## Bootstrap

自抽样自举（自助法）（Bootstrap）是（作用）利用有限样本数据信息来估计总体特征（如总体均值、总体分布等）时所采用的一种非参数方法。

(怎么做)在自助法中，我们是从已有样本中进行重复有放回的随机抽样，形成大小与原样本相同的一组新样本，并计算得到新样本的样本统计量（例如均值、标准差等,也可以是其他需要计算得到的参数）。然后重复进行这一过程k次（通常k>1000），得到k个统计量，将其作为总体统计量的近似值。

自助法可以通过模拟有限样本数据在整体样本上的性质，来对总体特征进行估计。它不需要对总体的分布做任何假设，不受异常值的影响，适用于一般的统计推断。

在统计学中，自助法的应用非常广泛，如在机器学习、回归分析、生存分析、时间序列分析等领域都有广泛应用。

```{r cars}

# 1、获取多元正态分布的随机数和引用bootstrap，需要的包MASS和boot

library(MASS)

library(boot)

# 2、设置随机数

set.seed(123)

# 3、随机数相关参数，大小，均值，方差1，方差2，协方差，方阵（协方差矩阵）

Size=1000;mean1=0;mean2=0;var1=1;var2=1.25;cov12=0.5;

covariance\_matrix=matrix(c(var1,cov12,cov12,var2),nrow=2)

# 4、mvrnorm()函数可以用来生成多元正态分布的随机数

# n: 生成的随机数的个数

# mu: 向量，代表多元正态分布的均值

# Sigma: 方阵，代表多元正态分布的协方差矩阵

# empirical: 逻辑变量，表示是否使用样本的协方差矩阵来代替理论协方差矩阵

# 生成符合多元正态分布的集合

sample\_dat1=mvrnorm(n=1000,mu=c(mean1,mean2),covariance\_matrix)

# 手动bootstrap

# 从已有样本中进行重复有放回的随机抽样，形成大小与原样本相同的一组新样本，并计算得到新样本的样本统计量（例如均值、标准差等）。然后重复进行这一过程k次（通常k>1000），得到k个统计量，将其作为总体统计量的近似值

for(i in 1:1000){

# 从已有的样本中取重复的随机抽样的索引，从1~nrow（dat1）中重复放回取nrow个

sample\_no=sample(nrow(sample\_dat1),nrow(sample\_dat1),replace = TRUE)

# 然后根据索引向量，取出新样本

sample\_dat=sample\_dat1[sample\_no,]

# 提取获得样本的相关统计量

var1\_vec=var(sample\_dat[,1])

var2\_vec=var(sample\_dat[,2])

cov12\_vec=cov(sample\_dat[,1],sample\_dat[,2])

alpha\_v=(var1\_vec-cov12\_vec)/(var1\_vec+var2\_vec-2\*cov12\_vec)#计算一个统计量alpha

if(i==1)

alpha\_vec=alpha\_v

else

alpha\_vec=c(alpha\_vec,alpha\_v)#获得k个统计量alpha

}

# 利用自抽样bootstrap函数

#函数这里需要两个参数，随机抽样后的数据集sample\_dat1，随机抽取的索引数组sample\_no)

alpha\_fun<-function(sample\_dat1,sample\_no){

sample\_dat=sample\_dat1[sample\_no,]

var1\_vec=var(sample\_dat[,1])

var2\_vec=var(sample\_dat[,2])

cov12\_vec=cov(sample\_dat[,1],sample\_dat[,2])

alpha\_ve=(var1\_vec-cov12\_vec)/(var1\_vec+var2\_vec-2\*cov12\_vec)

return (alpha\_ve)

}

boot1=boot(sample\_dat1,alpha\_fun,R=1000)

print(boot1)

```

在R语言中，`boot()`函数可以用于对数据进行非参数自助法（bootstrap）估计。该函数的一般形式为：

```

boot(data, statistic, R, ...)

```

其中各参数的含义为：

- `data`：原始数据集。

- `statistic`：自助样本的统计量（函数）。

- `R`：自助样本的数量。

- `...`：其他需要传入到`statistic`函数里的参数。

例如，假设我们需要对一组数据进行均值的自助法估计，可以使用以下代码：

```{r }

data <- rnorm(1000) # 随机生成1000个服从标准正态分布的数据

mean\_boot <- boot(data, mean, R = 1000) # 对数据进行1000次自助法重抽样，计算均值估计

```

这样，我们就可以得到自助法均值估计的结果。其中，变量`mean\_boot`是一个`boot()`对象，可以使用`summary()`函数查看结果摘要信息，包括估计值的置信区间和标准误等。

需要注意的是，在使用`boot()`函数进行自助法估计时，选取的统计量应该与问题相关。例如，对于回归问题，可以考虑使用平均绝对误差（Mean Absolute Error）或平均二乘误差（Mean Squared Error）等作为自助样本的统计量。同时，也可以自定义统计量函数以满足不同的需求。

## 交叉检验（cross validation）

留一法交叉验证(Leave -One -Out CV, LOOCV)

