# 第八章

**1. 请分析MASS包中的Boston数据集**

**(1) 利用LASSO、MCP和SCAD3种惩罚方法分析找出影响房价medv的因素，比较一下这些方法找出的影响因素。**

**(2) 比较LASSO方法和逐步回归方法筛选出来的结果。**

解：

先将Boston数据集分为自变量和因变量：

# 导入所需的包

library(MASS)

library(glmnet)

# 加载Boston数据集

data(Boston)

# 将数据集分为自变量和因变量

X <- as.matrix(Boston[, -14]) # 自变量

y <- Boston$medv # 因变量

1. LASSO：

# LASSO

lasso\_model <- cv.glmnet(X, y, family="gaussian")

lasso\_coef <- coef(lasso\_model)

lasso\_coef

resid1 <- (X %\*% lasso\_coef [ -1 ] + lasso\_coef [ 1 ] - y)

MSE1 <- sum (resid1 ^ 2)

MSE1

输出结果：

> lasso\_coef

14 x 1 sparse Matrix of class "dgCMatrix"

s1

(Intercept) 17.471486288

crim -0.021826808

zn .

indus .

chas 1.897718337

nox -3.341293717

rm 4.266279454

age .

dis -0.317122209

rad .

tax .

ptratio -0.787265492

black 0.006558204

lstat -0.518129464

> MSE1

[1] 12633.63

通过比较输出结果可发现，zn、indus、age、rad和tax均被移除，留下crim、chas、nox、rm、dis、ptratio、black、lstat8项。

②MCP：

# MCP

fit2 <- cv.ncvreg(X, y, family="gaussian")

fit.mcp <- fit2$fit

beta.fit2 <- fit.mcp$beta [ , fit2$min]

round( beta.fit2 , 3)

resid2 <- (X %\*% beta.fit2 [ -1 ] + beta.fit2 [ 1 ] - y)

MSE2 <- sum (resid2 ^ 2)

MSE2

得到如下的输出结果：

> round( beta.fit2 , 3)

(Intercept) crim zn indus chas

36.351 -0.108 0.046 0.000 2.719

nox rm age dis rad

-17.384 3.801 0.000 -1.493 0.300

tax ptratio black lstat

-0.012 -0.947 0.009 -0.523

> MSE2

[1] 11081.36

通过比较输出结果可发现， indus、age系数为0，均可移除，留下crim、zn、chas、nox、rm、dis、rad、tax、ptratio、black、lstat11项。

③SCAD：

# SCAD

fit3 <- cv.ncvreg(X, y, family="gaussian",penalty="SCAD")

fit.scad <- fit3$fit

beta.fit3 <- fit.scad$beta[, fit3$min]

round(beta.fit3, 3)

resid3 <- (X %\*% beta.fit3 [ -1 ] + beta.fit3 [ 1 ] - y)

MSE3 <- sum (resid3 ^ 2)

MSE3

输出结果：

> round(beta.fit3, 3)

(Intercept) crim zn indus chas

36.346 -0.108 0.046 0.000 2.719

nox rm age dis rad

-17.380 3.802 0.000 -1.493 0.300

tax ptratio black lstat

-0.012 -0.947 0.009 -0.523

> MSE3

[1] 11081.36

通过比较输出结果可发现， indus、age系数为0，均可移除，留下crim、zn、chas、nox、rm、dis、rad、tax、ptratio、black、lstat11项。

总的来说，LASSO模型中zn、indus、age、rad和tax均被移除，而MCP和SCAD均只移除了indus、age变量。通过比较MSE，发现MCP和SCAD的MSE相同，且均比LASSO模型的MSE小，所以MCP和SCAD的估计效果都比LASSO好，且两者相近。

（2）LASSO方法的结果在上题已有，逐步回归的代码：

# 逐步回归

stepwise\_model <- step(lm(y ~ ., data = as.data.frame(X)), direction = "both")

stepwise\_model

MSE4 <- sum(stepwise\_model$residuals^2)

MSE4

结果：

lm(formula = y ~ crim + zn + chas + nox + rm + dis + rad + tax +

ptratio + black + lstat, data = as.data.frame(X))

Coefficients:

