合是一门艺术, 有时候会陷入大量的试验和错误中.

为了开始聚类分析, 我们只考虑 36 个特征. 为方便, 我们创建一个只包含这些特征的数据框:

```
1 > interests <- teens[5:40]
```

在使用距离计算分析之前, 应该将数据特征标准化或者 z-score 标准化. z-score 标准化就是重新调整特征使得它们的均值为 0, 方差为 1, 这种转化某种意义上该变量特征的解释含义, 但是或许在这里会比较有用.

为了将 z-score 标准化应用到数据框 interests, 使用带有 lapply() 的 scale() 函数:

```
1 > interests_z <- as.data.frame(lapply(interests, scale))
```

因为 lapply() 返回一个矩阵, 所以必须要使用 as.data.frame()

我们最后的决定涉及确定使用多少类来细分数据. 用太多类, 可能会发现它们过于具体而没什么用; 选择过少的类可能会导致异质分组. 使用不同的 k 进行试验应该会感到轻松.

我们选 k=5: 将 teens 分为 5 类.

```
1 > teen_clusters <- kmeans(interests_z, 5)
```

9.1.5 第四步-评估模型性能

评估聚类的结果有一定的主观性. 因为本次分析的目标是为了确定具有相似兴趣的青少年的分类以达到营销的目的. 所以我们很大程度上定性度量我们的成功.

评估一个类是否有用最基本方法之一是检查落在每一组中的案例数, 过度过少的话这些类都不太有用.

```
1 > teen_clusters$size
2 [1] 20539 2518 5089 986 868
```

虽然有的类有点大, 但还好没有类只有一个人.

为了更深入了解类, 用 teen_clusters\$centers 分量来查看聚类质心的坐标, 前六个特征的如下:

```
1 > teen_clusters$centers
  basketball   football   soccer   softball   volleyball   swimming
 2 -0.18470350 -0.18843994 -0.0803871 -0.13446381 -0.13077025 -0.1080610
 3  1.41139990  1.25990865  0.4715473  1.17959841  1.10315092  0.1056866
 4 -0.04357814  0.02898553  0.0512752 -0.07102037 -0.06337936  0.2926831
 5  0.34332617  0.36267270  0.1142789  0.13033068  0.08696035  0.2598940
 6  0.14167017  0.22213753  0.1037974  0.02815663  0.16699415  0.2392009
```

输出的行号指的是类, 输出的数值表示位于该列顶部的兴趣变量的均值. 由于进行了 z-score 标准化, 所以负值表示低于所有学生的总体均值, 正值就是高于.

虽然只列出了八个, 但是我们已经可以推断出这些类的一些特征. 在所有列出来的体育运动中, 除了啦啦队变量, 类 4 在运动项目上数值显著大于均值, 表明该组可能包含运动员.

通过这种方式研究这些类, 可以构建一个表来列出每组的主要兴趣项目. 有趣的是, 类 5 被区分开是由于它的特征不显著的事实, 在每一个兴趣活动中, 它的特征的兴趣水平都低于平均值, 但从成员人

数方面来说, 他又是最大的单一类. 一种潜在的解释是, 这些用户组建了个人资料但是没有发布任何兴趣爱好.

类 1 (N = 3376)	类 (2N = 601)	类 (3N = 1036)	类 (4N = 3279)	类 (5N = 21 708)
游泳 (swimming)	乐队 (band)	运动 ()	篮球 (basketball)	???
啦啦对 (cheerleading)	游行 (marching)	性 (sex)	橄榄球 (football)	
聪明 (cute)	音乐 (music)	性感 (sexy)	足球 (soccer)	
性感 (sexy)	摇滚乐 (rock)	受欢迎 (hot)	垒球 (softball)	
受欢迎 (hot)		亲吻 (kissed)	排球 (volleyball)	
跳舞 (dance)		跳舞 (dance)	运动 (sports)	
打扮 (dress)		音乐 (music)	上帝 (god)	
头发 (hair)		乐队 (band)	教堂 (church)	
商场 (mall)		筛子 (die)	基督 (Jesus)	
Hollister 牌		死亡 (death)	圣经 (bible)	
Abercrombie 牌		醉酒 (drunk)		
购物 (shopping)		吸毒 (drugs)		
服装 (clothes)				
公主 (Princess)	聪明人 (Brain)	罪犯 (Criminal)	运动员 (Athlete)	没有特征 (basket case)

9.1.6 第五步—提高模型性能

因为聚类创造了新的信息, 所以一聚类算法的性能至少在某种程度上取决于这些类本身的质量以及如何处理这些信息.

我们已经证明了这 5 个类提供了关于青少年兴趣的有用的并且新颖的见解. 现在集中精力将这些见解转化为行动.

首先, 将把这些类应用回完整的数据集.

```
1 > teens$cluster <- teen_clusters$cluster
```

根据这个信息, 可以看每个人属于哪一类. 我们看看前五人:

```
1 > teens[1:5, c("cluster", "gender", "age", "friends")]
2 cluster gender     age friends
3 1       1      M 18.982      7
4 2       3      F 18.801      0
5 3       1      M 18.335     69
6 4       1      F 18.875      0
7 5       4     <NA> 18.995     10
```

用 aggregate() 从整体上查看这些类的人口统计特征. 根据聚类, 类之间每一类的年龄均值变化不大, 尽管我们认为不同年龄不一定在兴趣上有系统性的差异:

```
1 > aggregate(data = teens, age ~ cluster, mean)
2 cluster     age
3 1       1 17.30033
4 2       2 17.03740
5 3       3 17.16748
6 4       4 17.12691
```

另外, 根据聚类, 每一类的女性比例会由一些显著的差异.

```

1 > aggregate(data = teens, female ~ cluster, mean)
2   cluster   female
3   1       1 0.7007644
4   2       2 0.6926132
5   3       3 0.8650029
6   4       4 0.8002028
7   5       5 0.8364055

```

朋友:

```

1 > aggregate(data = teens, friends ~ cluster, mean)
2   cluster   friends
3   1       1 27.54384
4   2       2 35.47776
5   3       3 36.21045
6   4       4 30.65619
7   5       5 41.27419

```

组内成员身份 (年龄), 性别, 朋友数量, 之间的关系表明, 这些类是非常有用的预测因子. 以这种方式来验证这些类的预测能力, 使得将这些类在推销给营销团队时变得更加容易, 并且最终提高算法的性能.

9.2 总结

本章只是介绍了聚类的基本原理. 作为一种非常成熟的机器学习方法,k 均值算法有无数种变体, 还有很多其他可以给任务带来独特见解与启发的算法. 基于在这里学到的知识将能够理解和运用其他聚类方法去解决新的问题.

Chapter 10

模型性能评价

类似于编写考试问题的过程可以用来评估机器学习算法. 由于不同的算法具有不同的优缺点, 所以在评价算法对未来数据的性能时需要使用测试来展示不同算法之间的差异.

本章将提供一些用来评价机器学习算法的信息, 包括:

- 为什么预测准确度不足以度量性能, 以及替代度量行呢的方法.
- 用来确保能度量方法可以合理地反映模型对于位置数据地预测能力的办法.
- 如何使用 R 将这些更有用的度量应用到前面章节中学习的预测模型.

10.1 度量分类方法的性能

在前面的章节中, 我们使用的是准确度来度量我们模型的性能. 准确度就是将正确预测的部分除以预测的总数得到的数值.

但是, 假设如果给是否携带某一先天缺陷, 该先天缺陷每 100000 新生儿有 10 例. 如果不考虑任何情况全部预测为没有缺陷, 分类器的准确度也是 99.99%. 该分类器即使对绝大多数的数据都能预测正确, 但是对于其最初打算识别新生缺陷的目的却不管用.

对分类新能最好的度量方式就是要看其是否能成功地实现预期目标. 因此拥有能够度量实用性而不是原始准确度地模型性能评价是至关重要的.