假设数据集含有 n个数据 𝑥 1 , 𝑦 1 , … , 𝑥 𝑛 , 𝑦𝑛 ， LOOCV每次选择 1个数据作为 验证集 ，采用 MSE(Mean Squared Error平方差)作为评价指标 （1） 𝑥1, 𝑦1 作为验证集，其余 𝑛 − 1 个数据作为 训练集 𝑥2, 𝑦2 , … , 𝑥𝑛, 𝑦𝑛 : 𝑀𝑆𝐸1 = 𝑦1 − 𝑦ො1 2 （2） 𝑥2, 𝑦2 作为验证集，其余 𝑛 − 1 个数据作为训练 集 𝑥1, 𝑦1 , … , 𝑥𝑛, 𝑦𝑛 : 𝑀𝑆𝐸2 = 𝑦2 − 𝑦ො2 2 … （n） 𝑥𝑛, 𝑦𝑛 作为验证集，其余 𝑛 − 1 个数据作为训练 集 𝑥1, 𝑦1 , … , 𝑥𝑛−1, 𝑦𝑛−1 : 𝑀𝑆𝐸𝑛 = 𝑦𝑛 − 𝑦ො𝑛 2 LOOCV得到评价指标是 𝑛个检测误差的平均值

LOOCV优点：

（1）偏差更小： 使用的训练数据集有 𝑛 − 1 个数据，接近整个数据集 ，所以不大可能会高估检验错误率(Test error rate)。

（2）结果稳定： 验证集法每次的估计结果都可能发生变化，因为训练 和验证集是随机分割的。对于LOOCV，每次运行产 生的结果是不变的。

（3）可以与任何预测模型结合使用。

LOOCV缺点： 计算量巨大，需要计算𝑛次。

在R语言中，`cv()`函数可以用于进行交叉验证（cross-validation）操作，以评估模型的预测能力。该函数常常用于选择模型的最优参数或进行模型选择。`cv()`函数的一般形式为：

```

cv(object, K, ...)

```

其中，`object`是需要被评估的模型或预测函数；`K`是交叉验证的折数（fold number），通常取值为5或10；`...`可以传递额外参数给需要被评估的函数。

例如，假设我们要对线性回归模型的预测能力进行交叉验证，可以使用以下代码：

```R

library(ISLR) # 加载数据集

data(College) # 加载学院成绩数据集

fit <- lm(Apps ~ ., data = College) # 拟合线性回归模型

cv\_result <- cv(fit, K = 10) # 进行10折交叉验证

print(cv\_result$delta) # $delta ->打印均方误差（Mean Square Error）和R方值（R-Squared）

```

不太懂，感觉这个说不清楚

在R语言中，cv函数（cross-validation function）一般用于模型选择（model selection）或参数调优（parameter tuning），它可以通过交叉验证（cross-validation）方法评估模型的泛化性能。

cv函数的返回值通常是一个包含以下几个部分的列表（list）：

1. `call`：包含了cv函数的调用。

2. `model`：最终选择的模型，通常是拟合在整个数据集上的。

3. `k`：采用的折数（folds）。

4. `estimate`：一个与k长度相等的元素向量，表示每一折（fold）上用测试集（test set）对该模型计算出的预测值的误差估计（estimate）。

5. `prob`：一个与k长度相等的元素向量，通常只有在分类问题中出现。表示每个测试集被正确分类的概率（probability）。

6. `pred`：一个与k长度相等的元素列表，每个元素是一个向量，包含了用测试集计算出的预测（prediction）值。

7. `fit`：一个与k长度相等的元素列表，每个元素是一个包含所有训练集（training set）上的拟合值（fitted value）。

8. `call1`：在每个交叉验证的轮次中cv函数的调用。

9. `dots`：其他参数和选项（arguments and options）。

具体的返回值内容可能会因不同的算法或参数而有所不同。

在R语言中，cv.glm函数是用于交叉验证的广义线性模型（generalized linear model）拟合函数。它能够在给定的数据集上计算交叉验证误差（cross-validation error），并且可以通过交叉验证方法选择参数。cv.glm函数的返回值通常是一个包含以下几个部分的列表（list）：

1. `call`：cv.glm函数的调用。

2. `model`：最终选择的模型，它通常基于完整数据集（不是交叉验证的训练集）。

3. `k`：折数（folds）。

4. `lambda`：一个用于模型调整（模型选择）的正则化参数（regularization parameter），它可能是一个单独的值或是一个向量。当正则化策略为"LASSO"或"Elastic-Net"时，lambda值将设置一系列不同的值，即以产生一系列模型。

5. `cvm`：一个包含每一次交叉验证的错误平均值和标准误差（standard error）的交叉验证评估（evaluation）矩阵（matrix）。

6. `cvrisk`：一个包含了每个lambda值的平均验证误差（cross-validation risk）和标准误差。

7. `call1`：cv.glm函数在每个轮次的验证中所使用的参数和选项。

8. `dots`：其他参数和选项。

具体的返回值内容可能会因不同的算法或参数而有所不同。

在这个例子中，我们使用了`College`数据集，对该数据集的所有自变量拟合了一个线性回归模型。然后，使用`cv()`函数进行了10折交叉验证，并打印了交叉验证结果摘要信息，包括均方误差和R方值。需要注意的是，交叉验证的结果是一个含有多个元素的列表（list），其中的delta元素包含了交叉验证的结果信息。

