빅데이터 인사이트 실습

로지스틱 회귀 분석, 변수선택, 앙상블

목차

목차

로지스틱 회귀 분석

- 로지스틱 회귀 모형 : 출력변수가 범주형 변수인 경우(분류 문제)에 사용하는 회귀 모형
- 모형 :

$$Pr(Y = 1|x) = F(a + bx)$$

- F로 어떤 함수를 사용하는지에 따라 다양한 함수가 존재
- 로지스틱 모형 : $F(x) = \frac{exp(x)}{1+exp(x)}$ 이 계산의 편리성으로 인해 많이 사용됨.

Data - buytest 자료

- 설명변수 (25개):
  ID, AGE, INCOME, SEX, MARRIED (1: 결혼, 0: 미혼),
  FICO (신용점수), OWNHOME (자가 주택 소유 여부, 1: 소유),
  LOC (거주지, A-H), BUY6, 12, 18 (최근 6, 12, 18개월 간의 구입 횟수),VALUE24 (지난 24개월 간의 구입 총액),ORGSRC (고객 분류),
  DISCBUY (할인 고객 여부, 1: 할인 고객), RETURN24 (지난 24개월 간 상품 반품 여부), COA6 (6개월 간의 주소변경 여부, 1: 주소변경)
- 반응변수: RESPOND (DM에 대한 반응 여부)
- 자료 수: 10,000

로지스틱 회귀분석 with R

- 자료 불러오기

```
buytest = read.table("buytest.txt", header = T)
dim(buytest)
```

```
## [1] 10000    26
```

```
summary(buytest)
```

```
##           ID          RESPOND            AGE            INCOME       SEX
##  000054889:   1   Min.   :0.0000   Min.   :18.00   Min.   : 15.00    : 234
##  000219612:   1   1st Qu.:0.0000   1st Qu.:38.00   1st Qu.: 35.00   F:4489
##  001044039:   1   Median :0.0000   Median :44.00   Median : 50.00   M:5277
##  001079946:   1   Mean   :0.0767   Mean   :44.56   Mean   : 47.95
##  001108462:   1   3rd Qu.:0.0000   3rd Qu.:51.00   3rd Qu.: 61.00
##  001109024:   1   Max.   :1.0000   Max.   :75.00   Max.   :114.00
##  (Other)  :9994                    NA's   :234     NA's   :234
##     MARRIED            FICO          OWNHOME           LOC
##  Min.   :0.0000   Min.   :577.0   Min.   :0.0000   E      :2261
##  1st Qu.:0.0000   1st Qu.:676.0   1st Qu.:0.0000   F      :2168
##  Median :1.0000   Median :695.0   Median :0.0000   B      :1828
##  Mean   :0.5842   Mean   :694.3   Mean   :0.3341   H      :1093
##  3rd Qu.:1.0000   3rd Qu.:714.0   3rd Qu.:1.0000   G      : 950
##  Max.   :1.0000   Max.   :800.0   Max.   :1.0000   A      : 585
##  NA's   :234      NA's   :39      NA's   :234      (Other):1115
##     CLIMATE           BUY6            BUY12           BUY18
##  Min.   :10.00   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:20.00   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :20.00   Median :0.0000   Median :0.0000   Median :0.0000
```

로지스틱 회귀분석 with R

- 자료 전처리

```
buytest[buytest$SEX == "", 'SEX'] = NA
levels(buytest$SEX)[1] = NA
buytest[buytest$ORGSRC == "", 'ORGSRC'] = NA
levels(buytest$ORGSRC)[1] = NA
buydata = buytest[,-c(1, 10,19:26)] # 사용되지 않는 변수 제거
buydata = buydata[complete.cases(buydata),] # 결측치 제거
buydata = model.matrix(~., buydata)[,-1] # 가변수 생성
```

로지스틱 회귀분석 with R

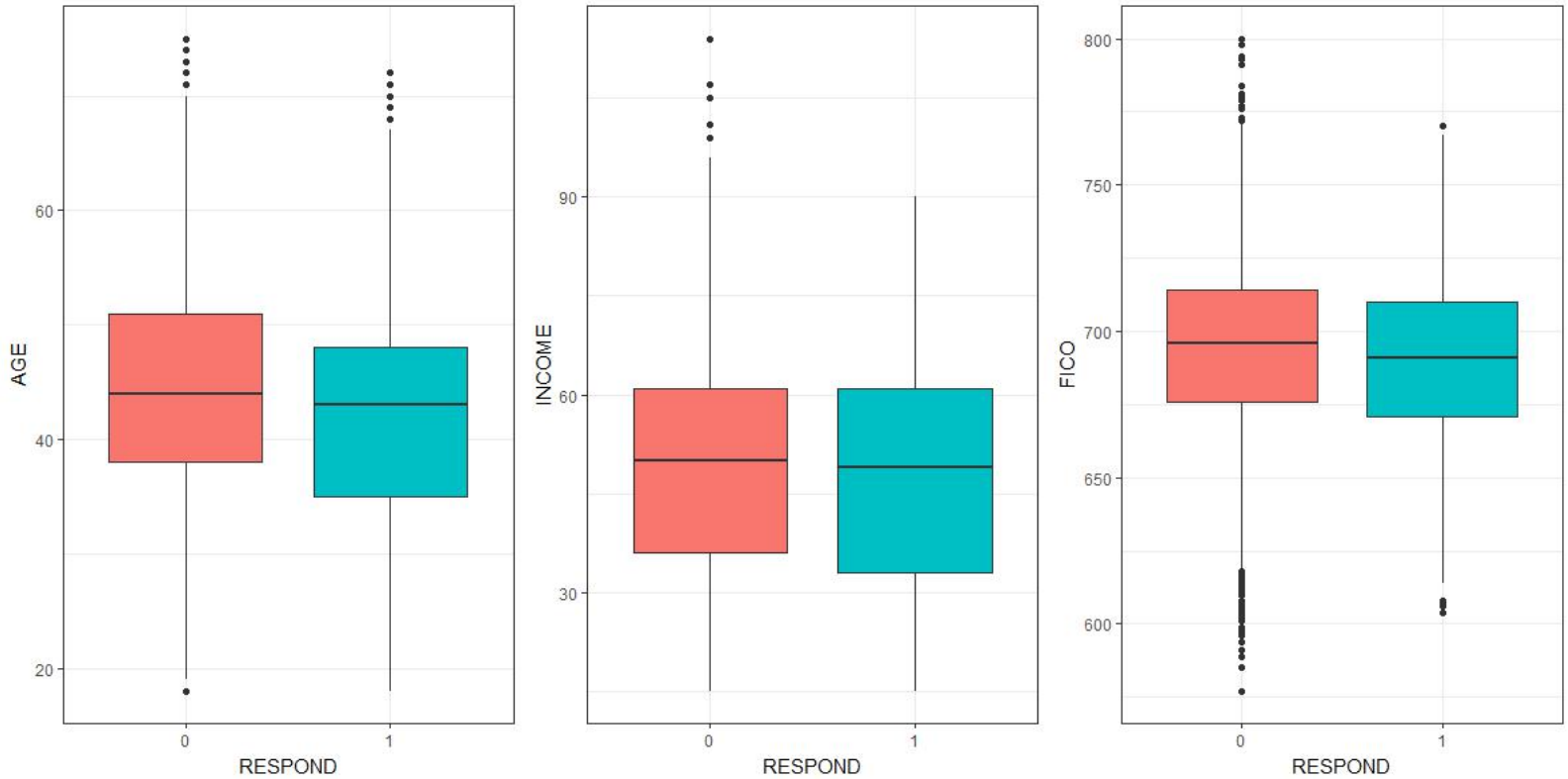
- 자료 전처리

```
head(buydata)
```

```
##   RESPOND AGE INCOME SEXM MARRIED FICO OWNHOME LOCB LOCC LOCD LOCE LOCF LOCG
## 1       0  71     67    1       1  719       0    0    0    0    0    0    0
## 2       0  53     72    1       1  751       0    0    0    0    0    0    0
## 3       0  53     70    0       1  725       0    0    0    0    0    0    0
## 4       0  45     56    0       0  684       0    0    0    0    0    0    0
## 5       0  32     66    0       0  651       0    0    0    0    0    0    0
## 6       0  35     48    0       0  691       1    0    0    0    0    0    0
##   LOCH BUY6 BUY12 BUY18 VALUE24 ORGSRCD ORGSRCI ORGSRCO ORGSRCP ORGSRCR ORGSRCU
## 1    0    1     1     1     318       0       0       1       0       0       0
## 2    0    0     0     0      83       0       0       0       0       1       0
## 3    0    1     1     1     265       1       0       0       0       0       0
## 4    0    0     0     1     448       0       0       1       0       0       0
## 5    0    0     0     0     161       0       0       0       0       1       0
## 6    0    0     0     0     250       0       0       0       0       0       0
##   DISCBUY RETURN24 COA6
## 1       1        0    0
## 2       0        0    0
## 3       0        0    0
## 4       1        0    0
## 5       0        0    0
## 6       0        1    0
```