10.1.1 在 R 中处理分类预测数据

有三种主要的数据类型可以用来评价分类器:

- 真实的分类值
- 预测的分类值
- 预测的估计概率

在之前的章节中, 我们只使用了前两种向量来检查分类方法的预报能力. 研究内部预测概率对于评估模型性能非常有用, 它同时也是第三种形式评估数据的来源. 如果两个模型在预测时发生了相同数目的错误, 但是其中一个能够更准确地估计它的不确定性, 那么这个模型就是更智能的模型.

理想的分类器在做出正确预测时能够非常自信, 但是在面对拿不准的情况时能够非常谨慎. 在信心和谨慎之间的平衡是模型评价中的一个关键部分.

不幸的是,要得到内部预测概率是一件棘手的事情,因为面对不同的分类器,计算方法是不同的.一般来说,分类器的 predict() 函数可以指定希望的预测类型.如果想要得到单一的预测类别(例如垃圾信息或者有用的信息),可以设定为"class"类型.如果要得到预测的概率,可以设定 prob,posterior,raw 或者 probability 等类型.

例如,如果要输出朴素贝叶斯分类的预测概率,可以参考 chp4 的描述,在预测函数中加上参数 type="raw".

如: predict_prob<-predict(model,test_data,type="raw")

需要记住的是,大部分情况下,predict() 函数将返回对结果不同水平的预测结果.例如,在结果是"yes/no"的二值模型中, predict_prob 将是一个矩阵或者数据框.

在构建测试数据集时要当心,从而确保对于感兴趣的类别使用了正确的概率.为了避免混淆,在二值结果结果的情况下,建议在以上两个输出向量中只保留一个.

为了举例,我们使用第四章开发的垃圾短信分类模型判断出的估计为垃圾短信的概率以真实类别和预测类别组成的数据框进行介绍.

```
1 > sms_results<-read.csv("E:/6. 机器学习 /ml&r/Machine Learning with R/
2 chapter 10/sms_results.csv")
3 > head(sms_results)
4   actual_type predict_type   prob_spam
5 1      ham        ham 2.560231e-07
6 2      ham        ham 1.309835e-04
7 3      ham        ham 8.089713e-05
8 4      ham        ham 1.396505e-04
9 5      spam       spam 1.000000e+00
10 6      ham        ham 3.504181e-03
```

第三列表示该短信是垃圾信息的概率.当预测类型为 ham 有用信息时,第三列概率非常接近零,为垃圾信息时概率为 1,表示模型有百分百的信心确定该短信为垃圾短信.

但是当预测值和真实值不同的时候呢?我们用 subset() 看看:

```
1 > head(subset(sms_results, actual_type != predict_type))
2   actual_type predict_type   prob_spam
3 53      spam        ham 0.0006796225
4 59      spam        ham 0.1333961018
5 73      spam        ham 0.3582665350
6 76      spam        ham 0.1224625535
7 81      spam        ham 0.0224863219
8 184     spam        ham 0.0320059616
```

可以发现这些概率没有那么极端.尽管有这些错误,模型是否仍然很有用?我们可以通过对测试数据应用各种误差度量的方法来回答这个问题.

10.1.2 深入讨论混淆矩阵

混淆矩阵 (confusion matrix) 是一张二维表, 它按照预测是否匹配数据的真实值来对预测值进行分类. 混淆矩阵可以用于预测多个类别的模型.

		两个类的情况		三个类的情况		
		预测类别		预测类别		
		A	B	A	B	C
实际类别	A	○	✗	○	✗	✗
	B	✗	○	✗	✗	○

对于分类模型的性能度量基于表的对角线和非对角线上预测值的个数. 最常见的模型性能度量方式主要考虑模型所在的所有分类中识别出某个分类的能力. 我们感兴趣的类别称为阳性 (positive), 其他所有类别称为阴性 (negative).

对于阳性或者阴性这样的术语并没有隐含任何的价值判断 (好或者坏), 同样也没有暗示出现或者不出现.

- 真阳性 (true positive, TP): 正确的分类为感兴趣的类别.
- 真阴性 (true negative, TN): 正确的分类为不感兴趣的类别.
- 假阴性 (false positive, FP): 错误的分类为感兴趣的类别.
- 假阳性 (false negative, FN): 错误的分类为不感兴趣的类别.

前面提到的先天缺陷分类器, 我们看看他的混淆矩阵图:

		预测先天缺陷	
		否	是
实际先天缺陷	否	TN	FP
	是	FN	TP

真阴性 假阳性
假阴性 真阳性

10.1.3 使用混淆矩阵度量性能

使用 2×2 的混淆矩阵, 可以用公式来表示预测准确度 (accuracy):

$$\text{准确度} = \frac{TP + TN}{TP + TN + FP + FN} \quad (10.1)$$

TP,TN 等表示模型的预测值落入这些类别中的次数.

错误率 (error rate):

$$\text{错误率} = \frac{FP + FN}{TP + TN + FP + FN} = 1 - \text{准确度} \quad (10.2)$$

10.1.3.1 得到混淆矩阵的方法

1. table() 函数

```
1 > table(sms_results$actual_type, sms_results$predict_type)
2
3     ham  spam
4 ham   1202     5
5 spam    29   154
6
7 # 使用公式界面
8 > xtabs(~ actual_type + predict_type, sms_results)
9
10    predict_type
11    actual_type      ham  spam
12          ham   1202     5
13          spam    29   154
```

2. 更详细的混淆矩阵,gmodels 包中的 CrossTable(). 我们用过很多次了.

```
1 > library(gmodels)
2 > CrossTable(sms_results$actual_type, sms_results$predict_type)
3
4 Cell Contents
5 |-----|
6 |           N |
7 | Chi-square contribution |
8 |           N / Row Total |
9 |           N / Col Total |
10 |          N / Table Total |
11 |-----|
12
13 Total Observations in Table:  1390
14
15 | sms_results$predict_type
```

16	sms_results\$actual_type	ham	spam	Row Total
17	-----	-----	-----	-----
18	ham	1202	5	1207
19		16.565	128.248	
20		0.996	0.004	0.868
21		0.976	0.031	
22		0.865	0.004	
23	-----	-----	-----	-----
24	spam	29	154	183
25		109.256	845.876	
26		0.158	0.842	0.132
27		0.024	0.969	
28		0.021	0.111	
29	-----	-----	-----	-----
30	Column Total	1231	159	1390
31		0.886	0.114	
32	-----	-----	-----	-----

准确度和错误率直接使用以上数据就可以计算. 在此略过.

虽然这些计算过程看起来非常简单, 但是通过联系可以更好地理解混淆矩阵中每个元素相对于其他元素的含义. 在下一节中, 我们将看到这些相同部分如何组成不同的方法, 提供另外大的度量性能的方式.

10.1.4 准确度之外的其他性能评价指标

分类和回归训练包 caret 包含了一些函数, 可以用来计算很多这样性能度量指标. 该包提供了大量的对机器学习模型和数据进行准备, 训练, 评估以及可视化工具.

caret 包给出了另一个创建混淆矩阵的函数, 语法与 table() 近似, 但是需要指定阳性的输出.

```

1 > library(lattice)
2 > library(ggplot2)
3 > library(caret)
4 > confusionMatrix(sms_results$predict_type, sms_results$actual_type,
5   positive = "spam")
6 Confusion Matrix and Statistics
7
8 Reference
9 Prediction  ham  spam
10 ham  1202    29
11 spam     5   154
12
13 Accuracy : 0.9755
14 95% CI : (0.966, 0.983)

```

```

15 No Information Rate : 0.8683
16 P-Value [Acc > NIR] : < 2.2e-16
17
18 Kappa : 0.8867          #Kappa 统计量
19
20 Mcnemar's Test P-Value : 7.998e-05
21
22 Sensitivity : 0.8415
23 Specificity : 0.9959
24 Pos Pred Value : 0.9686
25 Neg Pred Value : 0.9764
26 Prevalence : 0.1317
27 Detection Rate : 0.1108
28 Detection Prevalence : 0.1144
29 Balanced Accuracy : 0.9187
30
31 'Positive' Class : spam

```

10.1.4.1 Kappa 统计量

通过解释完全因为巧合而预测正确的概率, **Kappa 统计量** (在前面的输出结果中标记为 Kappa) 对准确度进行了调整. Kappa 最大范围是 1, 说明在预测值和真实值之间是完全一致的, 这样的情况非常少见. 小于 1 的值表示不完全一致.