(Intercept) crim zn chas

36.341145 -0.108413 0.045845 2.718716

nox rm dis rad

-17.376023 3.801579 -1.492711 0.299608

tax ptratio black lstat

-0.011778 -0.946525 0.009291 -0.522553

> MSE4

[1] 11081.36

逐步回归中剔除了indus和age变量，MSE也小于LASSO模型，所以逐步回归模型同样比LASSO模型更优。

**2. 请分析ISLR包中的Smarket数据集。以Direction为因变量，请用LASSO、MCP和SCAD3种惩罚方法分析找出影响股票交割涨跌方向的因素，并比较3种方法找出的影响因素是否一样。**

解：

本题自变量Direction为字符串类型“Up”和“Down”，故将其转换成1和-1表示方向：

# 导入所需的包

library(glmnet)

library(ncvreg)

library(ISLR)

# 加载Boston数据集

data(Smarket)

# 将数据集分为自变量和因变量

X <- as.matrix(Smarket[, -9]) # 自变量

y\_string <- Smarket$Direction

y <- ifelse(y\_string == "Up", 1, -1) # 转换

① LASSO

# LASSO

lasso\_model <- cv.glmnet(X, y, family="gaussian")

lasso\_coef <- coef(lasso\_model)

lasso\_coef

resid1 <- (X %\*% lasso\_coef [ -1 ] + lasso\_coef [ 1 ] - y)

MSE1 <- sum (resid1 ^ 2)

MSE1

输出结果：

> lasso\_coef

9 x 1 sparse Matrix of class "dgCMatrix"

s1

(Intercept) 0.03496241

Year .

Lag1 .

Lag2 .

Lag3 .

Lag4 .

Lag5 .

Volume .

Today 0.58551662

> MSE1

[1] 587.3379

可见LASSO回归只留下了Today变量，其余均被剔除，MSE为587.3379

② MCP：

# MCP

fit2 <- cv.ncvreg(X, y, family="gaussian")

fit.mcp <- fit2$fit

beta.fit2 <- fit.mcp$beta [ , fit2$min]

round( beta.fit2 , 3)

resid2 <- (X %\*% beta.fit2 [ -1 ] + beta.fit2 [ 1 ] - y)

MSE2 <- sum (resid2 ^ 2)

MSE2

输出结果：

> round( beta.fit2 , 3)

(Intercept) Year Lag1 Lag2 Lag3

-39.909 0.020 0.000 0.000 0.000

Lag4 Lag5 Volume Today

0.000 0.000 0.000 0.642

> MSE2

[1] 579.3507

可见MCP也只选择了Today，MSE为579.3507

③ SCAD：

# SCAD

fit3 <- cv.ncvreg(X, y, family="gaussian",penalty="SCAD")

fit.scad <- fit3$fit

beta.fit3 <- fit.scad$beta[, fit3$min]

round(beta.fit3, 3)

resid3 <- (X %\*% beta.fit3 [ -1 ] + beta.fit3 [ 1 ] - y)

MSE3 <- sum (resid3 ^ 2)

MSE3

输出结果：

> round(beta.fit3, 3)

(Intercept) Year Lag1 Lag2 Lag3

-58.260 0.029 -0.002 0.000 0.000

Lag4 Lag5 Volume Today

0.000 0.009 0.000 0.642

> MSE3

[1] 578.0169

可见SCAD选择了Year、Lag1、Lag5和Today，其MSE为578.0169。

总的来说，LASSO和MCP选择了同样的影响因素，但MCP的MSE较低，SCAD较它俩的MSE更低，且选择的变量多，故SCAD模型更优。

**3. 请模拟生成，由多元正态分布产生，p=100，n=100，对应的相关系数是, =0.1、0.5、0.9，回归系数=(1,1,1,1,1,0.5,0.5,0.5,0.5,0.5,0,…,0), 随机扰动项是标准正态分布，请模拟100次，分别用LASSO、MCP和SCAD筛选变量，比较变量筛选的FNR和FDR。**

解：

代码：

library(glmnet)

library(ncvreg)

set.seed(114514)

# 模拟参数

p <- 100

n <- 100

rho <- c(0.1, 0.5, 0.9)

beta <- c(rep(1, 5), rep(0.5, 5), rep(0, p-10))

num\_simulations <- 100

# 初始化结果向量

fnr\_lasso <- fnr\_mcp <- fnr\_scad <- rep(0, length(rho))

fdr\_lasso <- fdr\_mcp <- fdr\_scad <- rep(0, length(rho))

for (sim in 1:num\_simulations) {

for (r in 1:length(rho)) {

# 生成相关矩阵

Sigma <- matrix(0, nrow = p, ncol = p)

for (i in 1:p) {

for (j in 1:p) {

Sigma[i, j] <- rho[r]^(abs(i-j))

}

}

# 生成数据

X <- MASS::mvrnorm(n, rep(0, p), Sigma)

y <- X %\*% beta + rnorm(n)