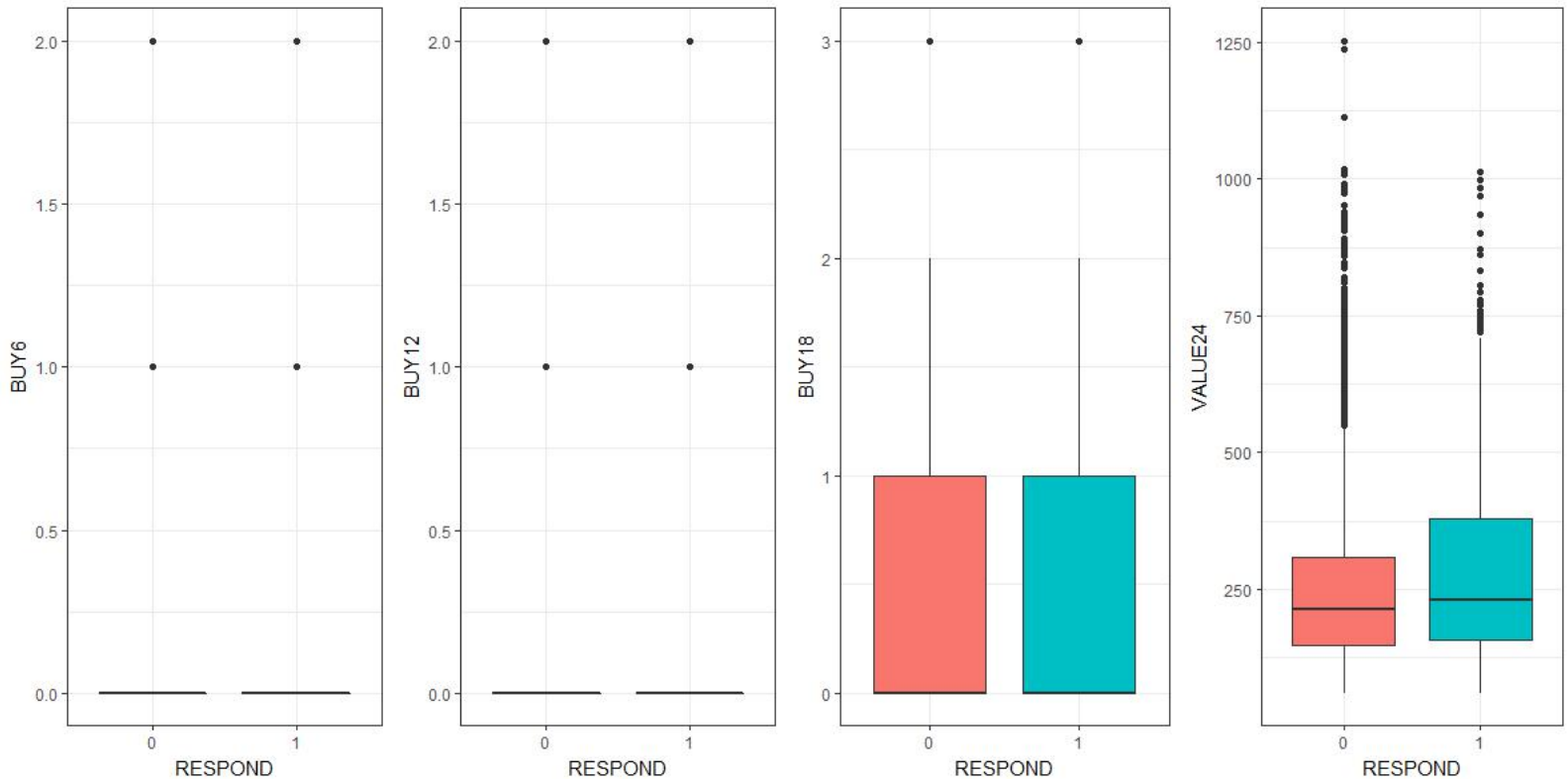
로지스틱 회귀분석 with R

- 자료 분포 살펴보기1



로지스틱 회귀분석 with R

- 자료 분포 살펴보기2



로지스틱 회귀분석 with R

- 모형 적합
- glm(formula, family, data)

```
logit = glm(RESPOND~., data = as.data.frame(buydata), family = 'binomial')
logit
```

```
##
## Call:  glm(formula = RESPOND ~ ., family = "binomial", data = as.data.frame(buydata
##
## Coefficients:
## (Intercept)          AGE       INCOME         SEXM      MARRIED         FICO
##   2.3973899   -0.0380312   -0.0015214   -0.0713741    0.5038395   -0.0049738
##     OWNHOME         LOCB         LOCC         LOCD         LOCE         LOCF
##  -0.4256962   -0.1262557    0.3478551    0.3462659   -0.1352524   -0.1635054
##         LOCG         LOCH         BUY6        BUY12        BUY18      VALUE24
##  -0.0434595   -0.1686800   -0.1093377   -0.4444969    0.7818865   -0.0002022
##     ORGSRCD      ORGSRCI      ORGSRCO      ORGSRCP      ORGSRCR      ORGSRCU
##  -0.1236215    0.0627375    0.0841392    0.1517176    0.0401122    0.0588698
##     DISCBUY     RETURN24         COA6
##  -0.0611812   -0.0846248    0.1299149
##
## Degrees of Freedom: 9220 Total (i.e. Null);  9194 Residual
## Null Deviance:     4915
## Residual Deviance: 4687  AIC: 4741
```

모형의 평가

- 자료분석 절차
  - 자료의 특성파악
  - 모형 적합
  - 모형 선택
  - 모형 평가
  - 결론 도출
- 모형을 평가하는 방법
  - 예측력 : 얼마나 예측을 잘 하는가?
  - 해석력 : 입력변수와 출력변수와의 관계를 잘 설명하는가?
  - 효율성 : 얼마나 적은 수의 입력변수로 모형을 구축했는가?
  - 안정성 : 모집단 내 다른 자료에 적용하는 경우 같은 결과를 주는가?

자료의 분할

- 주어진 자료로 예측력/안정성을 평가하기 위해서 자료를 분할한다.
  - Training set : 모형 적합에 사용할 자료
  - Test set : 모형 평가에 사용할 자료
- 모형 적합과 모형 평가에 사용되는 자료가 서로 다르게 하여 예측력/안정성을 확인할 수 있다.
- 이산형 출력변수의 경우 각 부분집합에 포함되는 출력변수의 비율이 비슷하게 하는것이 좋다.

자료의 분할

- Buydata training set을 7, test set을 3으로 나눈다.

```
set.seed(1)
train_ind = sample(1:nrow(buydata), size = floor(nrow(buydata)*0.7),
                   replace = F)
train = as.data.frame(buydata[train_ind,])
test = as.data.frame(buydata[-train_ind,])
X_train = buydata[train_ind, -1]
y_train = buydata[train_ind, 1]
X_test = buydata[-train_ind, -1]
y_test = buydata[-train_ind, 1]

dim(X_train)

## [1] 6454   26

dim(X_test)

## [1] 2767   26
```

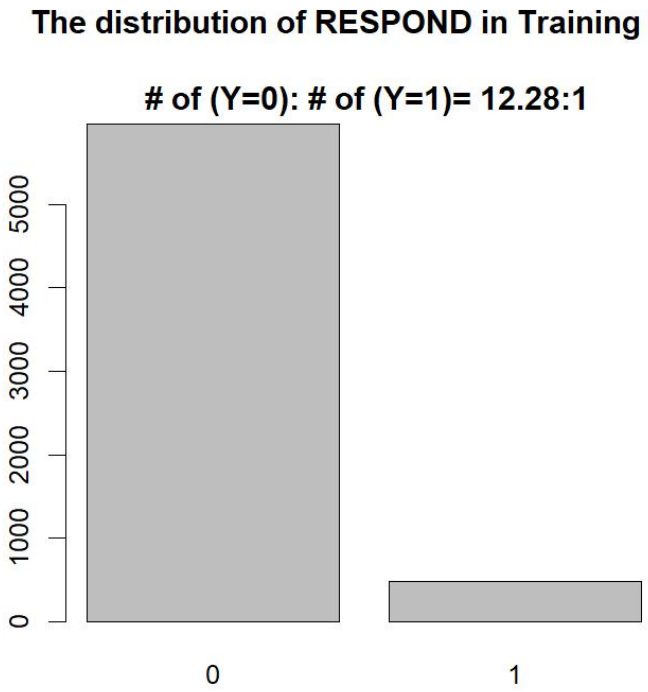
자료의 분할

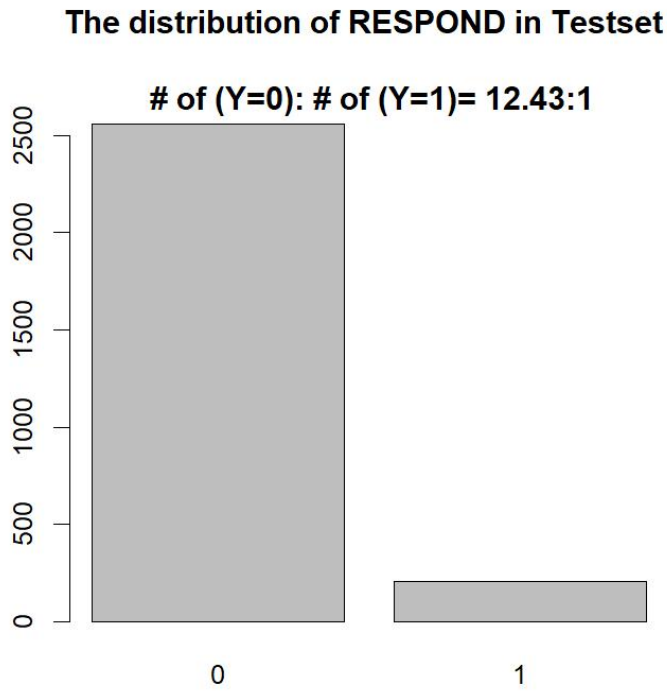
- 학습자료와 예측자료의 반응변수 분포

```
par(mfrow = c(1,2))
barplot(table(y_train),
        main = paste("The distribution of RESPOND in Training set
                     \n # of (Y=0): # of (Y=1)=",
                     paste(round(table(y_train)/sum(y_train == 1),2),
                           collapse = ":")))

barplot(table(y_test),
        main = paste("The distribution of RESPOND in Testset
                     \n # of (Y=0): # of (Y=1)=",
                     paste(round(table(y_test)/sum(y_test == 1),2),
                           collapse = ":")))
```

자료의 분할

- 학습자료와 예측자료의 반응변수 분포





모형의 평가

- 연속형 출력변수의 경우 R^2 값이나 AIC,BIC등을 이용한다.
- 이산형 출력변수
  - 분류가 얼마나 잘되었는지를 오분류율을 기준으로 사용한다.
  - 오류의 종류에 따라 손실이 다를 수 있으므로(예, 질병진단) 손실값을 기준으로 사용할 수 있다.