根据模型使用的目的, 对 Kappa 统计量的解释会有所不同. 但是一般的解释如下所示:

- 很差的一致性: 小于 0.2
- 尚可的一致性: 0.2~0.4
- 中等的一致性: 0.4~0.6
- 不错的一致性: 0.6~0.8
- 很好的一致性: 0.8~1

但是有个问题很重要, 这些类别是主观性的. 如果预测某人最喜欢的冰淇淋口味, 那么“不错的一致性”已经足够; 如果目标是让航天飞机在月球表面安全着陆, 那么“和那后的一致性”也有可能还不够.

下面是计算 Kappa 统计量的公式, Pr 是指分类器和真实值之间的真实一致性 (a) 和期望一致性 (e) 的比例:

$$k = \frac{Pr(a) - Pr(e)}{1 - Pr(e)} \quad (10.3)$$

该比例可以非常容易地通过混淆矩阵计算得到: 在 `CrossTable()` 中, 每格最底部的值表示落入这个格子的实例占所有实例的比例. 因此要计算的一致性 $Pr(a)$, 只要简单地将预测值与真实值一样的格子的比例加起来.

```

1 > pr_a <- 0.865 + 0.111
2 > pr_a
3 [1] 0.976

```

在这个分类器中, 观测值和预测值有 97.6% 的情况是一致的, 这个值和准确度是一样的. Kappa 使用期望的一致性 $Pr(e)$ 对准确度进行调整. $Pr(e)$ 是完全偶然性导致的预测值和实际值相同的概率, 当然需要遵循两者都是在观察到的比例下随机抽样这一假设.

为了找到这些观测比例, 我们知道同时选择 ham(有用讯息) 的概率:

$$Pr(actual_type \text{ is } ham) * Pr(predicted_type \text{ is } ham) \quad (10.4)$$

同时选择 spam(垃圾信息) 的概率:

$$Pr(actual_type \text{ is } spam) * Pr(predicted_type \text{ is } spam) \quad (10.5)$$

$Pr(e)$ 可以通过将”同时选择 ham(有用讯息) 的概率”和”同时选择 spam(垃圾信息) 的概率”相加得到:

```

1 > pr_e <- 0.868 * 0.886 + 0.132 * 0.114
2 > pr_e
3 [1] 0.784096
4
5 > k <- (pr_a - pr_e) / (1 - pr_e)
6 > k
7 [1] 0.8888395

```

Kappa 的值约为 0.89, 这与之前 `confusionMatrix()` 输出的是一样的 (细微不同是因为舍入误差). 利用之前推荐的解释, 我们可以认为该分类器在预测值和真实值中有着很好的一致性.

R 中有很多自动计算 Kappa 值的函数.

vcd 包中的 Kappa

可视化分类数据包 (visualizing categorical data, vcd) 中的 Kappa 函数使用预测值和实际值的混淆矩阵进行计算.

```

1 > library(grid)
2 > library(vcd)
3 > Kappa(table(sms_results$actual_type, sms_results$predict_type))
4 value      ASE      z Pr(>|z|)
5 Unweighted 0.8867 0.01909 46.45      0
6 Weighted   0.8867 0.01909 46.45      0

```

我们主要关注不加权的 Kappa 值, 0.89 符合我们的期望.

加权 Kappa 值主要用于存在不同尺度一致性的情况. 例如使用冷, 温和热的度量方式, 温和热之间的一致性要强于温和冷之间的一致性.

对于二分类, 如有用信息和无用信息, 加权不加权都一样.

irr 包中的 kappa2

评定者间可信度 (inter-rater reliability, irr) 包中的 kappa2() 函数可以直接使用数据框中的预测值向量和实际分类向量来计算 Kappa

```
1 > library(lpSolve)
2 > library(irr)
3 > kappa2(sms_results[1:2])
Cohen's Kappa for 2 Raters (Weights: unweighted)

5

6 Subjects = 1390
7 Raters = 2
8 Kappa = 0.887
9
10 z = 33.2
11 p-value = 0
```

这两种方法都 ok.

注意不要使用内置函数 kappa(), 和 Kappa 统计量没关系.

10.1.4.2 灵敏度和特异性

分类经常要在做决策时过于保守和过于激进之间权衡. 如筛选机制为宁可杀错一百不可放过一个, 就是太激进; 有的又太保守都留着. 这种权衡可以由灵敏度和特异性这对度量方式实现.

模型的**灵敏度** (也称为真阳性率) 度量了阳性样本被正确分类的比例.

$$\text{灵敏度} = \frac{TP}{TP + FN} \quad (10.6)$$

模型的**特异性** (也称真阴性率) 度量了阴性样本被正确分类的比例.

$$\text{特异性} = \frac{TN}{FP + TN} \quad (10.7)$$

给定混淆矩阵, 可以很轻易的计算它们:

```
1 > sens <- 154 / (154 + 29)
2 > sens
3 [1] 0.8415301
4 >
5 > spec <- 1202 / (1202 + 5)
6 > spec
7 [1] 0.9958575
```

caret 包提供了计算灵敏度和特异性的函数, 输入预测值向量和实际分类值即可.

```
1 > library(lattice)
2 > library(ggplot2)
3 > library(caret)
```

```

4 > sensitivity(sms_results$predict_type, sms_results$actual_type,
5   positive = "spam")
[1] 0.8415301
7 > specificity(sms_results$predict_type, sms_results$actual_type,
8   negative = "ham")
[1] 0.9958575

```

灵敏度和特异性的取值范围都是 0~1, 接近 1 的值更令人满意. 当然, 在两者之间找到一个合适的平衡点是很重要的.

使用灵敏度和特异性提供工具来考虑这种权衡. 典型做法是, 对模型进行调整或者使用不同的模型, 直到能通过灵敏度和特异性的阈值为止.

10.1.4.3 精确度和回溯精确度

与灵敏度和特异性紧密相关的是另一个性能度量指标, 同样与分类时的折中方案有关, 他们是预测精确度和回溯精确度. 最开始它们用于信息检索领域, 目的是提供对于模型结果的有趣和有关程度的描述, 或者说预测是否会因为无意义的噪声而减弱.

精确度 (也称为阳性预测值) 定义为真阳性在所有预测为阳性案例中的比例, 换句话说, 当一个模型预测阳性类别时, 总是正确的吗? 一个精确的模型只有在类别非常像阳性时才会预测为阳性. 这是非常可靠的.

$$\text{精确度} = \frac{TP}{TP + FP} \quad (10.8)$$

另一方面, 回溯精确度是关于结果完备性的度量. 和灵敏度一样, 只是解释不一样. 回溯精确度高的模型可以捕捉大量的阳性样本.

$$\text{回溯精确度} = \frac{TP}{TP + FN} = \text{灵敏度} \quad (10.9)$$

精确度和回溯精确度同样可以根据混淆矩阵来计算. caret 包也可以计算这两个值, 只要输入预测类别和真实类别的向量.

```

1 > library(caret)
2 载入需要的程辑包: lattice
3 载入需要的程辑包: ggplot2
4 > posPredValue(sms_results$predict_type, sms_results$actual_type,
5   positive = "spam")
[1] 0.9685535      # 精确度
7 > sensitivity(sms_results$predict_type, sms_results$actual_type,
8   positive = "spam")          # 回溯精确度 / 灵敏度
[1] 0.8415301

```

与灵敏度和特异性之间固有的权衡类似, 对于大多数问题, 很难建立一个同时有很高精确度和回溯精确度的模型. 高精确度只要目标易于分类, 高回溯精确度只要广泛撒网, 这意味着在预测阳性时过于激进.