# LASSO回归

fit\_lasso <- cv.glmnet(X, y, alpha = 1)

coef\_lasso <- coef(fit\_lasso, s = "lambda.min")

selected\_vars\_lasso <- which(coef\_lasso != 0)

# MCP回归

fit\_mcp <- cv.ncvreg(X, y, family="gaussian", penalty = "MCP")

coef\_mcp <- coef(fit\_mcp)

selected\_vars\_mcp <- which(coef\_mcp != 0)

# SCAD回归

fit\_scad <- cv.ncvreg(X, y, family="gaussian", penalty = "SCAD")

coef\_scad <- coef(fit\_scad)

selected\_vars\_scad <- which(coef\_scad != 0)

# 计算FNR和FDR

true\_vars <- which(beta != 0)

fnr\_lasso[r] <- fnr\_lasso[r] + sum(!(true\_vars %in% selected\_vars\_lasso)) / length(true\_vars)

fdr\_lasso[r] <- fdr\_lasso[r] + sum(!(selected\_vars\_lasso %in% true\_vars)) / length(selected\_vars\_lasso)

fnr\_mcp[r] <- fnr\_mcp[r] + sum(!(true\_vars %in% selected\_vars\_mcp)) / length(true\_vars)

fdr\_mcp[r] <- fdr\_mcp[r] + sum(!(selected\_vars\_mcp %in% true\_vars)) / length(selected\_vars\_mcp)

fnr\_scad[r] <- fnr\_scad[r] + sum(!(true\_vars %in% selected\_vars\_scad)) / length(true\_vars)

fdr\_scad[r] <- fdr\_scad[r] + sum(!(selected\_vars\_scad %in% true\_vars)) / length(selected\_vars\_scad)

}

}

# 计算平均值

fnr\_lasso <- fnr\_lasso / num\_simulations

fdr\_lasso <- fdr\_lasso / num\_simulations

fnr\_mcp <- fnr\_mcp / num\_simulations

fdr\_mcp <- fdr\_mcp / num\_simulations

fnr\_scad <- fnr\_scad / num\_simulations

fdr\_scad <- fdr\_scad / num\_simulations

# 输出结果

result <- data.frame(rho, fnr\_lasso, fdr\_lasso, fnr\_mcp, fdr\_mcp, fnr\_scad, fdr\_scad)

print(result)

输出结果：

> print(result)

rho fnr\_lasso fdr\_lasso fnr\_mcp fdr\_mcp fnr\_scad

1 0.1 0.001 0.6694312 0.008 0.34246268 0.003

2 0.5 0.000 0.5314077 0.085 0.32379577 0.056

3 0.9 0.028 0.3790730 0.566 0.09647619 0.568

fdr\_scad

1 0.5214510

2 0.4387296

3 0.1848373

可见，固定rho时，lasso的fnr都小于mcp和scad，dfr都大于mcp和scad。当rho逐渐增大，即扰动项增大时lasso的fnr较为平稳都接近0，fdr逐渐减小；而mcp的fnr逐渐增大，且当rho=0.9时较为明显，达到0.566，fdr逐渐下降；scad的fnr也逐渐上升，fdr逐渐下降；

# 第九章

1. 请分析ISLR包中的OJ（橘汁销售）数据集。该数据集包含了1070个顾客购买橙汁的记录。

（1） 用summary（）函数查看数据基本信息，按7：3的比例划分训练集和测试集。

# 导入ISLR包

library(ISLR)

# 查看数据基本信息

summary(OJ)

# 设置随机种子以确保可重复性

set.seed(2333)

# 划分训练集和测试集

train\_index <- sample(1:nrow(OJ), 0.7 \* nrow(OJ))

train\_data <- OJ[train\_index, ]

test\_data <- OJ[-train\_index, ]

（2） 将purchase作为响应变量，其余变量作为预测变量，对训练集建立一棵树。用summary函数查看树的输出信息、训练错误率及树的终端节点个数分别是多少。

# 导入tree包

library(tree)

# 建立一棵树

tree\_model <- tree(Purchase ~ ., data = train\_data)

# 查看树的输出信息

summary(tree\_model)

输出：

Classification tree:

tree(formula = Purchase ~ ., data = train\_data)

Variables actually used in tree construction:

[1] "LoyalCH" "PriceDiff" "StoreID" "ListPriceDiff"