불균형 자료 모형 평가를 위한 척도

		예측값		
		0	1	합계
실제값	0	a_{00}	a_{01}	$a_{0.}$
	1	a_{10}	a_{11}	$a_{1.}$
	합계	$a_{.0}$	$a_{.1}$	a

Table : 분류표

- 정분류율 $= (a_{00} + a_{11})/a$
- 오분류율 $= (a_{10} + a_{01})/a$
- 민감도 $= a_{11}/a_{1.}$
- 특이도 $= a_{00}/a_{0.}$

불균형 자료 모형 평가를 위한 척도

		예측값		
		0	1	합계
실제값	0	a_{00}	a_{01}	$a_{0.}$
	1	a_{10}	a_{11}	$a_{1.}$
	합계	$a_{.0}$	$a_{.1}$	a

Table : 분류표

- 위와 같이 주어진 분류표에 대해, precision과 recall은 아래와 같이 정의.

$$\begin{aligned} precision &= \frac{a_{11}}{a_{11}+a_{01}} \\ recall &= \frac{a_{11}}{a_{11}+a_{10}} \quad (= \text{민감도}) \end{aligned} \tag{1}$$

- F_1 스코어 : precision, recall의 조화평균.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

사후 확률

- training set으로 로지스틱 회귀모형을 적합한다.
- test set의 사후확률을 계산한다.

$$\Pr(class = 1|\mathbf{x})$$

```
logit = glm(RESPOND~., data = train, family = "binomial")
prob = predict(logit, test, type = 'response')
prob[1:10]
```

```
##          2          3         18         33         34         37         40
## 0.06071576 0.07857268 0.05092754 0.11571055 0.10254184 0.20019725 0.09803432
##         43         44         48
## 0.06987288 0.05551399 0.11749213
```

절단값

- 사후확률이 계산되었으면 확률을 이용하여 분류를 한다.
- 분류하기 위한 기준을 절단값이라고 하며 일반적으로 확률 0.5를 이용한다.
- 특수한 사전 정보나 이익(손실)함수가 있을 경우 절단값을 조절한다.

```
cutoff = 0.5
ifelse(prob[1:10] > cutoff,1,0)

##  2  3 18 33 34 37 40 43 44 48
##  0  0  0  0  0  0  0  0  0  0

cutoff = 1/6

classification = function(model, newdata, cutoff){
prob = predict(model,newdata,'response')
ifelse(prob > cutoff,1,0)
}
```

분류표

- test set의 실제 반응여부와 예측한 반응여부를 비교할 수 있다.

```
table(test$RESPOND, classification(logit, test, 0.5))

##
##        0
##   0 2561
##   1  206

crosstable = function(model, newdata, cutoff){
 table(test$RESPOND,classification(model,newdata,cutoff))
}
crosstable(logit, test, 1/4)

##
##        0    1
##   0 2547   14
##   1  203    3
```

분류표의 분석

```
cutoff_res = function(beta_hat = NULL, newx, response, cutoff, pred_prob = NULL){
  if (!is.null(beta_hat)) {
    X = cbind(1,as.matrix(newx))
    pred_prob = 1/(1+exp(-X%*%beta_hat))}
  pred = ifelse(pred_prob> cutoff, 1, 0)
  error_rate = mean(response != pred)
  sensitivity = sum(response == 1 & pred == 1)/sum(response == 1)
  specificity = sum(response == 0 & pred == 0)/sum(response == 0)
  precision = sum(response == 1 & pred == 1)/sum(pred == 1)
  recall = sensitivity
  if (sum(response == 1 & pred == 1) == 0) {f1 = 0}
  else {f1 = 2*(precision*recall)/(precision+recall)}
  cross_table = table(response, pred)
  return(list(res = c(cutoff, round(error_rate,4),
                      round(sensitivity,4), round(specificity,4), round(f1, 4)),
              cross_table = cross_table))
}
cutoff_res(logit$coefficients, X_test, y_test, 1/4)

## $res
## [1] 0.2500 0.0784 0.0146 0.9945 0.0269
##
## $cross_table
##          pred
## response    0    1
##        0 2547   14
##        1  203    3
```

ROC 곡선

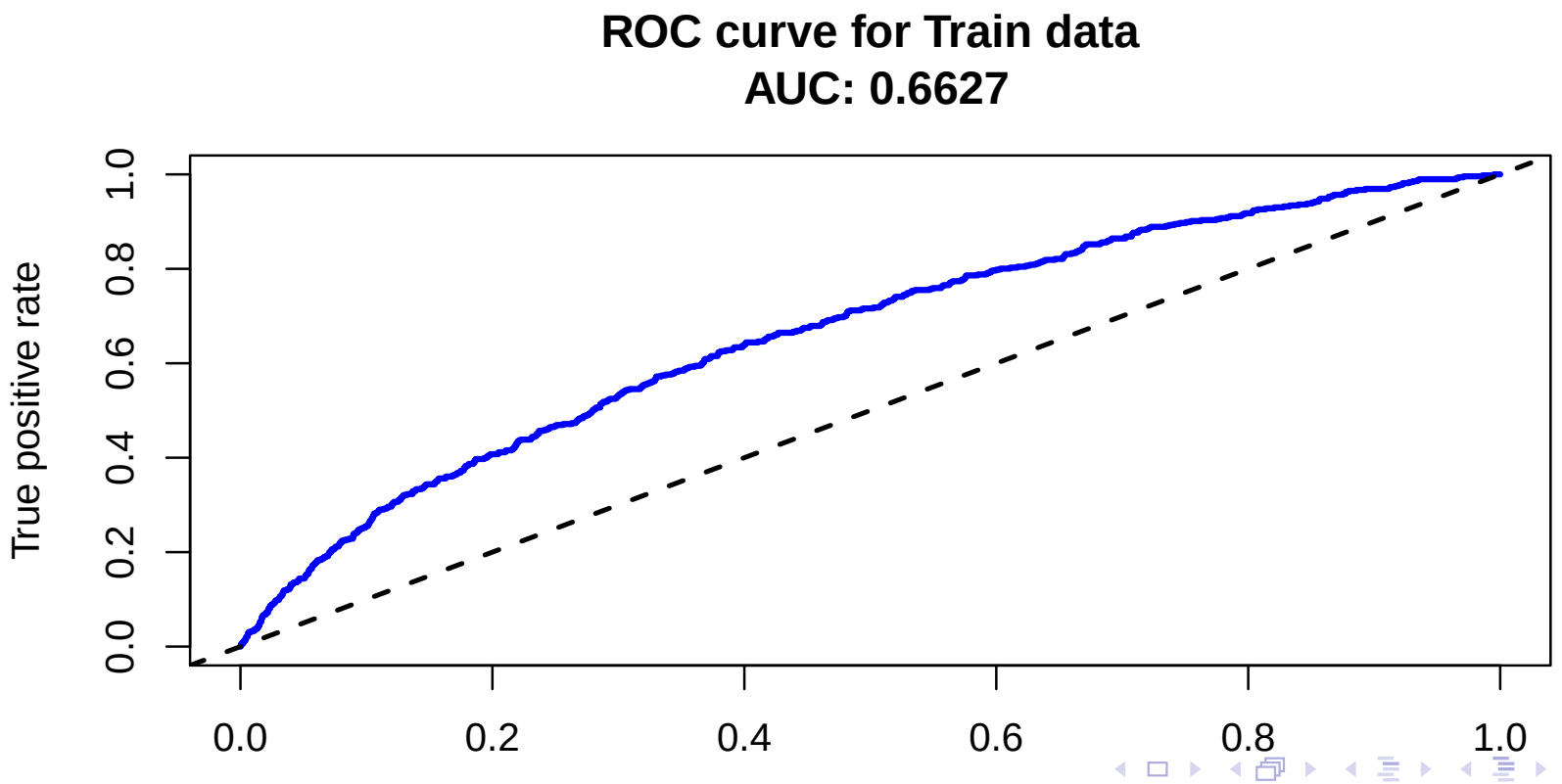
- 절단값에 따라서 특이도와 민감도, 오분류율이 달라진다.
- 절단값을 바꿔가면서 특이도와 민감도를 계산하여 그래프로 나타낸다.
- ROC 곡선
  - x축 : 1 - 특이도
  - y축 : 민감도

ROC 곡선 및 AUC

```
prob = predict(logit, train, type = 'response')
#### AUC
library(ROCR)

## Loading required package: gplots
##
## Attaching package: 'gplots'
## The following object is masked from 'package:stats':
##
##     lowess

####Training set
AUC = performance(prediction(prob , train['RESPOND']) , "auc")
AUC@y.values # area under the curve

## [[1]]
## [1] 0.6627131
```

ROC 곡선 및 AUC

```
#### ROC curve
ROC = performance(prediction(prob ,y_train) , "tpr","fpr")
plot(ROC , main = paste("ROC curve for Train data\n AUC:",
                        round(as.numeric(AUC@y.values),4)),
     col = "blue", lwd = 2.5)
abline(c(0,0), c(1,1), lty = 2, lwd = 2)
```



ROC 곡선

- AUC : 0.6627
- ROC 곡선으로 모형을 평가할 수 있다.
- ROC 곡선의 아래 면적 (Area Under Curve,AUC) 을 이용하여 모형 평가
- 대체로 AUC는 1보다 작고 0.5보다 크다.
- AUC가 크면 각 절단값에 대해서 민감도와 특이도가 높아 좋은 모형으로 볼 수 있다.

로지스틱 모형과 절단값의 선택

- 확률 추정 및 절단값 생성

```
cutoff_can = seq(0.01, 0.99, by = 0.01)

cutoff_out = t(sapply(1:length(cutoff_can),
                      function(i) cutoff_res(logit$coefficients, X_train,
                                             y_train, cutoff_can[i])[[1]]))
colnames(cutoff_out) = c("cutoff","error rate","sensitivity","specificity","f1 score")
```

로지스틱 모형과 절단값의 선택

- 절단값 선택

```
##         cutoff error rate sensitivity specificity f1 score
##   [1,]   0.01     0.9247      1.0000      0.0000   0.1401
##   [2,]   0.02     0.9112      0.9959      0.0149   0.1413
##   [3,]   0.03     0.8633      0.9856      0.0675   0.1467
##   [4,]   0.04     0.7541      0.9259      0.1905   0.1561
##   [5,]   0.05     0.6408      0.8519      0.3190   0.1668
##   [6,]   0.06     0.5217      0.7551      0.4558   0.1790
##   [7,]   0.07     0.4171      0.6584      0.5767   0.1921
##   [8,]   0.08     0.3342      0.5597      0.6744   0.2014
##   [9,]   0.09     0.2693      0.4650      0.7523   0.2064
##  [10,]   0.10     0.2166      0.3868      0.8157   0.2120
##  [11,]   0.11     0.1808      0.3333      0.8587   0.2173
##  [12,]   0.12     0.1554      0.2881      0.8899   0.2182
##  [13,]   0.13     0.1394      0.2284      0.9120   0.1979
##  [14,]   0.14     0.1253      0.1975      0.9298   0.1918
##  [15,]   0.15     0.1154      0.1708      0.9427   0.1822
##  [16,]   0.16     0.1074      0.1399      0.9539   0.1641
##  [17,]   0.17     0.0987      0.1193      0.9650   0.1541
##  [18,]   0.18     0.0942      0.0988      0.9715   0.1364
##  [19,]   0.19     0.0910      0.0802      0.9765   0.1173
##  [20,]   0.20     0.0875      0.0679      0.9812   0.1046
##  [21,]   0.21     0.0858      0.0514      0.9844   0.0828
##  [22,]   0.22     0.0832      0.0370      0.9884   0.0628
##  [23,]   0.23     0.0813      0.0329      0.9908   0.0575
##  [24,]   0.24     0.0793      0.0309      0.9931   0.0554
##  [25,]   0.25     0.0789      0.0267      0.9940   0.0486
```