需要特别注意的是，`cv()`函数需要被评估的模型或预测函数具有与常规预测函数相同的函数特性，即具有`predict()`函数的参数格式和返回值特性。如果需要使用自定义的预测函数进行交叉验证，则可以将其封装成满足这些特性的函数。

```{r cars}

library(ISLR2)

library(doBy)

# 手动计算loocv理解

# 第k个 作为验证集test，其余 𝑛 − 1 个数据作为训练 集train

for(k in 1:nrow(Auto)){

curr\_train=k

Auto\_train=Auto[-curr\_train,]

Auto\_test=Auto[curr\_train,]

for(i in 1:10){

# 这里的广义线性模型glm没有定义family，则默认为线性lm

glm1=glm(mpg~poly(horsepower,i,raw =TRUE),data=Auto\_train)

# poly是求多项式的函数，以第一个参数作为变量，逐级求x^1+x^2+ ```x^i结束

Auto\_test$pre=predict(glm1,Auto\_test)

# 计算均值差mean squared error

mse=mean((Auto\_test$pre-Auto\_test$mpg)^2)

if(i==1)mses=mse

else mses=c(mses,mse)

}

mses=data.frame(mses)

mses$dop=c(1:10)

mses$times=k

if(k==1)msess=mses

else msess=rbind(msess,mses)

}

msess$times=as.factor(as.character(msess$times))

# loocv函数计算

glm2=glm(mpg~poly(horsepower,3,raw =TRUE),data=Auto)

# $delta 包括了全部平方误差均值（Mean Square Error）和R方值（R-Squared）

loocv\_mse=cv.glm(Auto,glm2,K=5)$delta

#第一个参数data，导入数据集，第二个formula预测模型，第三个K交叉检验的折数

print(loocv\_mse)

```

## .广义线性函数glm

在R语言中，`glm()`函数可以用于拟合广义线性模型（GLM）。该函数的一般形式为：

```

glm(formula, family = gaussian, data, weights, subset, na.action, start = NULL, ...)

```

其中，各参数的含义为：

- `formula`：模型的公式，以响应变量和预测变量的关系表示。

- `family`：链接函数和误差分布函数的名称，控制拟合模型的性质，常用的包括：二项分布（binomial）、泊松分布（poisson）、高斯分布（gaussian）等。

- `data`：包含用于拟合模型的数据的数据框。

- `weights`：可选，用于赋予每个观察值不同的权重。

- `subset`：可选，包含用于拟合模型的子集的逻辑向量或整数向量。

- `na.action`：可选，指定在数据集中缺失值的处理方式。

- `start`：可选，参数的初始值。

- `...`：其他参数，用于控制模型拟合的细节。

需要注意的是，在上述参数中，`formula`是glm函数最重要的参数，它的格式为：“响应变量 ~ 预测变量1 + 预测变量2 + ...”，其中，响应变量和预测变量可以是离散变量或连续变量。

例如，假设我们在一个数据集中拟合一个二项分布的广义线性模型，可以使用以下代码：

```R

data <- read.csv("data.csv") # 加载数据集

fit <- glm(Response ~ Predictor1 + Predictor2, data = data, family = binomial) # 拟合二项分布的广义线性模型

summary(fit) # 查看模型的摘要信息

```

在这个例子中，我们使用`read.csv()`函数加载了一个数据集，并使用`glm()`函数拟合了一个二项分布的广义线性模型。需要特别注意的是，在这个例子中，需要自行替换`Response`和`Predictor1`、`Predictor2`等参数为具体的变量名。拟合完毕后，使用`summary()`函数可以查看模型的摘要信息，包括参数估计值、标准误、z值和p值等。

在实际应用中，还可以通过`predict()`函数对新数据进行预测，或通过绘制诊断图形来评估模型的合理性。需要注意的是，对于缺失值或异常值，需要进行适当的处理才能得到可靠的模型拟合结果。

## 决策树(Decision Trees)

1. 决策树是一种非参数(Non-parametric)算法。基于一定的规则，将数据集分割成更 小的子集，使得其响应变量值尽可能一致，且认为各子集的响应变量值是固定的. 最常见的决策树构建策略为Breiman提出的CART（Classification and Regression Tree）方法，binary classification。
2. 决策树模型预测结果基于：

（1）回归（Regression）：每个子集响应变量的平均值

（2）分类（Classification）：占据主导地位的类别(majority class)。

决策树（Decision Tree）是一种基于树形结构(通常树形结构遍历比线性的快，查找、插入、更改等更加方便)的机器学习算法，可以用于分类和回归问题。决策树的核心思想是将数据集递归地划分为不同的小区间，使得每个区间内的数据具有相似的特征或属性。在对新数据进行预测时，通过树形结构逐级判断其所属的区间，并对其进行分类或回归等预测。决策树算法的优势在于能够处理混合型数据、高维数据，且易于解释和理解。

以分类问题为例，决策树的构建过程通常包括以下几个步骤：

1. 确定根节点：选择一个最优的属性作为根节点，将数据集按照该属性的取值划分为不同的子集。

2. 递归地划分子节点：对于每个子集，选择最优的属性作为子节点，将其继续递归地划分为更小的子集，直至满足终止条件。

3. 构建决策树模型：根据划分结果生成树形结构，将每个节点所对应的属性和取值表示出来，形成一个决策树模型。

4. 对新样本进行分类：根据新样本的属性值，在决策树中逐级判断其所属的叶节点，并将其分类到对应的类别中。

在构建决策树时，通常需要使用一些指标来评估属性的优劣程度。常用的评估指标包括信息增益（Information Gain）、增益比（Gain Ratio）和基尼指数（Gini Index）等，其中信息增益是最常用的指标之一。信息增益的计算方法是通过熵（Entropy）来度量样本集合的纯度，即样本集合中包含的信息量的大小。指标越高表示属性的区分度越好，对分类任务的影响也越大。