Number of terminal nodes: 8

Residual mean deviance: 0.7527 = 557.7 / 741

Misclassification error rate: 0.1749 = 131 / 749

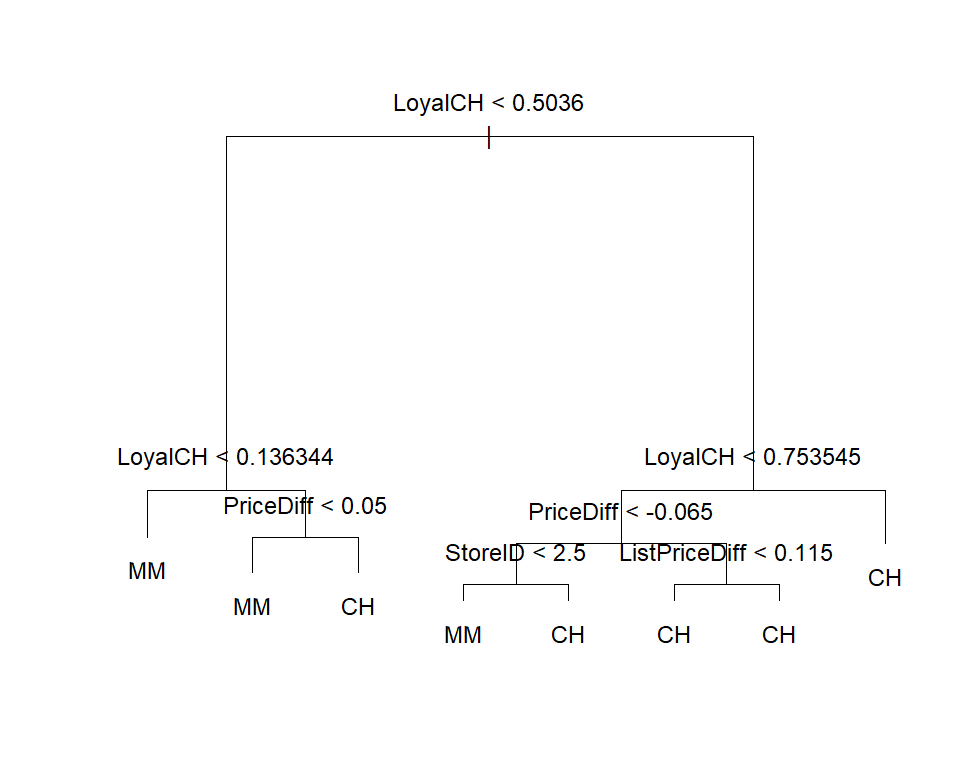
可见训练错误率约为0.1749，树的终端节点个数为8.

（3）画出所建立的树并解释结果

plot(tree\_model)

text(tree\_model, pretty=0)

输出图像：



（4）预测测试数据的响应值，并计算测试错误率

# 预测测试数据的响应值

test\_pred <- predict(tree\_model, test\_data,type="class")

# 计算测试错误率

table(test\_pred, test\_data $ Purchase)

输出结果：

test\_pred CH MM

CH 176 55

MM 13 77

> (176+77)/(176+55+13+77)

[1] 0.788162

结果显示，该模型预测准确率约为78.82%

（5）对树进行剪枝，用cv.tree()函数在训练集上确定最优的树。并画出错误率对size的函数。则建立一棵含5个终端节点的树。

# 进行树的剪枝

cv\_tree <- cv.tree(tree\_model,FUN = prune.misclass)

cv\_tree

# 画出错误率对size的函数图形

plot(cv\_tree$size, cv\_tree$dev, type = "b", xlab = "Size", ylab = "Deviance")

输出结果：

$size

[1] 8 7 5 4 2 1

$dev

[1] 141 140 160 164 165 285

$k

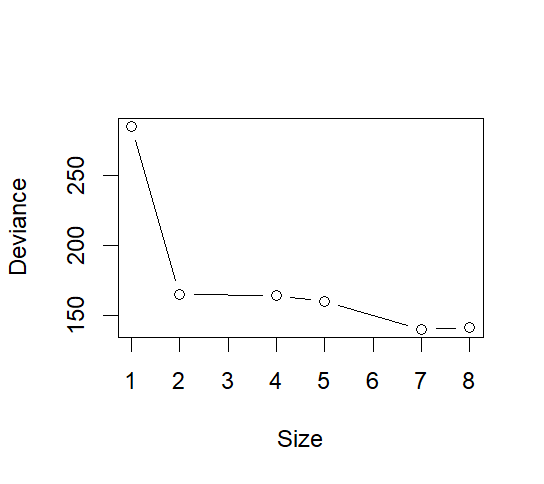
[1] -Inf 0 3 4 5 134

$method

[1] "misclass"

attr(,"class")