로지스틱 모형과 절단값의 선택

- 학습자료의 오분류율을 최소화하는 절단값 선택.

```
cutoff_out[which.min(cutoff_out[,2]),]
```

```
##      cutoff  error rate sensitivity specificity    f1 score
##      0.3900      0.0753      0.0021      0.9998      0.0041
```

```
mean(y_train ==1)
```

```
## [1] 0.07530214
```

```
cutoff_res(logit$coefficients, X_train, y_train,
           cutoff_out[which.min(cutoff_out[,2]), 1])[[2]]
```

```
##          pred
## response    0    1
##        0 5967    1
##        1  485    1
```

로지스틱 모형과 절단값의 선택

- 학습자료에서 민감도를 0.5 이상으로 하는 절단값 선택.

```
cutoff_out[tail(which(cutoff_out[,3] >= 0.5), n = 1),]
```

```
##      cutoff  error rate sensitivity specificity    f1 score
##      0.0800      0.3342      0.5597      0.6744      0.2014
```

```
cutoff_sel = cutoff_out[tail(which(cutoff_out[,3] >= 0.5), n = 1), 1]
cutoff_res(logit$coefficients, X_train, y_train, cutoff_sel)[[2]]
```

```
##          pred
## response    0    1
##        0 4025 1943
##        1  214  272
```

- 학습자료의 F_1 스코어를 최대화하는 절단값 선택.

```
cutoff_out[which.max(cutoff_out[,5]),]
```

```
##      cutoff  error rate sensitivity specificity    f1 score
##      0.1200      0.1554      0.2881      0.8899      0.2182
```

```
cutoff_res(logit$coefficients, X_train, y_train,
           cutoff_out[which.max(cutoff_out[,5]), 1])[[2]]
```

```
##          pred
## response    0    1
##        0 5311  657
##        1  346  140
```

로지스틱 모형과 절단값의 선택

- 예측자료에서의 비교.

```
cutoff = c()
cutoff[1] = cutoff_out[which.min(cutoff_out[,2]), 1]
cutoff[2] = cutoff_out[tail(which(cutoff_out[,3] >= 0.5), n = 1), 1]
cutoff[3] = cutoff_out[which.max(cutoff_out[,5]), 1]
as.data.frame(sapply(cutoff, function(cut) cutoff_res(logit$coefficients, X_test,
                                                      y_test, cut)[[1]]),
              row.names = colnames(cutoff_out))
```

```
##                    V1     V2     V3
## cutoff         0.3900 0.0800 0.1200
## error rate     0.0748 0.3332 0.1547
## sensitivity    0.0049 0.5000 0.2718
## specificity    0.9992 0.6802 0.8914
## f1 score       0.0096 0.1826 0.2074
```

목차

Variable selection

- 선형 회귀 : 과적합의 문제가 발생할 수 있다.
- 변수 선택을 통해 과적합을 피할 수 있다.
- 변수 선택법의 종류 : 전진 선택법, 후진 선택법
- 변수 선택의 기준 : AIC, Adjusted R^2, ...

Variable selection with R

- Logistic + Forward selection (AIC)

```
null = glm(RESPOND~1, data = train); full = glm(RESPOND~., data = train)
forward = step(null, scope = list(lower = null, upper = full),
               data = train, direction = "forward")
```

```
summary(forward)
```

```
##
## Call:
## glm(formula = RESPOND ~ BUY18 + OWNHOME + AGE + BUY12 + MARRIED +
##     LOCC + FICO + LOCD + LOCB + INCOME, data = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.26791  -0.09206  -0.06670  -0.03868   1.01028
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.3262361  0.0801814   4.069 4.78e-05 ***
## BUY18        0.0694760  0.0090448   7.681 1.81e-14 ***
## OWNHOME     -0.0329263  0.0070377  -4.679 2.95e-06 ***
## AGE         -0.0020054  0.0003530  -5.681 1.40e-08 ***
## BUY12       -0.0456320  0.0119280  -3.826 0.000132 ***
## MARRIED      0.0276245  0.0071956   3.839 0.000125 ***
## LOCC         0.0382158  0.0141253   2.705 0.006839 **
## FICO        -0.0002418  0.0001121  -2.157 0.031051 *
## LOCD         0.0228751  0.0144686   1.581 0.113923
## LOCB        -0.0131579  0.0085870  -1.532 0.125494
```

Penalized regression

- 다음과 같은 선형 모형에서

$$y = X\beta + \epsilon \quad , \quad \epsilon \sim N(0, \sigma^2 I)$$

$\hat{\beta}_{LSE} = (X^T X)^{-1} X^T y$은 $\hat{\beta}_{LSE} \sim N(\beta, (X^T X)^{-1}\sigma^2)$이므로 β의 불편 추정량이지만 분산이 매우 클수가 있다.

- 여기서 objective fucntion $= (y - X\beta)^T (y - X\beta)$

Penalized regression

- 최소화 하고 싶은 object function에 penalty를 줌으로써 불편 성질은 잃어버리더라도 분산을 줄일 수 있다.
- 대표적으로 Ridge와 Lasso라는 방법이 있다.
- Ridge

$$\hat{\beta} = \mathit{argmin}_{\beta}\left\{(y - X\beta)^T(y - X\beta) + \lambda\sum_{j=1}^{p}\beta_j^2\right\}$$

- Lasso

$$\hat{\beta} = \mathit{argmin}_{\beta}\left\{(y - X\beta)^T(y - X\beta) + \lambda\sum_{j=1}^{p}|\beta_j|\right\}$$

- Lasso는 shrinkage뿐만 아니라 변수 선택까지 해준다는 특징이 있다.
- tuning parameter λ는 cross validation등을 이용해 추정한다.

Penalized regression

- 벌점화 기법 : 자료의 손실을 줄이면서 과적합을 피하는 방법
- 회귀 계수에 벌점 함수를 줌으로써 과적합을 해결
- 반응변수가 연속형인 경우:

$$\underset{\beta}{\text{argmin}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} (Y_i - X_i'\beta)^2 + \lambda P(\beta) \right\}$$

  반응변수가 이산형인 경우: $\text{argmin}_\beta \{ -\frac{1}{n} \text{log-likelihood} + \lambda P(\beta) \}$

- Ridge penalty: $P(\beta) = \sum_j \beta_j^2$

  Lasso penalty: $P(\beta) = \sum_j |\beta_j|$

Penalized regression with R

- Logistic + Ridge

```
#---- logistic + ridge
library(glmnet)

## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-13

ridge.fit = glmnet(X_train, as.factor(y_train), alpha = 0,
                   family="binomial" )
ridge.fit$lambda[c(1, 10, 100)]

## [1] 26.175438761 11.330722582  0.002617544

ridge.fit$beta[,c(1, 10, 100)]

## 26 x 3 sparse Matrix of class "dgCMatrix"
##                       s0            s9           s99
## AGE      -1.556600e-39 -1.352294e-04 -0.0298988675
## INCOME   -4.868359e-40 -4.212515e-05 -0.0047845732
## SEXM     -1.505248e-39 -1.283891e-04  0.0005356394
## MARRIED   1.347485e-38  1.173292e-03  0.4010388756
## FICO     -2.504627e-40 -2.173753e-05 -0.0033362463
## OWNHOME  -3.683045e-38 -3.192917e-03 -0.5038841946
## LOCB     -1.319271e-38 -1.146848e-03 -0.3429212597
## LOCC      4.351218e-38  3.774385e-03  0.3137131718
## LOCD      2.881456e-38  2.499459e-03  0.1215334481
## LOCE     -1.382887e-38 -1.195488e-03 -0.2504779251
## LOCF     -7.235543e-39 -6.217229e-04 -0.1242547872
```

Penalized regression with R

- Logistic + Ridge with 10-fold CV

```
set.seed(1)
cv.ridge = cv.glmnet(X_train, as.factor(y_train), alpha = 0,
                     family = "binomial")
bestlam = cv.ridge$lambda.min
ridge.fit = glmnet(X_train,as.factor(y_train), alpha = 0,
                   lambda = bestlam, family = "binomial")
ridge.fit$beta
```

```
## 26 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## AGE      -0.0269156186
## INCOME   -0.0045085786
## SEXM      0.0007387316
## MARRIED   0.3520187373
## FICO     -0.0031073647
## OWNHOME  -0.4686363636
## LOCB     -0.2719477846
## LOCC      0.3464695216
## LOCD      0.1717206193
## LOCE     -0.1888539060
## LOCF     -0.0749223342
## LOCG     -0.0361349143
## LOCH     -0.1441467686
## BUY6     -0.0582406887
## BUY12    -0.2545706755
## BUY18     0.6696679228
## VALUE24  -0.0002322529
```

Penalized regression with R

- Logistic + Lasso

```
#---- logistic + lasso
lasso.fit = glmnet(X_train, as.factor(y_train), alpha = 1,
                   family="binomial")
lasso.fit$lambda[c(1, 5, 10)]

## [1] 0.02617544 0.01804171 0.01133072

lasso.fit$beta[,c(1, 5, 10)]

## 26 x 3 sparse Matrix of class "dgCMatrix"
##           s0        s4           s9
## AGE        . .         -0.005299303
## INCOME     . .          .
## SEXM       . .          .
## MARRIED    . .          .
## FICO       . .          .
## OWNHOME    . .         -0.166842286
## LOCB       . .          .
## LOCC       . .          .
## LOCD       . .          .
## LOCE       . .          .
## LOCF       . .          .
## LOCG       . .          .
## LOCH       . .          .
## BUY6       . .          .
## BUY12      . .          .
## BUY18      . 0.1907342  0.324379835
```

Penalized regression with R

- Logistic + Lasso with 10-fold CV

```
set.seed(1)
cv.lasso = cv.glmnet(X_train, as.factor(y_train), alpha = 1,
                     family="binomial")
bestlam = cv.lasso$lambda.min
lasso.fit = glmnet(X_train, as.factor(y_train), alpha = 1,
                   lambda = bestlam, family="binomial")
lasso.fit$beta
```

```
## 26 x 1 sparse Matrix of class "dgCMatrix"
##                      s0
## AGE      -0.025073002
## INCOME   -0.002932617
## SEXM      .
## MARRIED   0.305312003
## FICO     -0.002275609
## OWNHOME  -0.449116044
## LOCB     -0.123710234
## LOCC      0.368131200
## LOCD      0.179503732
## LOCE     -0.040834021
## LOCF      .
## LOCG      .
## LOCH      .
## BUY6      .
## BUY12    -0.215888169
## BUY18     0.595990891
## VALUE24
```

방법들 비교

- 위와 같은 모형 적합 및 평가, 절단값 선택과정을 전진선택법 (AIC 사용), Ridge, Lasso에 적용하여 예측자료에서 비교.
- 모형 평가에서는 각 모형에 대해 학습자료에서의 AUC, 예측자료에서의 AUC을 계산하여 비교.
- 학습자료에서의 절단값을 선택하는 여러가지 기준을 이용하여 예측자료에서 예측 성능 비교.