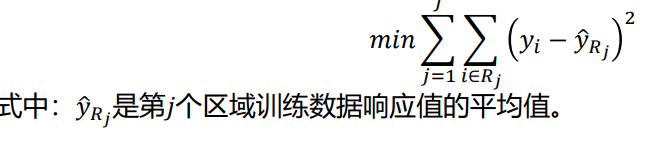
需要注意的是，决策树容易产生过度拟合现象，即过度依赖训练数据，而忽略了更普适的规律。在实际应用中，可以通过设置树的深度、剪枝等方式来避免该问题的出现。此外，决策树对异常值和噪声数据敏感，因此需要进行适当的预处理和数据清洗。

回归树步骤：

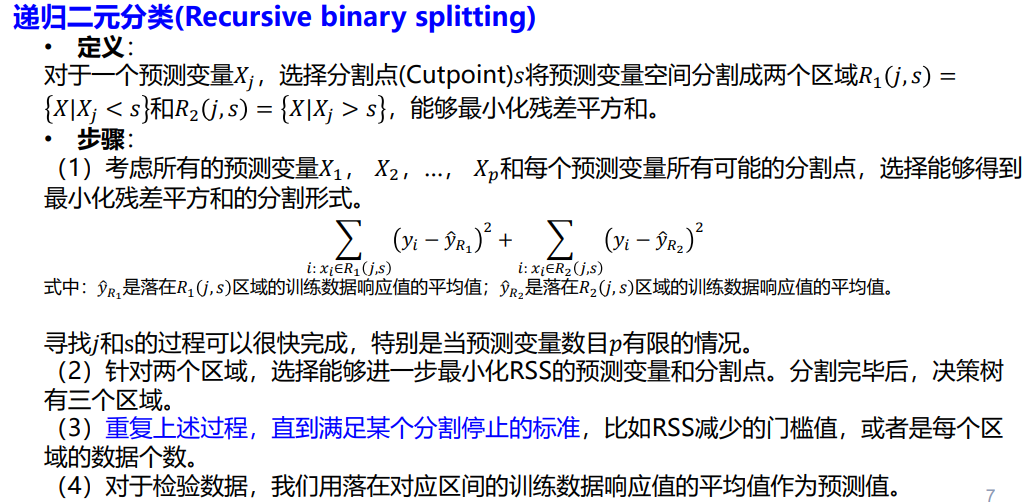
a. 将预测变量空间，也就是𝑋1， 𝑋2，…， 𝑋𝑝所有可能的值，划分为𝐽个不同且不重合 的区域𝑅1，𝑅2，…，𝑅𝐽。

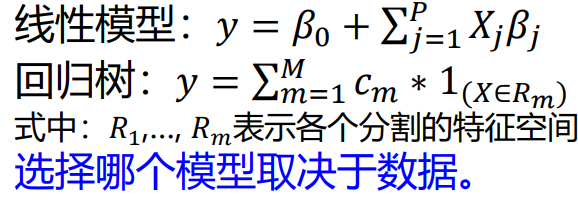
b. 对于落在区域𝑅𝑗的每个观测数据，我们对其响应值的预测值都是相同的，均为所有 落在区域𝑅𝑗的训练数据的响应值的平均值。

问题：如何来分割得到各区域？

方法：选择能够最小化残差平方和(RSS, residual sum of squares)的区域分割形式。

递归二元分类(Recursive binary splitting) •





1、在R语言中，已经有了两种不同的回归树算法实现：CART 和 C4.5。其中 CART 是 R 语言中内置的一个包，可以直接用 `tree` 函数进行调用，而 C4.5 可以使用 `party` 和 `rpart` 等多个包进行调用。下面我们就来看一下

`tree` 函数的一些主要参数及其含义。

- `formula`：表示目标变量 Y 与自变量 X 之间的关系式，即 Y ~ X1 + X2 +… ，其中 ~ 表示“是由”，而 + 则表示多条自变量。

- `data`：指定数据集。

- `subset`：指定数据集的子集。

- `weights`：指定样本权重。

- `na.action`：用于处理缺失值所采用的策略。

- `method`：为树的建立提供方法类型，并规定树的叶节点上所含值的表示类型。默认是 "anova"。此外还有 "exp" 和 "class" 两种选项。

- `split`：表示在树的压缩过程中，间隔出现的距离。默认为 "deviance"，可选的有 "misclass"、"error"、"gini"、"purity" 等。

- `minbucket`：表示最小的叶节点规模，即若节点中的样本数小于此规模，不再分裂。默认是 1 。

- `minsplit`：表示最少多少个样本才起始分裂的分割节点。默认是 20 。

- `maxdepth`：表示树的最大深度。默认为 -1，即无限制的向下增长。

- `xval`：表示使用多少折交叉验证来估计误差率。默认是 10 次。

以上是 `tree` 函数最常用的参数，其余的参数可以在函数调用时使用 `?tree` 命令进行查看。

2、在R语言的 `tree` 函数中，`method` 参数用于指定建立回归树的方法。`method` 有三个可选值，分别是 `"anova"`、`"exp"` 和 `"class"`。