相比之下, 同时具有高精确度和高回溯精确度非常具有挑战性. 因此测试各种模型是非常重要的.

10.1.4.4 F 度量

将精确度和回溯精确度合并成一个单一值得模型性能度量方式是 **F 度量** (有时也称为 F1 记分或者 F 记分).

F 度量使用调和平均值来正确精确度和回溯精确度. 因为预测精确度和回溯精确度都是 0~1 之间的比例, 所以使用调和平均值而不是更常用得算术平均值.

$$F\text{ 度量} = \frac{2 \times \text{精度} \times \text{回溯精确度}}{\text{回溯精确度} + \text{精度}} = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (10.10)$$

因为 F 度量将模型的性能指标变成了一个单一值, 所以它提供了一种便利的方式来比较多个模型的好坏. 不过这需要假设精确度和回溯精确度具有相同的权重, 然而这个假设并不总是正确的. 不同权重也 ok, 但是选择权数挺棘手的. 更好的实践方式是将诸如 F 度量之类的度量方式与其他更全局化考虑模型的优势和不足的方法联合起来, 见下.

10.1.5 性能权衡的可视化

可视化方法经常能帮助理解机器学习算法的性能在不同情况下是如何不同. 与成对的统计量不同, 可视化可以考察度量如何在大范围的值之间变化. 它们还提供了可以在单个图形中同时比较多个分类器的方法.

ROCR 包提供了一套易于使用的函数用于可视化分类模型性能的统计量. 使用 ROCR 创建可视化图形, 需要两个数据向量. 第一个必须包含预测的类别值, 第二个必须包含阳性类别的估计概率. 这些用来创建一个预测对象, 它可以被 ROCR 的绘图函数检测到.

短信分类器的预测对象可以用分类器的估计垃圾信息概率 (prob_spam) 和实际类别标签 (actual_type) 来生成. 可以使用 prediction() 函数来实现. ROCR 包提供了 performance() 函数来计算所预测对象的性能度量值, 例如前面的代码示例中使用过的 pred. 结果性能对象可以通过 plot() 函数进行可视化.

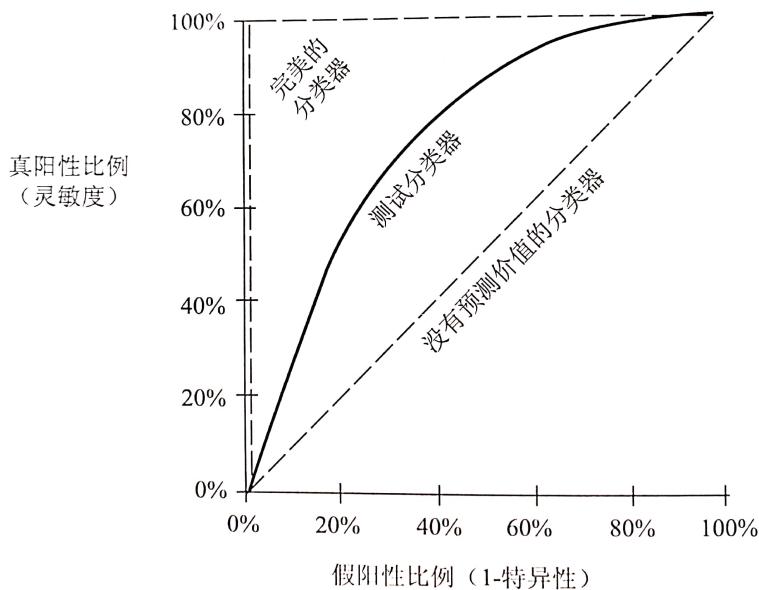
```
1 > library(ROCR)
2 > pred <- prediction(predictions = sms_results$prob_spam,
3 +                         labels = sms_results$actual_type)
```

10.1.5.1 ROC 曲线

ROC(receiver operating characteristic, 受试者工作特征) 曲线常常用来检查在找出真阳性和避免假阳性之间的权衡.

统计图形中的曲线, 纵轴表示真阳性的比例, 横轴表示假阳性的比例. 因为这两个值分别等于灵敏度和 1-特异性, 所以该图形也称为灵敏度/特异性图.

ROC 曲线上的点表示不同假阳性阈值上的真阳性的比例. 绘制曲线时, 分类器的预测值通过模型对阳性类别的估计概率排序, 最大值在最前面. 从原点开始, 每个预测值对应的真阳性和假阳性将导致曲线沿垂直方向 (正确的预测) 移动或者沿水平方向 (错误的预测) 移动.



为了说明这个概念，我们在上图中比较了 3 个假设的分类器。第一个是图中从左下角到右上角的直线，代表没有预测价值的分类器。这种分类器发现真阳性和假阳性的比率完全相同，这意味着该分类器无法识别两者之间的差别。这是评价其他分类器的基准线。ROC 曲线如果比较靠近这条线，则说明模型不是很有用。类似地，完美分类器拥有一条穿过了 100% 真阳性和 0% 假阳性点的曲线。它在不正确地分出任何阴性的结果之前已经正确地识别了所有的真阳性样本。大部分真实的分类器比较类似于测试分类器，它位于完美分类器和没有预测价值的（无用）分类器之间的区域

离完美分类器越接近说明能够越好地识别阳性值。可以使用 ROC 曲线下面积 (Area Under the ROC, AUC) 这个统计量来度量。与字面意思一样，AUC 将 ROC 图看成是 2 维正方形，然后测量 ROC 曲线下的面积。AUC 的值从 0.5 (无预测值的分类器) 到 1.0 (完美分类器)。通常使用类似于学校字母评分的体系来解释 AUC 的得分：

- 0.9 ~ 1.0 = A (优秀)。
- 0.8 ~ 0.9 = B (良好)。
- 0.7 ~ 0.8 = C (一般)。
- 0.6 ~ 0.7 = D (很差)。
- 0.5 ~ 0.6 = F (无法区分)。

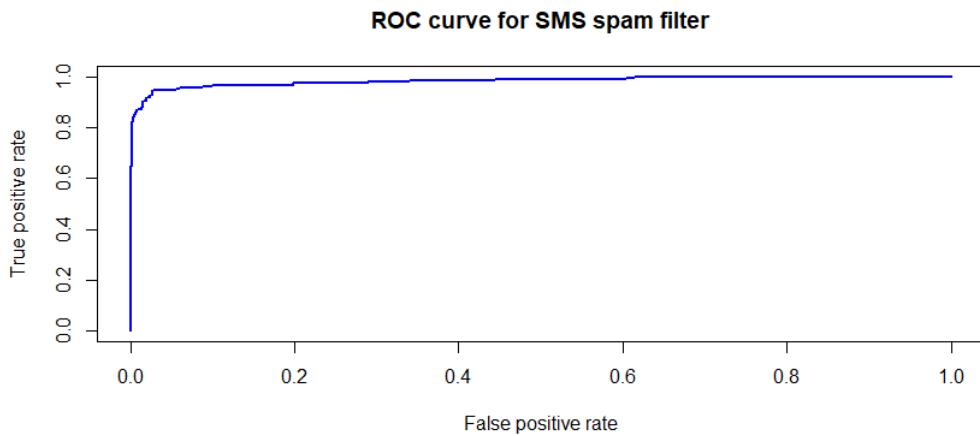
与类似的很多评价尺度一样，其水平可能在某些人任务上的表现要强于其他任务。这个分类方法比较主观。值得注意的是，两个 ROC 曲线可能形状不同但是具有一样的 AUC。由于这个原因，AUC 可能具有误导性。最好的方式是在使用 AUC 的同时也对 ROC 曲线进行定性分析。