[1] "prune" "tree.sequence"



从上述数据和图像发现，当终端结点为7时，错误率最低，误差为140。

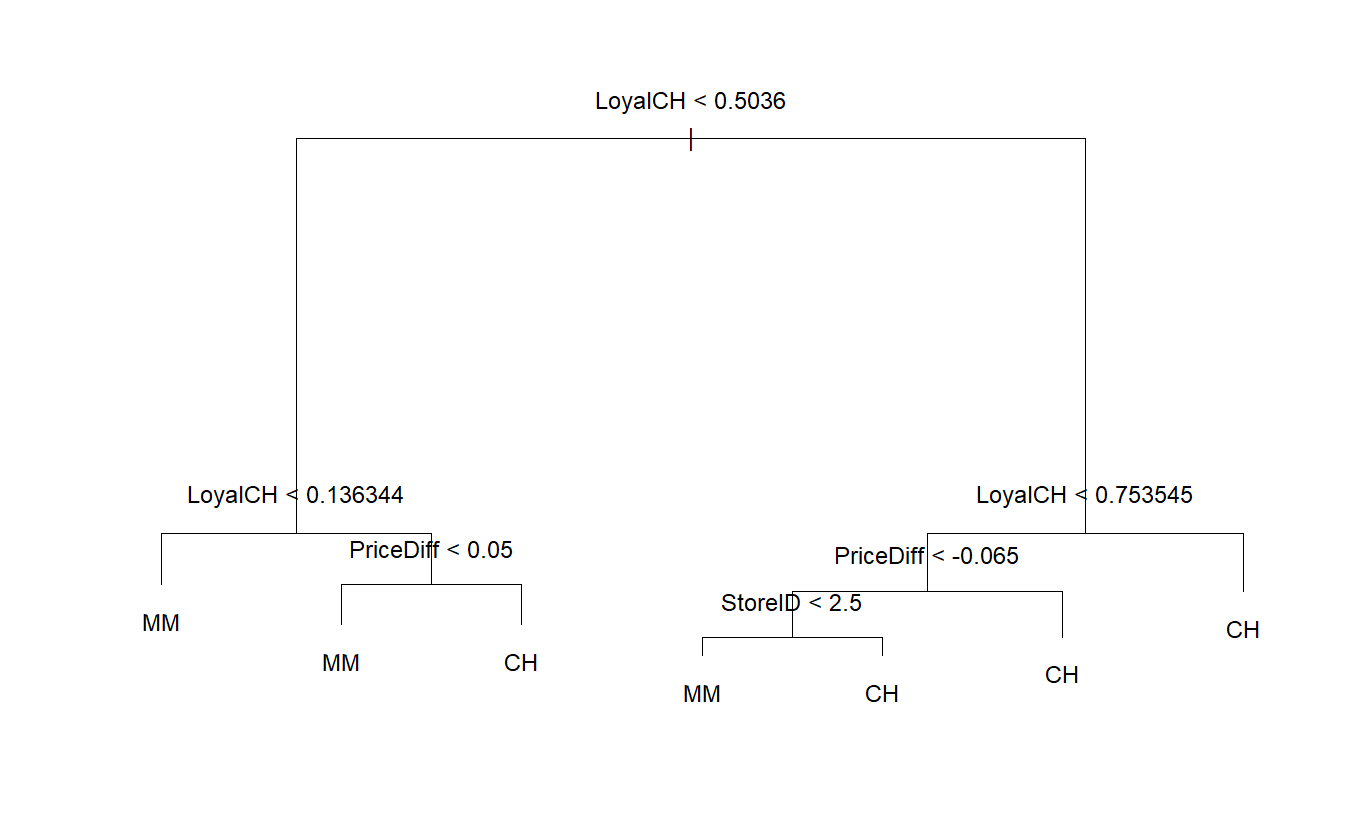
（6）用交叉验证得出的结果，生成经剪枝的树。如果交叉验证无法对剪枝之后的树进行选择，则建立一棵含5个终端节点的树。

# 确定最优的树

pruned\_tree <- prune.misclass(tree\_model, best = 7)

plot(pruned\_tree)

text(pruned\_tree, pretty=0)