방법들 비교

- 모형별 회귀계수 저장.

```
beta_hat = logit$coefficients

#-----------------------------------
# forward selection
#-----------------------------------
tmp = forward$coefficients
beta_forward = c()
for (i in 1:length(beta_hat)){
  if (names(beta_hat)[i] %in% names(tmp)) beta_forward[i] = tmp[names(beta_hat)[i]]
  else beta_forward[i] = 0
}
beta_hat = cbind(beta_hat, beta_forward)

#-----------------------------------
# Ridge & LASSO
#-----------------------------------
beta_hat = cbind(beta_hat, c(ridge.fit$a0, as.vector(ridge.fit$beta)),
                 c(lasso.fit$a0, as.vector(lasso.fit$beta)))
```

방법들 비교

- 모형별 AUC 값 비교.

```
#-----------------------------------
# Model Evaluation
#-----------------------------------
library(ROCR)

auc_res = function(beta = NULL, newx, newy, pred_prob = NULL){
  if (!is.null(beta)){
    X = cbind(1,as.matrix(newx))
    pred_prob = 1/(1+exp(-X%*%beta))    }
  AUC = performance(prediction(pred_prob , newy) , "auc")
  return(AUC@y.values[[1]])
}
auc_table = rbind(sapply(1:4, function(i) auc_res(beta_hat[,i], X_train, y_train)),
                  sapply(1:4, function(i) auc_res(beta_hat[,i], X_test, y_test)))
rownames(auc_table) = c('Training set', 'Test set')
colnames(auc_table) = model_names = c('Logistic','Logistic+AIC', 'Ridge', 'LASSO')
auc_table
```

```
##                Logistic Logistic+AIC     Ridge     LASSO
## Training set 0.6627131    0.6577249 0.6622918 0.6577115
## Test set     0.6417832    0.6444236 0.6408866 0.6374084
```

방법들 비교

- 학습자료에서의 오분류율을 최소화하는 절단값을 이용하여 예측자료에서 예측 성능 비교.

```
cut_sel = matrix(0, nrow = 4, ncol = 3)
for (i in 1:4){
  cutoff_out =  t(sapply(1:length(cutoff_can),
                      function(j) cutoff_res(beta_hat[,i], X_train,
                                             y_train, cutoff_can[j])[[1]]))
  cut_sel[i, 1] = cutoff_out[which.min(cutoff_out[,2]), 1]
  cut_sel[i, 2] = cutoff_out[tail(which(cutoff_out[,3] >= 0.5), n = 1), 1]
  cut_sel[i, 3] = cutoff_out[which.max(cutoff_out[,5]), 1]
}

matrix(t(sapply(1:4, function(i) cutoff_res(beta_hat[,i], X_test,
                                                y_test, cut_sel[i, 1])[[1]])),
       nrow = 4,
       dimnames =list(model_names, c("cutoff","error rate","sensitivity",
                                     "specificity","f1 score")))
```

```
##              cutoff error rate sensitivity specificity f1 score
## Logistic       0.39     0.0748      0.0049      0.9992   0.0096
## Logistic+AIC   0.57     0.0744      0.0000      1.0000   0.0000
## Ridge          0.34     0.0744      0.0049      0.9996   0.0096
## LASSO          0.35     0.0744      0.0000      1.0000   0.0000
```

방법들 비교

- 학습자료에서의 민감도를 0.5 이상으로 하는 절단값을 이용하여 예측자료에서 예측 성능 비교.

	cutoff	error rate	sensitivity	specificity	f1 score
Logistic	0.08	0.3332	0.5000	0.6802	0.1826
Logistic+AIC	0.52	0.3863	0.5485	0.6189	0.1745
Ridge	0.08	0.3325	0.5194	0.6794	0.1887
LASSO	0.08	0.3267	0.5146	0.6861	0.1900

- 학습자료에서의 F_1 스코어를 최대로 하는 절단값을 이용하여 예측자료에서 예측 성능 비교.

	cutoff	error rate	sensitivity	specificity	f1 score
Logistic	0.12	0.1547	0.2718	0.8914	0.2074
Logistic+AIC	0.53	0.1568	0.2864	0.8879	0.2138
Ridge	0.10	0.2035	0.3544	0.8321	0.2059
LASSO	0.10	0.2017	0.3301	0.8360	0.1960

목차

의사결정나무

- if-then 으로 표현되는 규칙으로 생성되는 모형
- root node, child node, parent node, terminal node, internal node, branch, depth 등으로 구성.
- 장점 : 좋은 해석력과 이상치에 둔감.
- 단점 : 예측력이 떨어지고 분산이 커서 추정량이 안정적이지 않음.

rpart 패키지

- CART와 거의 유사.
- Gini index를 사용하여 불순도 계산.
- 비용 복잡도 (cost-complexity)를 이용하여 가지치기 (pruning)를 실행.

$$\text{cost complexity}(a) = \text{나무 T의 오분류율} + a|T|$$

여기서 $|T|$은 나무 T의 끝마디 개수.

german dataset

- german : German Credit Data
- 데이터 설명
  - 설명변수: 예금 계좌의 상태, 기간, 신용 기록, 대출 목적, 대출량, 적립예금, 저축 채권, 취업 기간 등의 개체의 현재 재정 상태 또는 직장, 집에 관련된 20 개의 변수.
  - 반응변수: 데이터의 마지막 열은 해당 개체의 재정 상태 class label ("Good" 또는 "Bad") 이다.

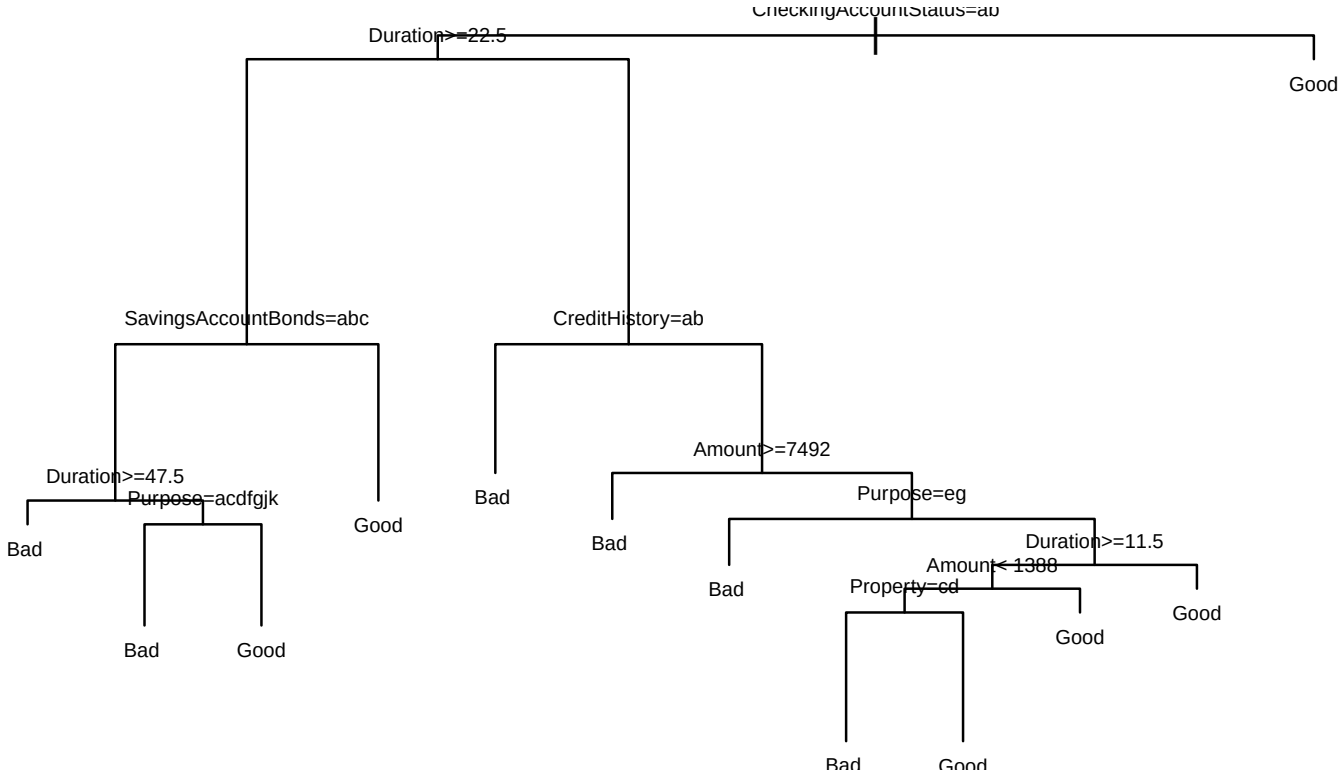
```
library(rchallenge) # for german data
data(german)
summary(german)
str(german)
```

german with decision tree

```
library(rpart)
tree.german = rpart(Class~., data = german)
tree.german

## n= 1000
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 1000 300 Good (0.3000000 0.7000000)
##     2) CheckingAccountStatus=lt.0,0.to.200 543 240 Good (0.4419890 0.5580110)
##       4) Duration>=22.5 237 103 Bad (0.5654008 0.4345992)
##         8) SavingsAccountBonds=lt.100,100.to.500,500.to.1000 196  74 Bad (0.6224490 0.3775510)
##          16) Duration>=47.5 36   5 Bad (0.8611111 0.1388889) *
##          17) Duration< 47.5 160  69 Bad (0.5687500 0.4312500)
##            34) Purpose=NewCar,Furniture.Equipment,Radio.Television,Repairs,Education,Business,Other 137  52 Bad (0.6204380 0
##            35) Purpose=UsedCar 23   6 Good (0.2608696 0.7391304) *
##         9) SavingsAccountBonds=gt.1000,Unknown 41  12 Good (0.2926829 0.7073171) *
##       5) Duration< 22.5 306 106 Good (0.3464052 0.6535948)
##        10) CreditHistory=NoCredit.AllPaid,ThisBank.AllPaid 28   7 Bad (0.7500000 0.2500000) *
##        11) CreditHistory=PaidDuly,Delay,Critical 278  85 Good (0.3057554 0.6942446)
##          22) Amount>=7491.5 7   1 Bad (0.8571429 0.1428571) *
##          23) Amount< 7491.5 271  79 Good (0.2915129 0.7084871)
##            46) Purpose=DomesticAppliance,Education 15   5 Bad (0.6666667 0.3333333) *
##            47) Purpose=NewCar,UsedCar,Furniture.Equipment,Radio.Television,Repairs,Retraining,Business,Other 256  69 Good (0
##              94) Duration>=11.5 183  60 Good (0.3278689 0.6721311)
##               188) Amount< 1387.5 65  31 Good (0.4769231 0.5230769)
##                 376) Property=CarOther,Unknown 20   3 Bad (0.8500000 0.1500000) *
##                 377) Property=RealEstate,Insurance 45  14 Good (0.3111111 0.6888889) *
##               189) Amount>=1387.5 118  29 Good (0.2457627 0.7542373) *
##              95) Duration< 11.5 73   9 Good (0.1232877 0.8767123) *
##     3) CheckingAccountStatus=gt.200,none 457  60 Good (0.1312910 0.8687090) *
```