使用 ROCR 包绘制 ROC 曲线需要为 pred 对象构建一个性能对象，我们之前计算过了。因为 ROC 曲线是真阳性和假阳性的线图，所以可以简单地调用 performance() 函数来绘图，只需要指定 tpr 和 fpr 度量：

```

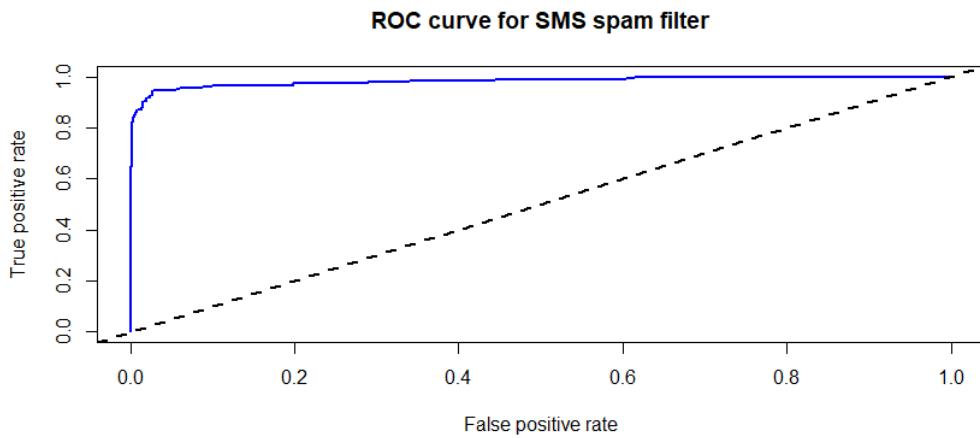
1 > perf <- performance(pred, measure = "tpr", x.measure = "fpr")
2 > plot(perf, main = "ROC curve for SMS spam filter", col = "blue", lwd = 2)

```



加一条参考线:a 表示截距,b 表示斜率

```
1 > abline(a = 0, b = 1, lwd = 2, lty = 2)
```



定性的分析,可以看到 ROC 曲线占据了图形左上角的区域,这意味着虚线代表的无用分类器相比,它更接近于完美分类器. 如果要定量地确认这个事实,可以用 ROCR 包计算 AUC:

首先,我们需要创建另一个性能类型对象,这次我们设定 `measure="auc"`:

```
1 > perf.auc <- performance(pred, measure = "auc")
```

因为 `perf.auc` 是一个对象 (明确的说是一个 S4 对象), 所以需要使用一个特殊的符号来访问存储在其中的值.S4 对象存储信息的位置称为槽. `str` 函数可以查看一个对象的槽.

```
1 > str(perf.auc)
2 Formal class 'performance' [package "ROCR"] with 6 slots
3 ..@ x.name      : chr "None"
4 ..@ y.name      : chr "Area under the ROC curve"
5 ..@ alpha.name  : chr "none"
6 ..@ x.values    : list()
7 ..@ y.values    :List of 1
8 ...$ : num 0.983
```

```
9 ..@ alpha.values: list()
```

注意, 槽的前缀是 @. 要访问 AUC 的值, 其作为列表对象存储在 y.values 槽内, 可以使用符号 @ 和 unlist() 函数将列表简化为数值向量:

```
1 > as.numeric(perf.auc@y.values)
2 [1] 0.9829999
```

短信分类器的 AUC 是 0.98, 非常高. 但是我们如何才能知道这个模型对于其他的数据集也是否表现得足够好? 见下.

10.2 评估未来的性能

有些 R 机器学习添加包在模型构建过程中显示混淆矩阵和性能度量指标. 这些统计量的目的是提供对模型再带入误差的认识, 虽然模型直接从数据中构建, 但是当训练数据进行了错误的预测时就会产生再带入误差. 该信息用于一个粗略的诊断工具, 尤其是用于识别明显不好的分类器.

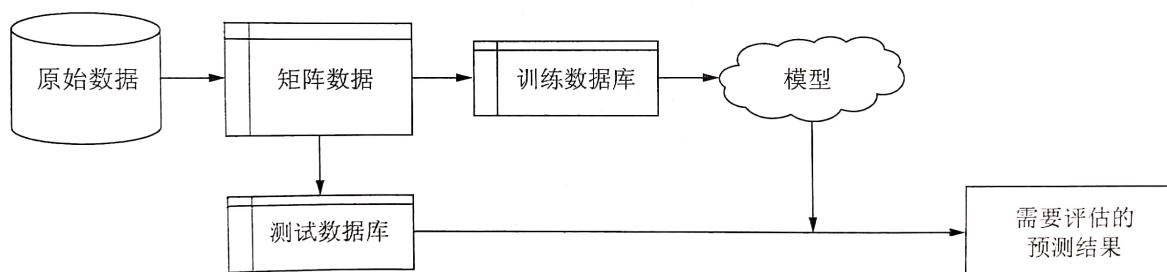
但是对于未来的性能, 再带入误差并不是很好的标识器. 例如, 如果一个模型通过死记硬背的方式对每个训练的个体都进行了完美分类 (再带入误差为 0), 那么对于它从来没有见过的数据将无法进行预测.

与信赖再带入误差相比, 更好的方式是评估模型对其从未见过数据的性能. 在前面的章节中我们使用过这种方法, 当时我们将数据分成了训练集和测试集. 但是在某些情况下, 创建训练集和测试集并不总是有用, 例如当数据集本来就很小时.

caret 包可以帮助我们.

10.2.1 保持法

在前面章节中, 我们将数据划分成训练集和测试集的过程称为保持法.



因为保持法对未来的性能能真实的精确估计, 所以绝不允许测试数据集的结果影响模型. 如果基于重复测试的结果选择一个最好的模型, 那么很容易在不知不觉中就违反了这个原则. 一种替代方法是继续对数据进行划分, 在训练数据集和测试数据集之外, 分出第三个数据集, 也就是验证数据集.

验证数据集用来对模型进行迭代和改善, 测试数据集只使用一次, 在最后的步骤中输出对未来预测的错误率的估计. 一种典型的划分方式是 50% 的训练数据集, 25% 测试数据集和 25% 验证数据集.

创建保持样本的一种简单方式是使用随机数发生器将记录划分到各个数据集. 这种技术最早在第五章用来创建训练数据集和测试数据集.

```
1 > credit<-read.csv("E:/6. 机器学习/ml&r/Machine Learning with R/
```

```

2 chapter 10/credit.csv")
3 > random_ids <- order(runif(1000)) #将1-1000的行id随机排序
4 > credit_train <- credit[random_ids[1:500],] #训练集
5 > credit_validate <- credit[random_ids[501:750], ] #验证集
6 > credit_test <- credit[random_ids[751:1000], ] #测试集

```

这种保持抽样的方式存在一个问题, 每个划分包含不同类别的数量可能过大或者过小. 为了降低这种情况的发生可能性, 可以使用分层随机抽样的方法. 虽然在平均的情况下, 简单随机抽样包含的各类别的比例大概与总体数据集中的比例相同, 但是分层随机抽样可以确保随机划分后每个类别的比例与总体数据中的比例近似相等.

caret 包的 createDataPartition() 函数可以基于分层抽样方法来创建随机的划分. 使用该函数时, 需要指定类别向量 (此处 default 变量标识是否贷款违约), 参数 p 变送包含在该划分中样本的比例. 参数 list=FALSE 防止结果存储成列表的形式.

```

1 > library(caret)
2 载入需要的程辑包: lattice
3 载入需要的程辑包: ggplot2
4 > in_train <- createDataPartition(credit$default, p = 0.75, list = FALSE)
5 > credit_train <- credit[in_train, ] #in_train表示包含在训练样本中的行号.
6 > credit_test <- credit[-in_train, ]

```

虽然这种方式可以确保分布的均匀性, 但是分层抽样并不能保证其他类型的代表性. 有些样本包含过多或者过少的困难样本、易预测样本或者极端值. 尤其是当数据量少时特别明显, 每个部分的数据集不足以包含所有的情况.

除了可能的有偏样本以外, 保持法的另一个问题是大量的数据部分都被用来测试和验证模型. 在模型的性能被度量之前都无法用来训练模型. 这种性能估计的方式过于保守.