输出结果：

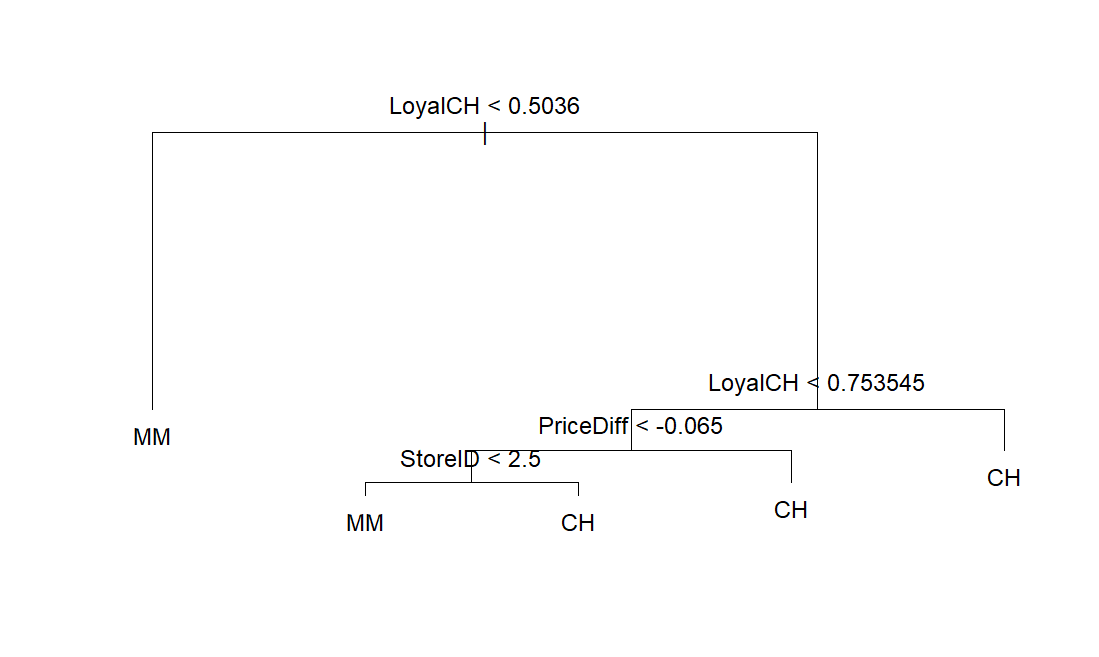
cv.tree.pred CH MM

CH 176 55

MM 13 77

> (176+77)/(176+55+13+77)

[1] 0.788162

发现准确率并没有提高，说明交叉验证并没有完成剪枝的选择，所以依照题目建立一棵含5个终端节点的树：  


输出结果：

cv.tree.pred CH MM

CH 151 23

MM 38 109

> (151+109)/(151+23+38+109)

[1] 0.8099688

发现准确率变成了0.81，提升了准确率。

（7）比较剪枝前后的训练错误率和测试错误率哪个更低。

# 剪枝前预测训练、测试数据的响应值

train\_pred <- predict(tree\_model, train\_data, type = "class")

test\_pred <- predict(tree\_model, test\_data,type="class")

# 剪枝前计算训练、测试错误率

table(train\_pred, train\_data $ Purchase)

table(test\_pred, test\_data $ Purchase)

#剪枝后训练、测试错误率

cv.train.pred <-predict(prune\_tree, train\_data, type = "class")

cv.test.pred <- predict(prune\_tree, test\_data, type="class")

table(cv.train.pred, train\_data$Purchase)

table(cv.test.pred, test\_data$Purchase)

输出结果：

> table(train\_pred, train\_data $ Purchase)

train\_pred CH MM

CH 437 104

MM 27 181

> (437+181)/(437+27+181+104)

[1] 0.8251001

> table(test\_pred, test\_data $ Purchase)

test\_pred CH MM

CH 176 55

MM 13 77

> (176+77)/(176+55+13+77)

[1] 0.788162

> table(cv.train.pred, train\_data$Purchase)

cv.train.pred CH MM

CH 365 38

MM 99 247

> (365+247)/(365+247+38+99)

[1] 0.8170895

> table(cv.test.pred, test\_data$Purchase)

cv.test.pred CH MM

CH 151 23

MM 38 109

> (151+109)/(151+23+38+109)

[1] 0.8099688

可以看出，剪枝前的训练错误率更低，测试错误率更高。

2. 请分析ISLR包中的Default数据集：p=100，n=100.