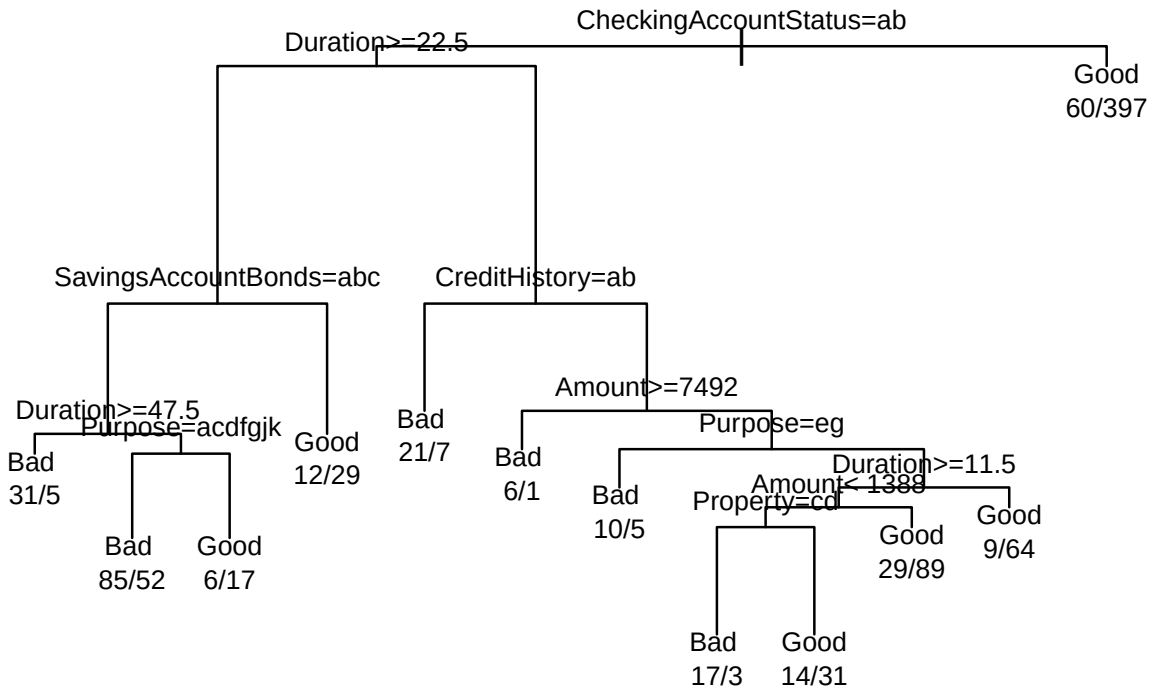
german with decision tree

```
plot(tree.german)
text(tree.german, cex = 0.5)
```



german with decision tree

```
plot(tree.german, margin = 0.1)
text(tree.german, cex = 0.7, use.n =T)
```



Pruning, 가지치기

- cost-complexity을 최소화하는 모형 선택

```
printcp(tree.german)

##
## Classification tree:
## rpart(formula = Class ~ ., data = german)
##
## Variables actually used in tree construction:
## [1] Amount                CheckingAccountStatus CreditHistory
## [4] Duration              Property              Purpose
## [7] SavingsAccountBonds
##
## Root node error: 300/1000 = 0.3
##
## n= 1000
##
##         CP nsplit rel error  xerror     xstd
## 1 0.051667      0   1.00000 1.00000 0.048305
## 2 0.046667      3   0.84000 0.98000 0.048024
## 3 0.018333      4   0.79333 0.88333 0.046521
## 4 0.016667      6   0.75667 0.89000 0.046632
## 5 0.015556      8   0.72333 0.87667 0.046408
## 6 0.010000     11   0.67667 0.86000 0.046120
```

Pruning, 가지치기

- `CP` : complexity parameter (cost-complexity)
- `nsplit` : $|T| - 1$
- `rel error` : $RSS(|T|)/RSS(1)$, $RSS(k)$: residual sum of squares for the tree with k terminal nodes.

$$\text{나무 T의 오분류율} = \texttt{rel error} \times \texttt{Root node error}$$

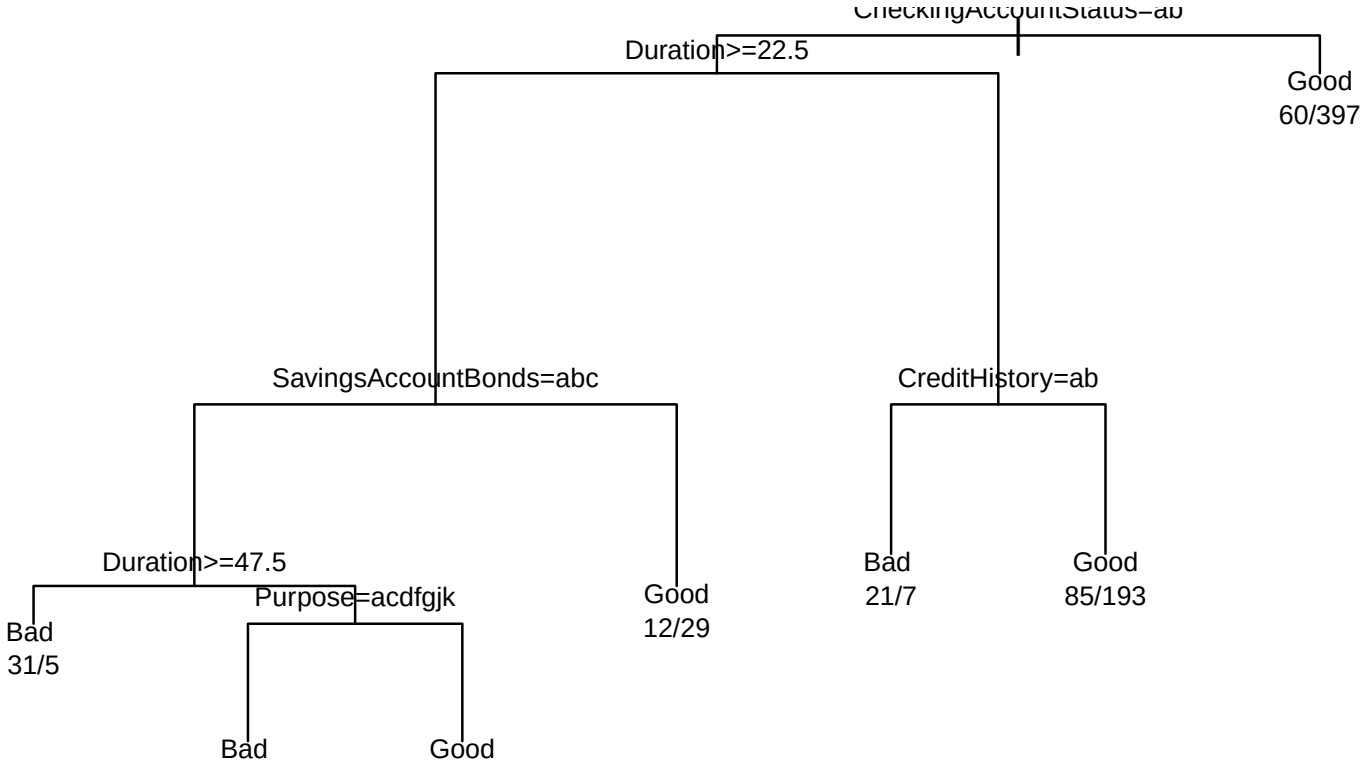
- `xerror` $\times$ `Root node error` : cross-validation error rate (10-fold CV)
- `xstd` : the standard error of `xerror`

Pruning, 가지치기

```
prune.german = prune(tree.german, cp = 0.017)
prune.german

## n= 1000
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1000 300 Good (0.3000000 0.7000000)
##    2) CheckingAccountStatus=lt.0,0.to.200 543 240 Good (0.4419890 0.5580110)
##      4) Duration>=22.5 237 103 Bad (0.5654008 0.4345992)
##        8) SavingsAccountBonds=lt.100,100.to.500,500.to.1000 196  74 Bad (0.6224490 0.377
##         16) Duration>=47.5 36   5 Bad (0.8611111 0.1388889) *
##         17) Duration< 47.5 160  69 Bad (0.5687500 0.4312500)
##           34) Purpose=NewCar,Furniture.Equipment,Radio.Television,Repairs,Education,Busi
##           35) Purpose=UsedCar 23   6 Good (0.2608696 0.7391304) *
##        9) SavingsAccountBonds=gt.1000,Unknown 41  12 Good (0.2926829 0.7073171) *
##      5) Duration< 22.5 306 106 Good (0.3464052 0.6535948)
##       10) CreditHistory=NoCredit.AllPaid,ThisBank.AllPaid 28   7 Bad (0.7500000 0.250000
##       11) CreditHistory=PaidDuly,Delay,Critical 278  85 Good (0.3057554 0.6942446) *
##    3) CheckingAccountStatus=gt.200,none 457  60 Good (0.1312910 0.8687090) *
```

Pruning, 가지치기

```
plot(prune.german)
text(prune.german, cex = 0.7, use.n =T)
```



Pruning, 가지치기

- CV을 이용하여 적절한 가지치기 찾기

```
set.seed(1)
K = 10
sample.ind = sample(1:K, size = nrow(german), replace = T)
cp = seq(from = 0.005, to = 0.03, length = 40)
error = matrix(0, nrow = length(cp), K)
for (i in 1:length(cp)){
  for (j in 1:K){
    tmp = rpart(Class~., data = german[sample.ind != j,], cp = cp[i])
    error[i,j] = sum(predict(tmp, german[sample.ind == j,], type = "class")
                     != german[sample.ind == j,]$Class)/sum(sample.ind == j)
  }
}
rowMeans(error)

##  [1] 0.3050697 0.3012241 0.2939457 0.2912742 0.2837259 0.2819868 0.2778273
##  [8] 0.2789384 0.2761240 0.2770855 0.2778928 0.2787391 0.2809854 0.2818945
## [15] 0.2818945 0.2810249 0.2800634 0.2800634 0.2800634 0.2809562 0.2797934
## [22] 0.2789601 0.2800964 0.2800964 0.2800964 0.2800964 0.2806363 0.2806363
## [29] 0.2806363 0.2806363 0.2806363 0.2858537 0.2858537 0.2901548 0.2901548
## [36] 0.2901548 0.2901548 0.2901548 0.2901548 0.2901548

cp[which.min(rowMeans(error))]

## [1] 0.01012821
```

Gradient Boosting

- AdaBoost: 자료의 오분류 여부에 의해 가중치를 업데이트.
- GBM (Gradient Boosting Method): 그래디언트 강하 기법으로 부스팅을 설명.
  - Shrinkage (Friedman 2001)
    $F_n = F_{n-1} + f_n$으로 업데이트를 하는 대신, $F_n = F_{n-1} + \eta f_n$, $\eta \in (0, 1)$으로 업데이트.