一种称为重复保持的技术有时用来缓解随机构建训练集的问题. 重复保持法是保持法的一种特殊形式, 它对多个随机保持样本的模型分别评估, 然后用结果的均值来评价整个模型的性能. 使用多重保持样本时, 模型在无代表性数据上训练和测试的可能性就比较小了. 我们将在下一节中对这个思路进行扩展.

10.2.2 交叉验证

重复保持法是 k 折交叉验证 (或者 k 折 CV) 的基础. k 折交叉验证已经成为业界评估模型性能的标准. 与可能对同一条记录使用多次重复随机抽样的方法不同, k 折 CV 将数据随机地分成 k 个完全隔开的部分, 这些部分称为折.

虽然 k 可以设置为任意数值, 但是到目前为止最常用的惯例是使用 10 折交叉验证 (10 折 CV). 经验证据告诉我们, 使用更大的数后带来的好处并不明显. 对于这 10 折中的每一折 (每一折包含总数据的 10%), 机器学习模型使用剩下 90% 的数据建模. 包含 10% 数据的这一折用来评估. 训练和评估模型的过程重复 10 次 (10 次不同的训练和测试) 之后, 将输出所有折的平均性能指标.

我们可以使用 caret 包中的 creatFolds() 函数来创建交叉验证的数据集.

```

1 > folds <- createFolds(credit$default, k = 10)

```

```

2 > str(folds)
3 List of 10
4 $ Fold01: int [1:100] 22 31 32 39 48 53 65 70 72 84 ...
5 $ Fold02: int [1:100] 3 23 40 52 55 57 74 78 89 94 ...
6 $ Fold03: int [1:100] 1 5 10 12 17 18 37 42 61 62 ...
7 $ Fold04: int [1:100] 25 29 44 46 60 66 67 68 92 96 ...
8 $ Fold05: int [1:100] 6 27 30 79 85 100 118 120 121 129 ...
9 $ Fold06: int [1:100] 8 15 24 47 49 54 73 76 82 103 ...
10 $ Fold07: int [1:100] 4 9 13 16 26 50 63 80 98 105 ...
11 $ Fold08: int [1:100] 11 19 21 36 58 59 64 81 83 86 ...
12 $ Fold09: int [1:100] 28 33 34 35 38 43 56 95 99 104 ...
13 $ Fold10: int [1:100] 2 7 14 20 41 45 51 71 75 91 ...

```

createFolds 的结果是一个列表, 包含了 10 个向量, 每个向量都是这一折所抽取数据的行号.

要创建训练集和测试集来建模和评估, 需要一个额外的步骤. 下面的代码显示了通过第一折创建数据的过程. 与我们在分层抽样时做的一样:

```

1 > credit01_train <- credit[folds$Fold01, ]
2 > credit01_test <- credit[-folds$Fold01, ]

```

要想执行完整的 10 折 CV, 这个步骤需要重复总共 10 次, 每一次都要建立模型, 然后计算模型的性能. 最后通过对所有的性能度量取均值得到总体的性能. 用 10 折 CV 方法对 credit 数据集建立 C5.0 决策树模型, 然后估计 Kappa 统计量.

载入添加包:

```

1 > library(caret)
2 > library(C50)
3 > library(irr)
4 载入需要的程辑包: lpSolve

```

其次, 像先前一样创建 10 折列表, 使用 set.seed 函数是为了保证读者自己运行后面的代码的结果与书中一致.

```

1 > set.seed(123)
2 > folds <- createFolds(credit$default, k = 10)

```

最后使用 lapply 对列表每个元素进行相同的操作.

```

1 > cv_results <- lapply(folds, function(x) {
2   +   credit_train <- credit[x, ]
3   +   credit_test <- credit[-x, ]
4   +   credit_model <- C5.0(default ~ ., data = credit_train)
5   +   credit_pred <- predict(credit_model, credit_test)
6   +   credit_actual <- credit_test$default
7   +   kappa <- kappa2(data.frame(credit_actual, credit_pred))$value
8   +   return(kappa)

```

```
9     + })
```

结果的 Kappa 统计量保存在 `cv_results` 对象的列表中. 最后来个均值: 由于 `cv_results` 并不是数值向量, 所以应该使用 `unlist` 函数来消除列表的结构.

```
1 > str(cv_results)
2 List of 10
3 $ Fold01: num 0.243
4 $ Fold02: num 0.243
5 $ Fold03: num 0.147
6 $ Fold04: num 0.08
7 $ Fold05: num 0.156
8 $ Fold06: num 0.212
9 $ Fold07: num 0.118
10 $ Fold08: num 0.156
11 $ Fold09: num 0.0882
12 $ Fold10: num 0.195
13 > mean(unlist(cv_results))
14 [1] 0.1637667
```

不幸的是, 这个 Kappa 值非常低, 在解释评分体系中对应着”很差”, 说明这个信用记分模型的效果并不比随机猜测好很多. 下一章我们基于 10 折 CV 方法研究自动方法帮助改进这个模型.

10.2.3 自助法抽样

还有一种虽然不如 k 折 CV 这么受欢迎但是也被广泛使用的方法是自助法抽样 (bootstrap sampling)，简称自助法 (bootstrap)。一般来说，它主要指一些统计方法，通过对数据进行随机抽样的方式来估计大数据集的内容。当该原理应用到机器学习模型性能时，意味着创建一些随机选取的训练集和测试集，然后用来估计性能的统计量。各种随机产生的数据集的结果可以通过平均值计算得到一个最终的估计值，用来评估未来的性能。

那么，这个过程与 k 折 CV 有什么不同呢？交叉验证方式将数据分成彼此分隔的部分，每个样本只能出现一次，而自助法通过有放回的抽样方式使得每个样本可以被选择多次。这意味着假设原始数据包含 n 个样本，那么自助法可以创建一个或者多个仍然包含 n 个样本的数据集，有些样本是重复的。相应的测试数据集仍然由未选入训练集中的样本来构建。

使用之前提到的有放回的抽样方式，每个样本包含在训练集中的概率是 63.2%。因此，样本包含在测试集中的概率是 36.8%。换句话说，测试集只能代表 63.2% 的可能样本，因为很多样本重复了。相比之下，10 折 CV 方法可以使用 90% 的数据用来训练，自助法抽样对完整数据集的代表性更弱。

显然，只利用 63.2% 的数据训练出来的模型的性能不如更大量数据训练出来的模型，对自助法性能的估计也远不如使用全体数据训练的结果。有一种特殊的自助法的情况可以处理这个问题，称为 **0.632 自助法**，通过训练数据集（过于乐观）和测试数据集（过于悲观）的函数来计算最终的性能度量。最终的错误率可以由以下公式计算：

$$\text{错误率} = 0.632 \times \text{错误率}_{\text{测试}} + 0.368 \times \text{错误率}_{\text{训练}}$$

自助法比交叉验证具有一个优势，它对于小数据集的效果更好。此外，自助法抽样在性能度量之外还有其他的应用。特别地，第 11 章将学习如何利用自助法抽样的原理来改善模型的性能。

Chapter 11

提高模型性能

本章将基于之前的内容介绍一些能提高机器学习算法预测性能的技术

- 如何通过寻找训练条件的优化集合来调整机器学习模型的性能.
- 将多个模型组合起来从而处理更富挑战性问题的方法
- 获得机器学习算法最大程度性能的最先进的技术

并不是所有这些方法都适用于每个问题, 但是你会发现在机器学习竞赛中的胜利者至少会用到其中一种方法.

11.1 调整多个模型来提高性能

有些机器学习问题都很适合使用前几章介绍的多个模型. 在这种情况下, 我们不需要花太多时间进行迭代和优化, 因为他已经足够好了. 另一方面, 有些问题本质上就很难. 需要学习的潜在概念极其复杂, 需要对很多微妙的关系具有很好的理解, 或者可能具有随机元素, 使得从噪声中分离有用的信号变得很难.