（1）请将数据集按照6：4的比例划分训练集和测试集，请分别利用决策树、随机森林、Adaboost、XGboost和logistic回归对训练集构建模型，利用测试集对所构建的模型进行测试。比较这几个模型的训练集预测准确率及测试集的预测准确率。

①导入包和划分数据集：

# 导入ISLR包

library(ISLR)

# 设置随机种子以确保可重复性

set.seed(233)

# 加载所需的包

library(tree)

library(randomForest)

library(gbm)

library(xgboost)

library(pROC)

# 加载Default数据集

data(Default)

# 划分训练集和测试集

train\_index <- sample(1:nrow(Default), 0.6 \* nrow(Default))

train\_data <- Default[train\_index, ]

test\_data <- Default[-train\_index, ]

1. 决策树：

# 决策树模型

tree\_model <- tree(default ~ ., data = train\_data)

tree\_train\_pred <- predict(tree\_model, train\_data,type="class")

tree\_test\_pred <- predict(tree\_model, test\_data,type="class")

table(tree\_train\_pred, train\_data$default)

table(tree\_test\_pred, test\_data$default)

输出结果：

tree\_train\_pred No Yes

No 5759 123

Yes 37 81

> (5759+81)/(5759+123+37+81)

[1] 0.9733333

> table(tree\_test\_pred, test\_data$default)

tree\_test\_pred No Yes

No 3848 87

Yes 23 42

> (3848+42)/(3848+87+23+42)

[1] 0.9725

可见决策树的训练准确率为0.9733，测试准确率为0.9725

③随机森林

# 随机森林模型

rf\_model <- randomForest(default ~ ., data = train\_data, importance=TRUE)

rf\_train\_pred <- predict(rf\_model, train\_data, type="class")

rf\_test\_pred <- predict(rf\_model, test\_data, type="class")

table(rf\_train\_pred, train\_data$default)

table(rf\_test\_pred, test\_data$default)

输出结果：

rf\_train\_pred No Yes

No 5789 135

Yes 7 69

> (5789+69)/6000

[1] 0.9763333

> table(rf\_test\_pred, test\_data$default)

rf\_test\_pred No Yes

No 3854 97

Yes 17 32

> (3854+32)/4000

[1] 0.9715

可见随机森林的训练准确率为0.9763333，测试准确率为0.9715

④Adaboost

对于Adaboost模型，先将非数值项编码为数字，然后我们取n.trees=5000和interaction.depth=4作限制，阈值取0.5，代码：

# 将非数值项转换为0或1

train.num <- train\_data

test.num <- test\_data

train.num$default<-as.numeric(train\_data$default) - 1

train.num$student<-as.numeric(train\_data$student) - 1

test.num$default<-as.numeric(test\_data$default) - 1

test.num$student<-as.numeric(test\_data$student) - 1

adaboost\_model <- gbm(default ~ ., data = train.num, distribution = "adaboost",n.trees = 5000,interaction.depth = 4)

summary(adaboost\_model)

adaboost\_train\_pred <- predict(adaboost\_model, train.num, n.trees = 5000,type="response")

adaboost\_train\_pred <- ifelse(adaboost\_train\_pred > 0.5, 1, 0)

adaboost\_test\_pred <- predict(adaboost\_model, test.num, n.trees = 5000,type="response")

adaboost\_test\_pred <- ifelse(adaboost\_test\_pred > 0.5, 1, 0)

table(adaboost\_train\_pred, train.num$default)

table(adaboost\_test\_pred, test.num$default)

输出结果：

var rel.inf

balance balance 72.8925814

income income 26.8165937

student student 0.2908249

adaboost\_train\_pred 0 1

0 5796 0

1 0 204

> 1

[1] 1

> table(adaboost\_test\_pred, test.num$default)

adaboost\_test\_pred 0 1

0 3815 80

1 56 49

> (3815+49)/4000

[1] 0.966

可以看出在阈值为0.5时，Adaboost模型的训练准确率为1，测试准确率为0.966。

⑤XGboost

# XGBoost模型

xgtrain\_s <-Matrix::sparse.model.matrix(default~.-1, data = train\_data)

xgtest\_s <-Matrix::sparse.model.matrix(default~.-1, data = test\_data)

dtrain <- xgb.DMatrix(data = xgtrain\_s,label = train\_data$default)

dtest <- xgb.DMatrix(data = xgtest\_s,label = test\_data$default)

xgboost\_model <- xgboost(data = dtrain, max.depth=10,min\_child\_weight=1,gamma=0.1,colsample\_bytree=0.8,subsample=0.8,scale\_pos\_weight=1,eta=0.1,eval\_metric="auc",nround=10000,silent=TRUE)

xgboost\_train\_pred <- predict(xgboost\_model, dtrain)

xgboost\_test\_pred <- predict(xgboost\_model, dtest)

table(as.numeric(xgboost\_train\_pred > 0.5), train\_data$default)

table(as.numeric(xgboost\_test\_pred > 0.5), test\_data$default)