Gradient Boosting with xgboost

xgboost is short for eXtreme Gradient Boosting package.

- objective
  - reg:linear: for linear regression
  - binary:logistic: for logistic regression for classification
- eta: step size of each boosting step
- max.depth: maximum depth of tree
- nround: the max number of iterations

```
library(xgboost)
```

Gradient Boosting with xgboost

데이터 agaricus 설명

- Mushroom Data set 변형
  (http://archive.ics.uci.edu/ml/datasets/Mushroom 참고)
- Mushroom Data 각각의 범주형 변수에 대해 가변수 생성
- 버섯 개체에 대해 독성이 있는지 여부 판단 (edible $= 0$, poisonous $= 1$)

```
data(agaricus.train, package='xgboost')
data(agaricus.test, package='xgboost')
dim(agaricus.train$data)

## [1] 6513  126

class(agaricus.train$data)

## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"

table(agaricus.train$label)

##
##    0    1
## 3373 3140
```

Gradient Boosting with xgboost

```
# training
train = agaricus.train
test = agaricus.test
boost.fit = xgboost(data = train$data, label = train$label, max.depth = 2,
                    eta = 0.1, nround = 2, objective = "binary:logistic")

## [1] train-error:0.046522
## [2] train-error:0.041609

# testing
pred = predict(boost.fit, test$data)
mean(test$label != round(pred))

## [1] 0.04034761
```

Gradient Boosting with xgboost

- Validation for nround

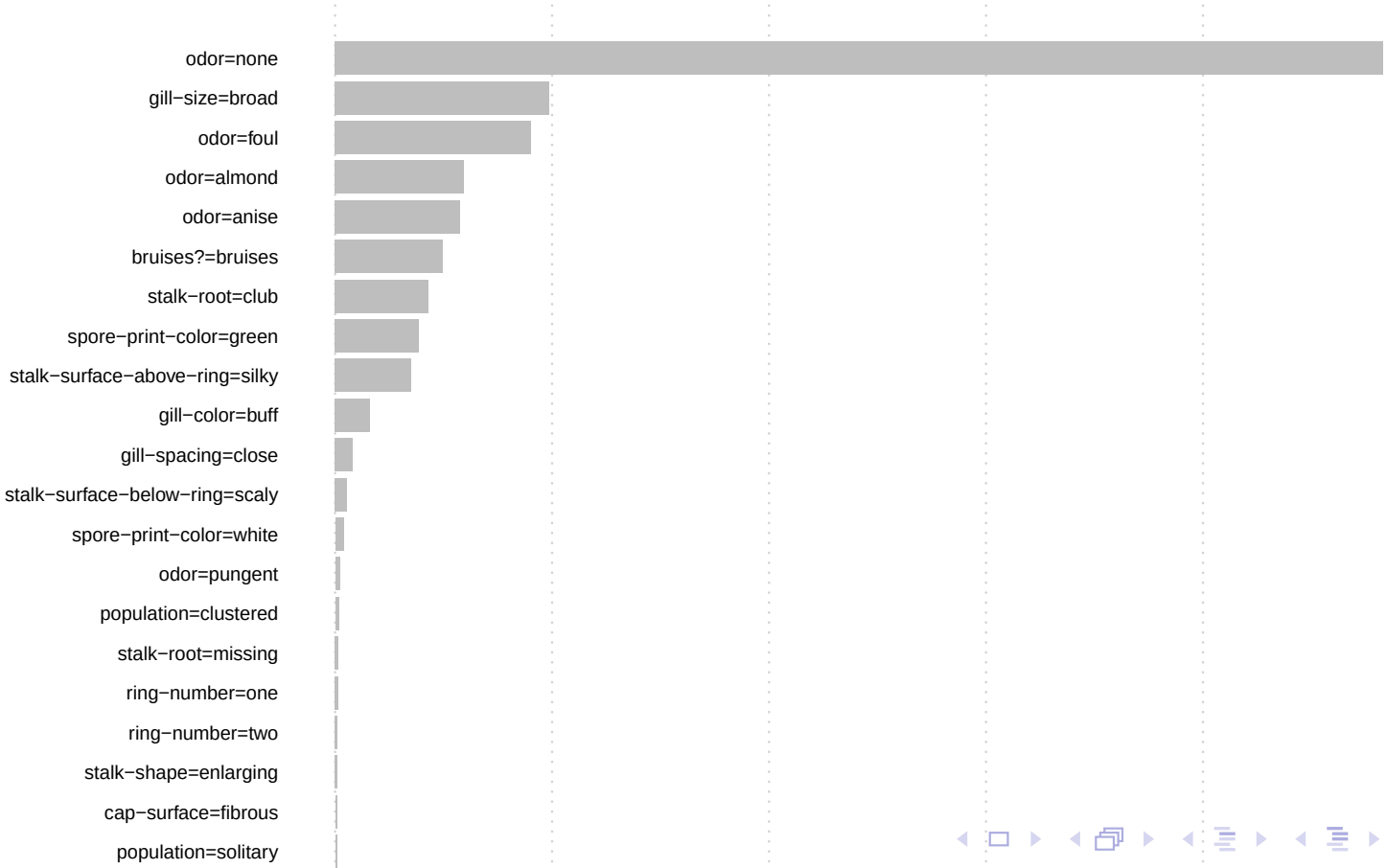
```
# validataion set
set.seed(123)
val.ind = sample(1:nrow(train$data), size = floor(nrow(train$data)*0.5))
val.err = c()
candidates = seq(from = 10, length = 30, by = 10)

for (i in candidates){
  boost.val = xgboost(data = train$data[-val.ind,], label = train$label[-val.ind], max.dept
                      eta = 0.1, nround = i, objective = "binary:logistic", verbose = 0)
  pred.val = predict(boost.val, train$data[val.ind,])
  val.err = c(val.err, mean(train$label[val.ind] != round(pred.val)))
}
val.err

##  [1] 0.021805897 0.020884521 0.021498771 0.011670762 0.001842752 0.001842752
##  [7] 0.001842752 0.001842752 0.001842752 0.001228501 0.001228501 0.001228501
## [13] 0.001228501 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [19] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [25] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000

which.min(val.err)

## [1] 14
```

Gradient Boosting with xgboost

```
boost.fit = xgboost(data = train$data, label = train$label, max.depth = 2,
                    eta = 0.1, nround = candidates[which.min(val.err)], objective = "binar
pred = predict(boost.fit, test$data)
mean(test$label != round(pred))

## [1] 0

# relative influence
import_mat = xgb.importance(train$data@Dimnames[[2]], model = boost.fit)
print(import_mat)

##                            Feature         Gain        Cover   Frequency
##  1:                       odor=none 4.832089e-01 2.660698e-01 0.172514620
##  2:                  gill-size=broad 9.864972e-02 9.068992e-02 0.064327485
##  3:                       odor=foul 9.019080e-02 1.189049e-01 0.061403509
##  4:                     odor=almond 5.937469e-02 8.998939e-02 0.105263158
##  5:                      odor=anise 5.750730e-02 8.793655e-02 0.102339181
##  6:                bruises?=bruises 4.966822e-02 3.070950e-02 0.008771930
##  7:                 stalk-root=club 4.304230e-02 2.212148e-02 0.005847953
##  8:         spore-print-color=green 3.872550e-02 1.095561e-01 0.116959064
##  9: stalk-surface-above-ring=silky 3.494063e-02 6.802667e-02 0.064327485
## 10:                 gill-color=buff 1.609949e-02 2.771736e-02 0.014619883
## 11:              gill-spacing=close 8.049881e-03 2.265660e-02 0.090643275
## 12: stalk-surface-below-ring=scaly 5.591321e-03 1.938208e-02 0.014619883
## 13:         spore-print-color=white 3.950073e-03 1.272012e-02 0.023391813
## 14:                   odor=pungent 2.212515e-03 6.118102e-03 0.008771930
## 15:           population=clustered 1.738157e-03 9.916562e-03 0.038011696
## 16:             stalk-root=missing 1.567230e-03 5.523625e-04 0.026315789
```

Gradient Boosting with xgboost

```
library(Ckmeans.1d.dp)
xgb.plot.importance(importance_matrix = import_mat)
```



Bagging & RandomForest

- Boostrap aggregating:

  Boostrap 표본을 이용하여 모형을 적합.

  적합된 모형을 이용하여 예측값을 구하고 그 평균이나 최빈값을 이용하여 예측.

- Random Forest:

  Bagging와 흡사한 방식으로 적합하지만 전체 변수 p을 사용하지 않고 $m \approx \sqrt{p}$개의 변수를 랜덤하게 선택하여 이 변수들에서만 나무 모형을 적합.