1. （默认）`"anova"`：表示使用 ANOVA 方法构建回归树。ANOVA 指方差分析，这一方法的原理是利用自变量将因变量的方差分解成项的和。操作上，该方法会在每个节点处按照最小化节点和两个子分支之间的不同量进行相应的分割。

2. `"exp"`：表示使用指数回归树进行模型构建。这种模型常用于对 Y 的非负值进行预测。指数回归树的叶节点存储的是 Y 的对数值。

3. `"class"`：表示使用分类回归树方法构建模型。在模型建立时，会将 Y 转换成类别变量，构建分类树。它的分割标准是样本的类别纯度，一般使用 Gini 指数或信息熵计算。

在应用中，一般使用默认选项 `"anova"` 来构建回归树。当因变量是非负值时，使用 `"exp"` 得到的预测效果更好。如果要分类建树，建议使用 `"class"`。

3、`prune()`函数是 R 语言中 `tree` 包提供的一个函数，用于对建立好的决策树模型进行剪枝。剪枝是为了降低过拟合现象的发生。

`prune()` 函数的常用参数如下：

- `x`：需要剪枝的决策树模型。

- `newdata`：包含响应变量的测试数据集，用于评估剪枝的影响。如果测试数据集为 `NULL`，则使用原始数据集进行剪枝。

- `best`：逻辑值，是否显示最优子树。

- `method`：剪枝方法，可选 "deviance","misclass","risk"，默认是 "deviance"。

1. "deviance"：这是 C4.5 剪枝方法中最常用的一种。剪枝后的树在测试数据集上的对数似然（也称为偏差）应减少的最多。
2. "misclass"：这种剪枝方法中的终止准则是分类错误的比例。剪枝后的子树被预测分类错误的样本个数应尽量少。
3. "risk"：使用这一剪枝方法，会在子树的误差和复杂度之间进行权衡，并选择损失最小的子树。如果你想结合决策树的理论结果和你的数据集进行决策，这是一个非常有用的方法。

- `...`：其它的可选参数，如惩罚因子 k 和最优子树的叶子节点数。

例如，假设我们已经用 `tree()` 函数建立了一个决策树模型 `fit`，然后想通过剪枝来防止过拟合，我们可以使用以下方法：

```

# 构建原始的决策树

fit <- tree(y ~ x1 + x2 + x3, data=mydata)

# 剪枝

pruned\_fit <- prune(fit, newdata=mytest, best=TRUE)

# 显示最优子树

plot(pruned\_fit)

text(pruned\_fit, pretty=0)

```

经过剪枝后，我们可以选择显示最优子树直观地观察决策树的结构和性能。通常而言，剪枝后的决策树模型会比原始模型性能更好，是更为稳健的预测模型。

```{r }

library(tree)

library(ISLR2)

set.seed(1)

# 取样本子集subset

train=sample(1:nrow(Boston),nrow(Boston)/2)

# 调用tree决策树，预测模型formula = medv~. 取样本子集subset

tree\_boston=tree(medv~.,Boston,subset=train)

# 画出回归树

plot(tree\_boston)

text(tree\_boston,pretty=0)

# cv校验后看图得到最适子树best值，然后剪枝prune，画出剪枝后的回归树

cv\_boston=cv.tree(tree\_boston)

plot(cv\_boston$size,cv\_boston$dev,type="b")

# prune剪枝前的模型formula，新的数据newdata：包含响应变量的测试数据集，用于评估剪枝的影响。如果测试数据集为 NULL，则使用原始数据集进行剪枝。，最优子树best，剪枝方法method，，可选 “deviance”,“misclass”,“risk”，默认是 “deviance”。

prune\_boston=prune.tree(tree\_boston,best=5)

plot(prune\_boston)

text(prune\_boston)

# 回归树可以模型预测数据

y=predict(tree\_boston,newdata=Boston[-train,])

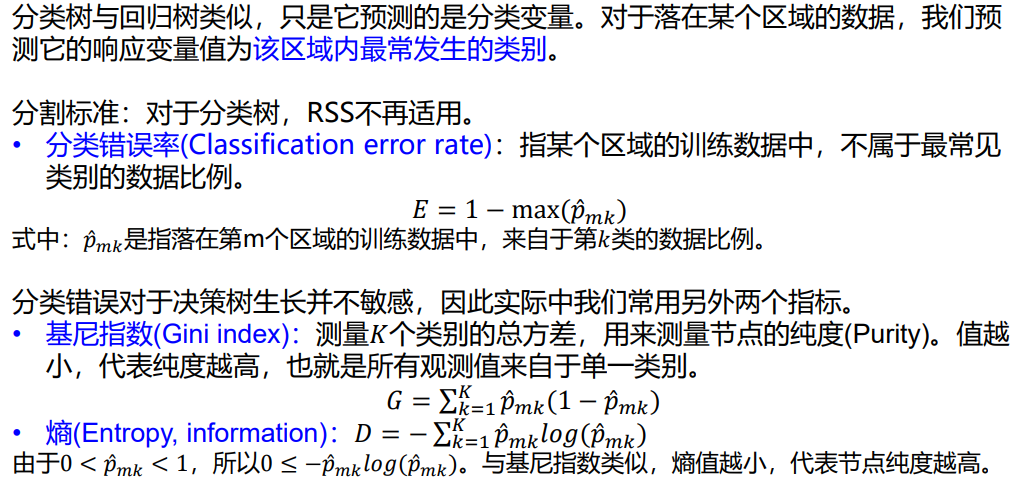
boston\_test=Boston[-train,"medv"]

plot(y,boston\_test)

abline(0,1)

print(mean(y-boston\_test)^2)