在第五章中, 我们尝试了一个复杂的问题, 识别可能违约的贷款. 虽然我们可以使用性能调节方法得到一个可以接受的分类准确度 72%, 但是根据第十章更仔细的审视之后, 我们认识到这种高准确度可能会造成误导. 尽管准确度非常合理, 但是 Kappa 统计量只有大约 0.2, 这说明了该模型实际的性能其实是有些糟糕的.

在第五章中, 我们首先使用股票 C5.0 决策树来对这个信用数据建立分类模型. 然后我们尝试通过调节 `trials` 参数来增加提升 (boosting) 迭代的次数从而提升模型的性能. 通过将默认的 1 次增加到 10 和 100 次, 可以增加模型的准确度. 调节模型合适的选项的过程称为**参数调整**.

参数调整不仅限于决策树. 例如, 我们通过寻找最合适的 `k` 值来调整 `k` 邻近模型. 在神经网络和支持向量机模型中使用大量的选项. 大多数机器学习算法都可以调整至少一个参数, 而大多数复杂的模型都提供了很大数目的方式来调整模型从而进行更好的拟合. 虽然这儿可以让模型更适合数据, 但是尝试所有可能选项的复杂度是吓人的. 需要使用一种更系统的方式.

11.1.1 使用 caret 进行自动参数调整

与其对模型的每个参数选择任意值, 不如对搜索参数值的过程进行引导从而找到最优的组合.`caret` 包提供了很多工具可以帮助自动调整参数. 最核心的功能是提供了一个 `train()` 函数作为标准接口, 为

分类和回归任务训练 150 种不同的机器学习模型. 使用该函数时, 可以选择评估得方法和度量, 它自动搜寻一个最优的模型.

自动参数调整需要考虑以下三个问题:

- 需要使用数据来训练哪种机器学习模型 (或者算法的具体实现)?

需要在机器学习的任务和 150 个模型之间做适当的匹配. 显然需要对这些机器学习模型的广度和深度有很好的理解. 本书提供了前者所需要的背景, 额外的练习对后者很有用. 此外, 排除法也很有用: 通过判断任务是分类还是回归就可以排除几乎一半的模型; 其他的可以通过数据的形式或者避免黑箱模型来排除. 不管怎样, 你还可以尝试多种模型, 然后通过比较模型的结果来选择一个最好的.

- 哪些模型参数是可以调整的, 它们能调整的空间有多大?

这个问题很大程度上是被模型的选择所决定的, 因为每个算法使用唯一的一套参数. 本书设计的预测模型的可调节参数都列在下表. 注意虽然有些模型还有没列出来的选项, 但是 caret 包支持下表中列出的选项. 自动调整的目标时搜索候选模型的集合, 该集合由所有参数组合的矩阵或者网格的形式构成. 因为要想遍历所有参数的所有可能值是难以实现的.caret 默认对每个参数最多搜索 3 个可能值, 假设一共有 p 个参数, 这意味着 3^p 个候选模型将会被测试.

模型	学习任务	方法名	参数
k 近邻	分类	knn	k
朴素贝叶斯	分类	nb	fL, usekernel
决策树	分类	C5.0	model, trials, winnow
OneR 规则学习器	分类	OneR	无
RIPPER 规则学习器	分类	JRip	NumOpt
线性回归	回归	lm	无
回归树	回归	rpart	cp
模型树	回归	M5	pruned, smoothed, rules
神经网络	二者皆可	nnet	size, decay
支持向量机 (线性核)	二者皆可	svmLinear	C
支持向量机 (径向基核)	二者皆可	svmRadial	C, sigma
随机森林	二者皆可	rf	mtry

- 使用何种标准来评估模型从而找到最优的候选者?



因为 `caret` 默认的搜索网格对于实际的机器学习问题可能不是很理想，所以该函数还允许用户自定义搜索网格，通过简单的命令即可定义。我们会在后面进行介绍。

自动模型调整的第三个也是最后一个步骤是选择一种方法从候选者中识别最好的模型。我们使用第 10 章讨论过的方法（比如，选择重采样的策略）来创建训练集和测试集，或者使用度量预测准确度的模型性能统计量。

`Caret` 添加包支持我们介绍过的所有重采样策略和大部分的性能统计量，包括准确度和 Kappa 值（用于分类器）以及 R 方值或者 RMSE（用于数值模型）等统计量。如果需要的话，也可以使用代价敏感度方面的度量方式，比如灵敏度、特异性、ROC 曲线下面积（AUC）等。

默认情况下，`caret` 在选择最优模型时，通过这些性能度量指标的最大值来选择。因为这样的方法在实践中有时会通过大量增加模型复杂度的方式来选择边际性能递增的模型，该添加包也提供了可供选择的其他函数。

在各种选项中，大部分默认值都是比较合理的。例如，在分类模型中使用自助法抽样的预测准确度来选择最优性能的优化器。从这些默认值开始，我们使用 `train()` 函数来设计各种实验。

11.1.1.1 创建简单的调整的模型

使用 `caret` 包的默认设置来调整信誉评分模型。从这里学习如何调整选项。

最简单的调整模型的方式只需要通过 `method` 参数来指定模型的类型：

```
1 > credit<-read.csv("E:/6. 机器学习/ml&r/Machine Learning with R/chapter 11/credit.csv")
2 > library(caret)
3 载入需要的程辑包: lattice
4 载入需要的程辑包: ggplot2
5
6
7 > set.seed(300)
8 > # 初始化R的随机数发生器，设定任意一个seed参数使随机数遵循一个预先设定
9 # 的序列
10
11 > m <- train(default ~ ., data = credit, method = "C5.0")
12 > #R的公式接口可以把树定义成default ~ .，表示对贷款违约的状态(yes/no)
13 #进行建模，使用了信用数据集中的所有其他变量。参数method= "C5.0"告诉caret
14 #使用C5.0决策树算法。
```

如果要查看实验结果，命令 `str(m)` 会列出所有相关的信息，但是列出的信息太多了，简单键入该变量名 `m`，就会得到摘要的结果。

```
1 > m
2 C5.0
3
4 # 输入数据的简单描述
```

```

5 1000 samples
6 16 predictor
7 2 classes: 'no', 'yes'
8
9 # 预处理和重抽样方法应用情况信息: 25个自助法样本, 每个都包含1000个样本用来建模.
10 No pre-processing
11 Resampling: Bootstrapped (25 reps)
12 Summary of sample sizes: 1000, 1000, 1000, 1000, 1000, 1000, ...
13 Resampling results across tuning parameters:
14
15 # 候选模型评估的列表: 在这一节里, 我们可以确信12个不同的模型被测试到了, 基于
16 # 3个C5.0调整参数的组合: model, trias, winnow. 每个候选模型的准确度和Kappa统
17 # 计量的均值都得到了展示.
18 model winnow trias Accuracy Kappa
19 rules FALSE 1 0.6918852 0.2749843
20 rules FALSE 10 0.7144478 0.3112184
21 rules FALSE 20 0.7270463 0.3344054
22 rules TRUE 1 0.6947856 0.2752201
23 rules TRUE 10 0.7144334 0.3150388
24 rules TRUE 20 0.7268154 0.3393965
25 tree FALSE 1 0.6909682 0.2569822
26 tree FALSE 10 0.7256597 0.2996244
27 tree FALSE 20 0.7322016 0.3157605
28 tree TRUE 1 0.6920217 0.2604578
29 tree TRUE 10 0.7279264 0.3099733
30 tree TRUE 20 0.7299631 0.3131074
31
32 # 最佳模型的选择: 根据提示, 具有最大准确度(0.73)的模型被选为最佳.
33 Accuracy was used to select the optimal model using the largest value.
34 The final values used for the model were trias = 20, model = tree and winnow
35 = FALSE.

```

train() 函数使用最佳模型中的参数对所有数据建立模型, 并将其储存在 m\$finalModel 对象中.