Bagging with R

```
library(randomForest)

## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.

set.seed(1)
p = dim(as.matrix(train$data))[2]
bag.fit = randomForest(x= as.matrix(train$data), y = as.factor(train$label),
                       mtry = p, ntree = 10, importance = T)
bag.fit

##
## Call:
##  randomForest(x = as.matrix(train$data), y = as.factor(train$label),      ntree = 10, mt
##                Type of random forest: classification
##                      Number of trees: 10
## No. of variables tried at each split: 126
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##      0    1 class.error
## 0 3334    0           0
## 1    0 3108           0

mean(test$label != predict(bag.fit, test$data))

## [1] 0
```

Bagging with R

- Validation for `ntree`

```
set.seed(1)
val.err = c()
for (i in candidates){
  bag.val = randomForest(x= as.matrix(train$data)[-val.ind,],
                         y = as.factor(train$label)[-val.ind],
                         mtry = p, ntree = i, importance = F)
  pred.val = predict(bag.val, as.matrix(train$data)[val.ind,])
  val.err = c(val.err, mean(as.factor(train$label)[val.ind] != pred.val))
}
val.err
```

```
##  [1] 0.0009213759 0.0009213759 0.0009213759 0.0009213759 0.0009213759
##  [6] 0.0009213759 0.0009213759 0.0009213759 0.0009213759 0.0009213759
## [11] 0.0009213759 0.0009213759 0.0009213759 0.0009213759 0.0009213759
## [16] 0.0009213759 0.0009213759 0.0009213759 0.0009213759 0.0009213759
## [21] 0.0009213759 0.0009213759 0.0009213759 0.0009213759 0.0009213759
## [26] 0.0009213759 0.0009213759 0.0009213759 0.0009213759 0.0009213759
```

```
which.min(val.err)
```

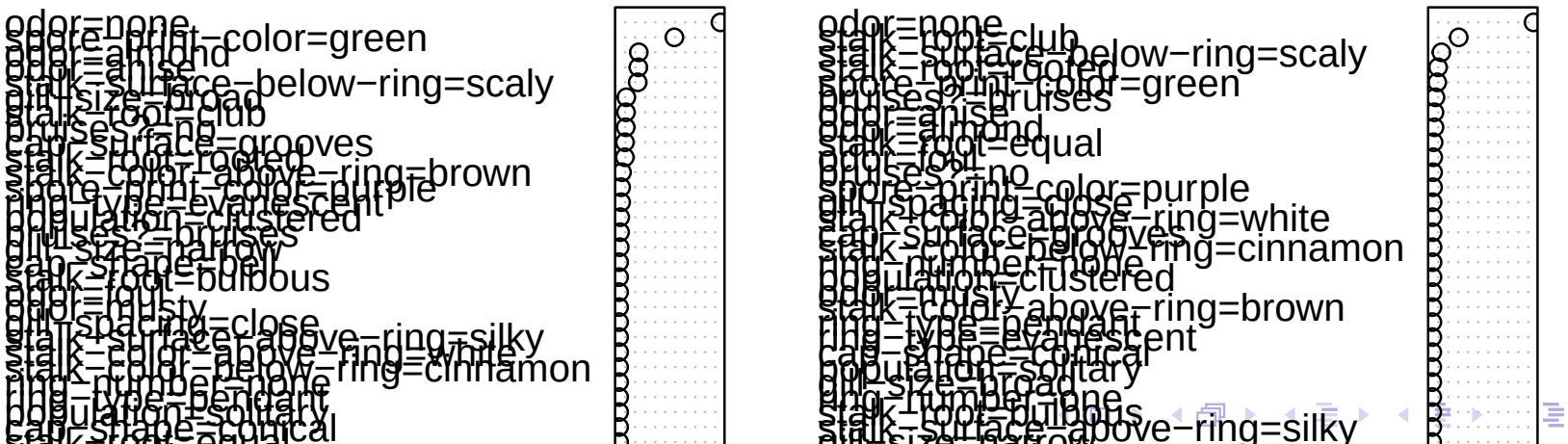
```
## [1] 1
```

Bagging with R

- 변수의 중요도 평가

```
bag.fit = randomForest(x= as.matrix(train$data), y = as.factor(train$label),
                       mtry = p, ntree = candidates[which.min(val.err)], importance = T)
pred = predict(bag.fit, as.matrix(test$data))
mean(test$label != pred)

## [1] 0

varImpPlot(bag.fit)
```

bag.fit



RandomForest with R

```
set.seed(1)
rf.fit = randomForest(x= as.matrix(train$data), y = as.factor(train$label),
                      mtry = floor(sqrt(p)), ntree = 10, importance = T)
rf.fit

##
## Call:
##  randomForest(x = as.matrix(train$data), y = as.factor(train$label),      ntree = 10, mt
##                Type of random forest: classification
##                      Number of trees: 10
## No. of variables tried at each split: 11
##
##         OOB estimate of  error rate: 0.03%
## Confusion matrix:
##      0    1  class.error
## 0 3339    1 0.0002994012
## 1    1 3107 0.0003217503

mean(test$label != predict(rf.fit, test$data))

## [1] 0
```

RandomForest with R

- Validation for `ntree`

```
set.seed(1)
val.err = c()
for (i in candidates){
  rf.val = randomForest(x= as.matrix(train$data)[-val.ind,],
                        y = as.factor(train$label)[-val.ind],
                        mtry = floor(sqrt(p)), ntree = i, importance = F)
  pred.val = predict(rf.val, as.matrix(train$data)[val.ind,])
  val.err = c(val.err, mean(as.factor(train$label)[val.ind] != pred.val))
}
val.err
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
which.min(val.err)
```

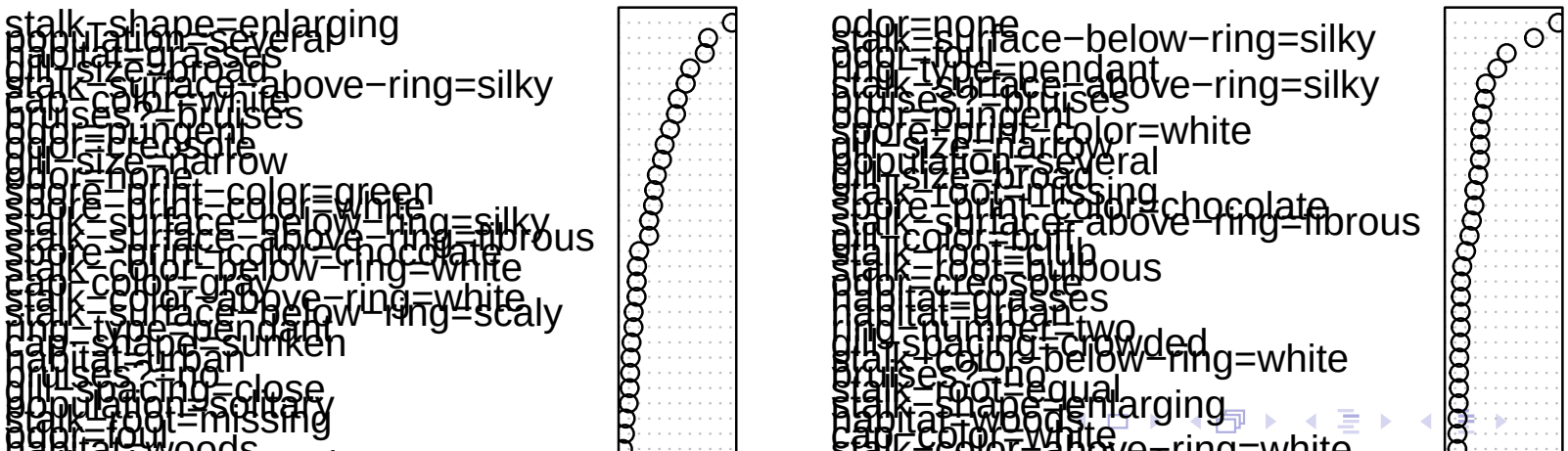
```
## [1] 1
```

RandomForest with R

- 변수의 중요도 평가

```
rf.fit = randomForest(x= as.matrix(train$data), y = as.factor(train$label),
                      mtry = floor(sqrt(p)), ntree = candidates[which.min(val.err)], import
pred = predict(rf.fit, as.matrix(test$data))
mean(test$label != pred)

## [1] 0

varImpPlot(rf.fit)
```

rf.fit



목차

Case-control sampling: 표본 추출 방법

- 표본 추출 방법 : 자료의 불균형으로 생기는 문제를 해결하기 위한 가장 쉬운 접근 방법 중 하나.
- 과다 추출법 vs. 과소 추출법
- 실습에서는 앙상블 기법을 이용한 보정된 과소 추출법을 다룬다.
- 알고리즘
  1. 자료를 정상 자료 $S_{\max}$와 비정상 자료 $S_{\min}$로 분리한다.
  2. 정상 자료 중 $|S_{\min}|$개 만큼의 자료를 랜덤하게 추출한다.
  3. 2에서 추출한 자료와 $S_{\min}$을 이용하여 분류모형을 생성한다.
  4. 2-3을 충분히 반복한 후, 반복을 통해 만들어진 여러 모형을 합쳐서 최종 앙상블 모형을 생성한다.

Case-control sampling: 표본 추출 방법

- buytest 자료.

```
set.seed(1)
train_ind = sample(1:nrow(buydata), size = floor(nrow(buydata)*0.7),
                   replace = F)
train = as.data.frame(buydata[train_ind,])
test = as.data.frame(buydata[-train_ind,])
X_train = buydata[train_ind, -1]
y_train = buydata[train_ind, 1]
X_test = buydata[-train_ind, -1]
y_test = buydata[-train_ind, 1]
```

Case-control sampling: 표본 추출 방법

- buytest 자료에서 "RESPOND=0"인 자료를 "RESPOND=1"인 자료 수의 2배만큼 랜덤하게 추출. 이 때 반복횟수는 50으로 설정.

```
n1 = sum(y_train == 1)
set.seed(6)
cc_ind = sample(1:sum(y_train == 0),
                size = 2*n1, replace = F)
cc_data = rbind(train[y_train == 1,],
                train[y_train == 0,][cc_ind,])
cc_data = cc_data[sample(1:nrow(cc_data), nrow(cc_data), replace = F),]
table(cc_data$RESPOND)
```

```
##
##   0   1
## 972 486
```

```
dim(cc_data)
```

```
## [1] 1458   27
```

Case-control sampling: 표본 추출 방법

- buytest 자료에서 "RESPOND=0"인 자료를 "RESPOND=1"인 자료 수의 2배만큼 랜덤하게 추출. 이 때 반복횟수는 50으로 설정.

```
tol_iter = 50
beta_list = list()
set.seed(6)
for (iter in 1:tol_iter){
  cc_ind = sample(1:sum(y_train == 0),
                  size = 2*n1, replace = F)
  cc_data = rbind(train[y_train == 1,],
                  train[y_train == 0,][cc_ind,])
  beta_list[[iter]] = coef(glm(RESPOND~., data = cc_data, family = 'binomial'))
}
train_pred_probs = sapply(1:tol_iter,
                          function(iter) 1/(1+exp(-cbind(1, X_train)%*%
                                                    beta_list[[iter]])))
train_pred_prob = rowMeans(train_pred_probs)
test_pred_probs = sapply(1:tol_iter,
                          function(iter) 1/(1+exp(-cbind(1, X_test)%*%
                                                    beta_list[[iter]])))
test_pred_prob = rowMeans(test_pred_probs)
```

Case-control sampling: 표본 추출 방법

- 보정된 과소 추출법 이용, 로지스틱 회귀분석의 학습자료와 예측자료에서 AUC.

```
auc_res(newx = X_train, newy = y_train, pred_prob = train_pred_prob)
## [1] 0.6646239
auc_res(newx = X_test, newy = y_test, pred_prob = test_pred_prob)
## [1] 0.6444104
```

- (직접해보기) 위와 같은 보정된 과소 추출법을 전진선택법 (AIC), Ridge, LASSO, RandomForest, Boosting에 대해 실행하고, AUC 비교 여러 절단값 선택 방법들을 시행하여 예측성능 비교.