```



在 R 语言中，使用 `cv.tree()` 函数对决策树模型进行交叉验证。`cv.tree()` 函数的主要参数如下：

- `tree`: 构建的分类或回归树。

- `FUN`: 交叉验证迭代需要进行的函数，取值为内建求平均数的 `mean` 函数或 `NaiveBayes`，这取决于数据的特征。

- `K`: 交叉验证的折数。

- `deviance`: 交叉验证的误差类型，用来衡量交叉验证误差的度量，有 `"tree"` 和 `"model"` 两种，分别表示在每个节点上拆分或完整树。

- `y`: 响应变量。

- `X`: 预测变量。

- `keep`: 返回交叉验证结果的哪些信息，可以是`"tree"`, `"model"`, `"alpha"`, `"call"`, `"k"`, `"dev"`

其中，

- `"tree"`表示选择的树。

- `"model"`表示选择的树的模型。

- `"alpha"`表示损失指数呈现的 alpha 值。

- `"call"`表示函数运行调用。

- `"k"`表示发生和使用的折数。

- `"dev"`表示验证折得到的误差。

需要注意的是，不同的参数组合会对最终的交叉验证结果产生影响，需要根据具体的任务需求，对交叉验证函数进行相应的调整。

cv.tree是一个用于在决策树模型中进行交叉验证的函数。它可以帮助找到最佳的树的复杂度参数，以便获得最好的模型表现。

在cv.tree的函数定义中，FUN参数是一个可选参数，用于指定一个要在每一次交叉验证中对数据进行拟合和预测的函数。默认值为rpart函数，这是一个用于创建回归树模型的函数。

如果您使用cv.tree来构建分类树模型，您可以将FUN参数设置为其他相应的分类树模型函数，如ctree、rpart.control等。绝大多数情况下，您不需要改变这个默认值，因为rpart函数可以满足大多数需求。

prune.misclass是一个用于决策树模型剪枝的函数，它基于分类误差最小化的原则进行决策树剪枝。该函数可以根据交叉验证误差最小化原则，找到最优的决策树模型。

在使用prune.misclass进行决策树剪枝时，模型会将树划分为两部分：剪枝前的树和剪枝后的树。然后使用交叉验证方法来确定哪些节点应该被剪枝，以最小化模型的误差。

prune.misclass函数的参数包括一个剪枝前的树对象、一个测试集数据对象以及一个选择出最小交叉验证误差的标准。如果使用默认值，该函数将根据1-SE规则选择最优子树。

需要注意的是，prune.misclass函数只适用于分类问题的决策树剪枝。如果您想要进行回归问题的决策树剪枝，请使用prune函数并将误差衡量指标设置为mse。

```{r }

library(tree)

library(ISLR2)

set.seed(2)

# 取样本子集subset

Carseats$High=factor(ifelse(Carseats$Sales<=8,"No","Yes"))

train=sample(1:nrow(Carseats),200)

train\_Carseats=Carseats[train,]

test\_Carseats=Carseats[-train,]

tree\_Carseats=tree(High~.-Sales,data=train\_Carseats)

test\_Carseats$pre=predict(tree\_Carseats,newdata=test\_Carseats,type="class")

# table(test\_Carseats$pre,test\_Carseats$High)

# 在cv.tree的函数定义中，FUN参数是一个可选参数，用于指定一个要在每一次交叉验证中对数据进行拟合和预测的函数。默认值为rpart函数，这是一个用于创建回归树模型的函数。

# 如果您使用cv.tree来构建分类树模型，您可以将FUN参数设置为其他相应的分类树模型函数，如prune.misclass ，ctree、

cv\_Carseats=cv.tree(tree\_Carseats,FUN=prune.misclass)

plot(cv\_Carseats$size,cv\_Carseats$dev,type="b")

# prune.misclass函数只适用于分类问题的决策树剪枝,在使用prune.misclass进行决策树剪枝时，模型会将树划分为两部分：剪枝前的树和剪枝后的树。然后使用交叉验证方法来确定哪些节点应该被剪枝，以最小化模型的误差。

prune\_Carseats=prune.misclass(tree\_Carseats,best=9)

plot(prune\_Carseats)

text(prune\_Carseats)

```

## Predict:

在 R 语言中，使用 `predict()` 函数可以对给定的未知数据进行预测。 `predict()` 函数常常与已经拟合好的模型结合使用，以对未知数据作出预测。

`predict()` 函数的常用参数如下：

- `object`：需要预测的模型对象。

- `newdata`：需要预测的未知数据集。该参数对应于模型中的测试数据集。

- `type`：字符串，表示模型预测的类型。针对不同类型的模型，type 参数默认有不同的取值。

- `...`：其它可选参数。

例如，对于回归树模型，可以使用以下方法对新数据进行预测：

```

# 建立回归树模型

fit <- tree(y ~ x1 + x2 + x3, data=mydata)

# 对新数据进行预测

predict(fit, newdata=mynewdata)