输出：

> table(as.numeric(xgboost\_train\_pred > 0.5), train\_data$default)

No Yes

1 5796 204

> (0+204)/6000

[1] 0.034

> table(as.numeric(xgboost\_test\_pred > 0.5), test\_data$default)

No Yes

1 3871 129

> (0+129)/4000

[1] 0.03225

可以看出准确率很低，模型并不好。

⑥logistic回归

# Logistic回归模型

logistic\_model <- glm(default ~ ., data = train\_data, family = "binomial")

logistic\_train\_pred <- predict(logistic\_model, train\_data, type = "response")

logistic\_test\_pred <- predict(logistic\_model, test\_data, type = "response")

logistic\_train\_pred <- ifelse(logistic\_train\_pred > 0.5, 1, 0)

logistic\_test\_pred <- ifelse(logistic\_test\_pred > 0.5, 1, 0)

table(logistic\_train\_pred, train\_data$default)

table(logistic\_test\_pred, test\_data$default)

输出结果：

logistic\_train\_pred No Yes

0 5772 132

1 24 72

> (5772+72)/6000

[1] 0.974

> table(logistic\_test\_pred, test\_data$default)

logistic\_test\_pred No Yes

0 3850 91

1 21 38

> (3850+38)/4000

[1] 0.972

可以看出logistic回归的训练准确率为0.974，测试准确率为0.972。

（2）利用ROC曲线比较模型的好坏，并计算模型的AUC值。

# 计算训练集和测试集的预测准确率

train\_accuracy <- function(pred, actual) {

mean(pred == actual)

}

test\_accuracy <- function(pred, actual) {

mean(pred == actual)

}

# 计算各模型的训练集和测试集预测准确率

# 计算各模型的ROC曲线和AUC值

train\_accuracy <- function(pred, actual) {

mean(pred == actual)

}

test\_accuracy <- function(pred, actual) {

mean(pred == actual)

}

train.num$default

models <- c("决策树", "随机森林", "Adaboost", "XGBoost", "Logistic回归")

train\_acc <- c(train\_accuracy(tree\_train\_pred, train\_data$default),

train\_accuracy(rf\_train\_pred, train\_data$default),

train\_accuracy(adaboost\_train\_pred, train.num$default),

train\_accuracy(as.numeric(xgboost\_train\_pred > 0.5), train.num$default),

train\_accuracy(logistic\_train\_pred, train.num$default))

test\_acc <- c(test\_accuracy(tree\_test\_pred, test\_data$default),

test\_accuracy(rf\_test\_pred, test\_data$default),

test\_accuracy(adaboost\_test\_pred, test.num$default),

test\_accuracy(as.numeric(xgboost\_test\_pred > 0.5), test.num$default),

test\_accuracy(logistic\_test\_pred, test.num$default))

result <- data.frame(Model = models, Train\_Accuracy = train\_acc, Test\_Accuracy = test\_acc)

result

# auc

roc\_curve <- roc(test\_data$default, as.numeric(tree\_test\_pred))

auc\_tree <- auc(roc\_curve)

roc\_curve <- roc(test\_data$default, as.numeric(rf\_test\_pred))

auc\_rf <- auc(roc\_curve)

roc\_curve <- roc(test\_data$default, as.numeric(adaboost\_test\_pred))

auc\_adaboost <- auc(roc\_curve)

roc\_curve <- roc(test\_data$default, as.numeric(xgboost\_test\_pred))

auc\_xgboost <- auc(roc\_curve)

roc\_curve <- roc(test\_data$default, as.numeric(logistic\_test\_pred))

auc\_logistic <- auc(roc\_curve)

# 打印结果

auc\_result <- data.frame(Model = models, AUC = c(auc\_tree, auc\_rf, auc\_adaboost, auc\_xgboost, auc\_logistic))

auc\_result

输出结果：

> result

Model Train\_Accuracy Test\_Accuracy

1 决策树 0.9733333 0.97250

2 随机森林 0.9763333 0.97150

3 Adaboost 1.0000000 0.96600

4 XGBoost 0.0340000 0.03225

5 Logistic回归 0.9740000 0.97200

> auc\_result

Model AUC

1 决策树 0.6598199

2 随机森林 0.6218352

3 Adaboost 0.6826892

4 XGBoost 0.8246152

5 Logistic回归 0.6445743

（3）请思考如何选择最优模型来预测。

从roc和auc值综合考虑，Adaboost模型最优。