在大多数情况下, 无需直接操作 finalModel 子对象, 而是使用 predict() 通过 m 对象来进行预测, 同时还提供了一些额外的功能. 将最佳模型应用到训练数据中进行预测, 可以使用以下命令:

```

1 > p <- predict(m, credit)
2 > table(p, credit$default)
3
4 p      no yes
5   no    700   2
6   yes     0 298

```

在用来训练最终模型的 1000 个样本中, 只有 2 个被错误分类. 记住这只是重新代入误差, 不能看成是对未来数据性能的度量. 自助法估计的 73%(输出结果有显示) 是对未来性能的更现实的估计.

将 `predict()` 直接用于 `train()` 对象还会有额外的好处. 注意不是调用 `finalModel` 子对象或者使用最佳模型的参数来训练新模型.

首先, `train()` 函数应用到数据中的任何数据预处理方法也会用类似的方式应用到产生预测的数据中. 这包括中心化和标准化 (使用 k 邻近) 的数据转换, 缺失值处理以及一些其他方法. 这确保用来建模的数据准备步骤在模型部署后仍然保留.

其次, `predict()` 函数提供了标准的接口用来得到预测的类别值和概率—即使是通常的模型也需要额外的步骤来得到这些信息. 预测的类别的会默认提供:

```
1 > head(predict(m, credit, type = "raw"))
2 [1] no  yes no  no  yes no
3 Levels: no yes
```

设置参数 `type="prob"` 得到每一类估计的概率:

```
1 > head(predict(m, credit, type = "prob"))
2      no      yes
3 1 0.9606970 0.03930299
4 2 0.1388444 0.86115560
5 3 1.0000000 0.00000000
6 4 0.7720279 0.22797207
```

11.1.1.2 定制调整的过程

我们之前创建的决策树证明 `caret` 包可以在最少介入下生成最优模型. 默认设置可以使高性能的模型能够很容易创建. 但可能需要改变默认的评估标准从而跟更适合自己的机器学习问题. 之前介绍的每一个步骤都是可以针对具体学习任务进行定制.

为了说明, 我们对之前信贷决策树的工作进行一些修改. 第十章我们使用了 10 折交叉验证来估计 Kappa 统计量. 这里我们会做相同的事情, 使用 Kappa 优化决策树的提升 (boosting) 参数 (提升第五章介绍的决策树的准确度).

1. 用 `trainControl()` 函数创建控制列表

用来创建一系列的配置选项, 也称为控制对象, 与 `train()` 函数一起使用. 这些选项考虑到了诸如重抽样策略以及用于选择最佳模型的度量这些模型评价标准的管理. 虽然这些函数可以用于几乎所有参数调整的方面, 但是我们只专注于两个重要的参数:`method` 和 `selectionFunction`.

对于 `trainControl()` 函数, `method` 参数用来设置重抽样的方法, 例如保持抽样或者 k 折交叉验证. 下表列出了 `caret` 调用这些方法时使用的缩写以及用来调节样本量和迭代次数的所有额外参数:

重抽样方法	方法名	额外的选项和默认值
保持抽样	LGOCV	p = 0.75 (训练数据比例)
k 折交叉验证	cv	number = 10 (折的数目)
重复 k 折交叉验证	repeatedcv	number = 10 (折的数目) repeats = 10 (迭代的次数)
自助法抽样	boot	number = 25 (重抽样迭代的次数)
0.632 自助法	boot632	number = 25 (重抽样迭代的次数)
留一交叉验证法	LOOCV	无

selectionFunction 参数可以设定一函数用来在各个候选者中选择最优的模型. 其中包含了三个参数.

- best 函数简单地选择具有最好的某特定度量值的候选者, 这是默认选项. 下面两个用来在最好模型性能的特定阈值之内选择最节俭的 (最简单的).
- oneSE 函数选择最好性能标准差之内的最简单的候选者.
- Tolerance 选择某个用户指定比例之内的最简单的候选者.

使用 10 折交叉检验和 oneSE 选择函数可以创建一个名为 ctrl 的控制对象

```

1 > ctrl <- trainControl(method = "cv", number = 10,
2 +                               selectionFunction = "oneSE")
```

2. 创建用来优化参数的网格

网格的每一列表示模型中的一个参数, 前面用点号做前缀. 因为使用 C5.0 决策树, 故名为.model,.trials,.winnow 的列.

使用 expand.grid() 函数创建这样的数据框. 假设需要参数 model = "tree" 和 winnow = "FALSE" 保持不变, 同时 trials 需要选择 8 个值.

```

1 > grid <- expand.grid(.model = "tree",
2 +                         .trials = c(1, 5, 10, 15, 20, 25, 30, 35),
3 +                         .winnow = "FALSE")
4
5 > grid
6 .model .trials .winnow
7 1 tree      1 FALSE
8 2 tree      5 FALSE
9 3 tree     10 FALSE
10 4 tree     15 FALSE
11 5 tree     20 FALSE
12 6 tree     25 FALSE
13 7 tree     30 FALSE
14 8 tree     35 FALSE
```

每一行都可以用来创建一个候选模型, 使用该行中模型参数的组合来创建.

3. 准备运行完全定制的 train() 实验

设置随机数种子来确保可重复的结果. 但是这一次, 我们传送的是自己定制的控制对象和调整参数的网格, 同时添加参数 `matrix="Kappa"`, 表示模型评估函数用到的统计量-在这个例子中, 是 `oneSE`.

```
1 > library(caret)
2 载入需要的程辑包: lattice
3 载入需要的程辑包: ggplot2
4 > set.seed(300)
5 > m <- train(default ~ ., data = credit, method = "C5.0",
6 +               metric = "Kappa",
7 +               trControl = ctrl,
8 +               tuneGrid = grid)
9 Warning message:
10 In Ops.factor(x$winnow) : '!' not meaningful for factors
11 > m
12 C5.0
13
14 1000 samples
15 16 predictor
16 2 classes: 'no', 'yes'
17
18 No pre-processing
19 Resampling: Cross-Validated (10 fold)
20 Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...
21 Resampling results across tuning parameters:
22
23   trials  Accuracy   Kappa
24   1        0.710    0.2859380
25   5        0.726    0.3256082
26  10        0.725    0.3054657
27  15        0.726    0.3204938
28  20        0.733    0.3292403
29  25        0.732    0.3308708
30  30        0.733    0.3298968
31  35        0.738    0.3449912
32
33 Tuning parameter 'model' was held constant at a value of tree
34 Tuning
35 parameter 'winnow' was held constant at a value of FALSE
36 Kappa was used to select the optimal model using the one SE rule.
37 The final values used for the model were trials = 5, model = tree and winnow
38 = FALSE.
```

```

39 > View(credit)
40 > load("D:/Program Files (x86)/Rstudio/.RData")
41 > library(caret)
42 > set.seed(300)
43 > m <- train(default ~ ., data = credit, method = "C5.0",
44 +               metric = "Kappa",
45 +               trControl = ctrl,
46 +               tuneGrid = grid)
47 Warning message:
48 In Ops.factor(x$winnow) : '!' not meaningful for factors
49 > m
50 C5.0
51
52 1000 samples
53 16 predictor
54 2 classes: 'no', 'yes'
55
56 No pre-processing
57 Resampling: Cross-Validated (10 fold)
58 Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...
59 Resampling results across tuning parameters:
60
61   trials  Accuracy   Kappa
62   1        0.710    0.2859380
63   5        0.726    0.3256082
64  10        0.725    0.3054657
65  15        0.726    0.3204938
66  20        0.733    0.3292403
67  25        0.732    0.3308708
68  30        0.733    0.3298968
69  35        0.738    0.3449912
70
71 Tuning parameter 'model' was held constant at a value of tree
72 Tuning
73 parameter 'winnow' was held constant at a value of FALSE
74 Kappa was used to select the optimal model using the one SE rule.
75 The final values used for the model were trials = 5, model = tree and winnow
76 = FALSE.

```