```

这里，`fit` 为已经拟合好的模型对象，`newdata` 为新的待预测数据集。

需要注意的是，对于不同类型的模型对象，`predict()` 函数的输出结果也是不同的，比如，在逻辑回归模型中，`predict()` 函数的输出结果是分组变量的概率。因此，需要根据具体的模型类型，选择正确的 `predict()` 函数输出结果，以获得最准确的预测结果。

在 R 语言中的 `predict()` 函数中，`type` 参数用于指定模型预测的类型。`type` 的取值与不同的模型类型相关，这里主要介绍一下针对回归模型和分类模型的 `predict()` 函数的 `type` 参数。

1. 回归模型：

- `type="response"`： 预测的是类别，返回的是预测类别的概率、频率和交叉验证预测。

- `type="vector"`： 预测一个或多个连续型变量的值。

- `type="prob"`： 预测类别的概率或频率。

2. 分类模型：

- `type="class"`： 预测后的分类结果是一个因子，就是各个分类的名称。

- `type="raw"`： 预测后的分类结果是一个矩阵，对应于测试集中的每一个样本。

- `type="probs"`： 预测结果是一个矩阵，展示了每个样本属于不同类别的条件概率。

需要注意的是，不同的模型类型对应于不同的预测类型，因此在使用 `predict()` 函数时，需要先根据具体的问题，选择正确的 `type` 参数。

## 聚类

非监督学习是机器学习中的一种方法，与监督学习相对应。在非监督学习中，数据没有带有目标变量的标签或类别信息。这意味着，非监督学习算法不需要通过已有的标签或类别信息来训练模型，而是试图从数据集本身中发掘数据内在的模式、结构或规律。

非监督学习通常用于探索性数据分析、数据降维、聚类、异常检测等应用场景中。聚类是一种常见的非监督学习方法，它试图将具有相似特征的数据点分组到同一类中，使得同一类内的数据相互之间具有较高的相似性，而不同类之间的数据则具有较低的相似性。

常见的非监督学习算法包括k均值聚类(kmean)、层次聚类、主成分分析（PCA）、因子分析、独立成分分析（ICA）、自编码器（Autoencoder）等。

K-means聚类：将数据划分为我们预先设定的类别个数

步骤：

（1）确定类别数目：没有确定 方法，可以通过尝试得到

（2）将每一个观测数据划分到 某一个类别中

如何确定分类个数k值

```{r }

set.seed(2)

x=matrix(rnorm(50\*2),ncol=2)

# par(mfrow=c(1,2))

x[1:25,1]=x[1:25,1]+3

x[1:25,2]=x[1:25,2]-4

for(i in 1:10){

km=kmeans(x,centers = i,nstart = 20)

print(km)

print(table(km$type,km$cluster))#数据种类及个数

plot(x,col=(km$cluster+1),main=i,pch=20,cex=2)#好像data是二维的可以直接画图，更高维的会画很多个

#

}

```

要使用R语言中的nbclust包确定K-means的最优聚类数量，可以按照以下步骤进行操作：

1. 首先，安装nbclust包。通过执行以下命令进行安装：

```{r}

install.packages("nbclust")

```

2. 然后，按照下面的模板来运行代码，进行聚类分析。其中，data是数据集，cluster是聚类的结果，和kmeans函数的参数一样，index是指标，method是距离计算方法，distance是相应的距离矩阵。

```{r}

library(nbclust)

set.seed(1234)

clus <- kmeans(data, centers=4, nstart=25)

result <- NbClust(data, min.nc=2, max.nc=10, method="kmeans", distance="euclidean", index="all")

```

其中，参数说明如下：

- data：进行聚类分析的数据集。

- centers：聚类的数量。

- nstart：初始点的数量。

- min.nc：最小的聚类数量。

- max.nc：最大的聚类数量。

- method：聚类分析的方法，默认是"kmeans"。

- distance：计算相似度的方法，默认是"euclidean"。

- index：当需要进行多种指标评估时，可以用"all"表示同时查看所有指标的结果。

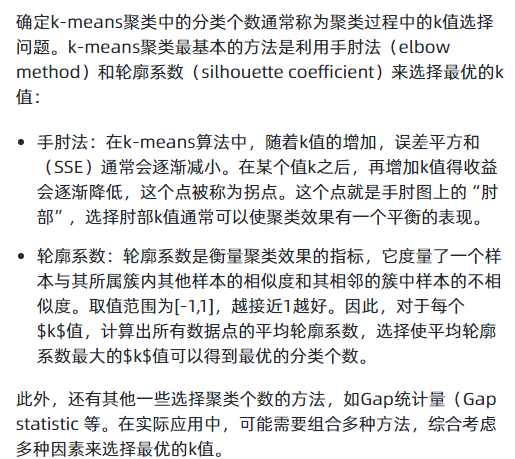
3. 最后，可以通过result$Best.nc来确定最优的聚类数量。比如：

```{r}

result$Best.nc

```

这里的输出是一个数字，表示最优的聚类数量。

```

Cluster means:聚类均值

[,1] [,2]

1 3.3339737 -4.0761910

2 -0.1956978 -0.1848774

Clustering vector: 聚类向量

[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

[43] 2 2 2 2 2 2 2 2

Within cluster sum of squares by cluster: 聚类内按聚类的平方和

[1] 63.20595 65.40068

(between\_SS / total\_SS = 72.8 %)

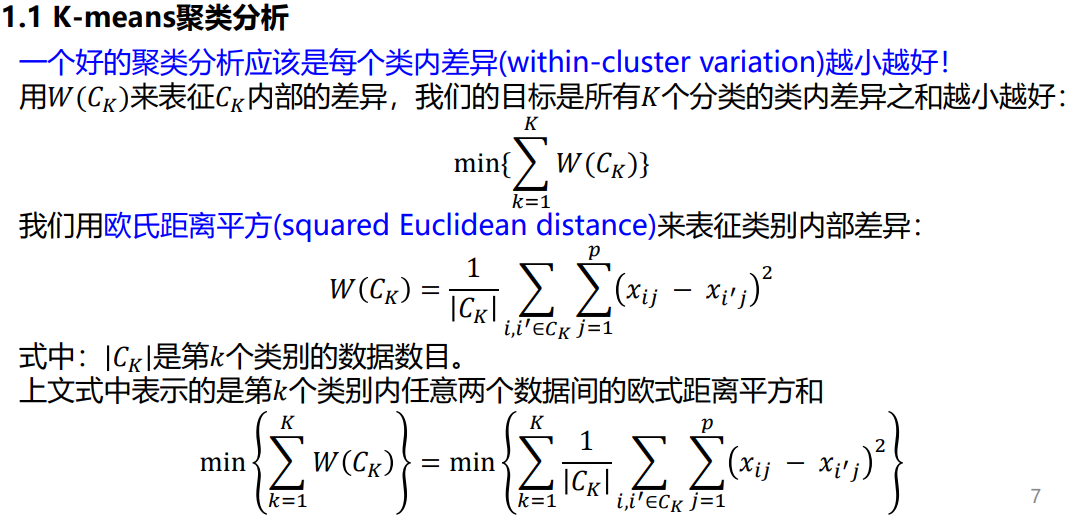
Available components:

[1] "cluster" "centers" "totss" "withinss" "tot.withinss"

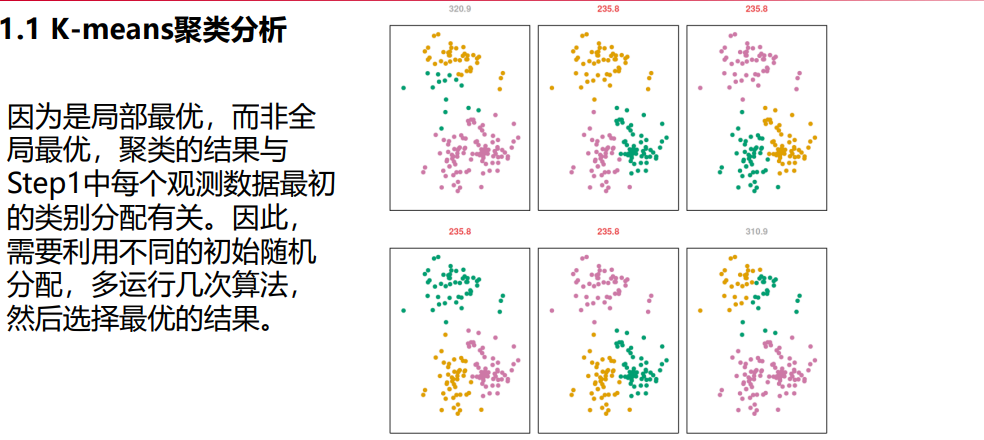
[6] "betweenss" "size" "iter" "ifault"

K-means clustering with 3 clusters of sizes 23, 10, 17

具有 3 个大小为 23、10、17 的聚类的 K 均值聚类



如何划分数据？有一个简单的找到局部最优(local optimum)的方法。



改变nstart，初始随机分配次数

在R语言中，scale函数常用于数据标准化（data scaling）操作，即将数据按照各个特征的均值和标准差进行变换，使得每个特征具有相同的数据范围和单位。

类似正态分布的标准化，mean=0，var=1

scale函数的语法如下：

```R

scaled\_data <- scale(x, center = TRUE, scale = TRUE)

```

其中，x表示要进行标准化操作的数据；center参数（缺省值为TRUE）表示是否对数据进行中心化，即减去各个特征的均值；scale参数（缺省值为TRUE）表示是否对中心化后的数据进行缩放，即除以各个特征的标准差。

例如，对于一个数据矩阵data，我们可以对其进行标准化操作：

```R

scaled\_data <- scale(data)

```

这将构造一个新的数据矩阵scaled\_data，该矩阵每个特征都被标准化。可以使用summary函数验证标准化结果：

```R

summary(scaled\_data)

```

这将显示每个特征的均值和标准差，可以验证每个特征已经被标准化为均值为0，标准差为1的分布。

```{r }

library(rattle)

set.seed(402)

par(mfrow=c(1,2))

km=kmeans(wine[-1],3,nstart = 25)

print(table(wine$Type,km$cluster))#列出一维数据的聚类结果，画出频数直方图

hist(km$cluster,xlab="Cluster", ylab="Frequency",col="steelblue")

wine\_scale=scale(wine[-1])#数据标准化后，聚类效果更好

km=kmeans(wine\_scale,3,nstart =25)

print(table(wine$Type,km$cluster))

hist(km$cluster,xlab="Cluster", ylab="Frequency",col="skyblue")

```