

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

로지스틱 회귀 분석

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

- F 로 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	
##	Mean :20.04	Mean :0.1889	Mean :0.2889	Mean :0.2474	

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)
```

[illegible]

로지스틱 회귀분석 with R

A boxplot showing FICO scores for two groups: RESPOND = 0 (red box) and RESPOND = 1 (teal box). The y-axis represents FICO scores from 600 to 800. The x-axis is labeled RESPOND. For RESPOND = 0, the median is approximately 695, with a box from 675 to 715 and whiskers from 600 to 800. For RESPOND = 1, the median is approximately 690, with a box from 670 to 710 and whiskers from 610 to 770. Individual data points are overlaid on the boxplots.

로지스틱 회귀분석 with R

The figure displays four plots illustrating the relationship between the variable **RESPOND** (categorical, with levels 0 and 1) and various response variables.

- BUY6:** A dot plot showing the relationship between **RESPOND** and **BUY6**. The y-axis ranges from 0.0 to 2.0. For **RESPOND** = 0, there are 10 dots at 1.0 and 10 dots at 2.0. For **RESPOND** = 1, there are 10 dots at 1.0 and 10 dots at 2.0.
- BUY12:** A dot plot showing the relationship between **RESPOND** and **BUY12**. The y-axis ranges from 0.0 to 2.0. For **RESPOND** = 0, there are 10 dots at 1.0 and 10 dots at 2.0. For **RESPOND** = 1, there are 10 dots at 1.0 and 10 dots at 2.0.
- BUY18:** A bar plot showing the relationship between **RESPOND** and **BUY18**. The y-axis ranges from 0 to 3. For **RESPOND** = 0, the bar height is 1.0. For **RESPOND** = 1, the bar height is 1.0.
- VALUE24:** A box plot showing the relationship between **RESPOND** and **VALUE24**. The y-axis ranges from 0 to 1250. For **RESPOND** = 0, the median is approximately 200, the interquartile range (IQR) is from approximately 100 to 300, and the range is from approximately 50 to 1250. For **RESPOND** = 1, the median is approximately 250, the IQR is from approximately 150 to 350, and the range is from approximately 100 to 1000.

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
##  ORGSRC      ORGSRCI   ORSRCO   ORSRCR   ORSRCR   ORSRCU
## -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?

- 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
```

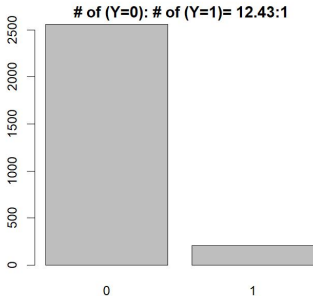
- ```
par(mfrow = c(1,2))
barplot(table(y_train),
 main = paste("The distribution of RESPOND in Training set
 \n # of (Y=0): # of (Y=1)=",
 paste(round(table(y_train)/sum(y_train == 1),2),
 collapse = ":")))

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

### The distribution of RESPOND in Testset





## 모형의 평가

- 연속형 출력변수의 경우  $R^2$  값이나 AIC, BIC 등을 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

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

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

- $F_1$  스코어 : precision, recall의 조화평균.

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

## 사후 확률

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

```
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

- ◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ≡ ↺ 🔍 ↻



## 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

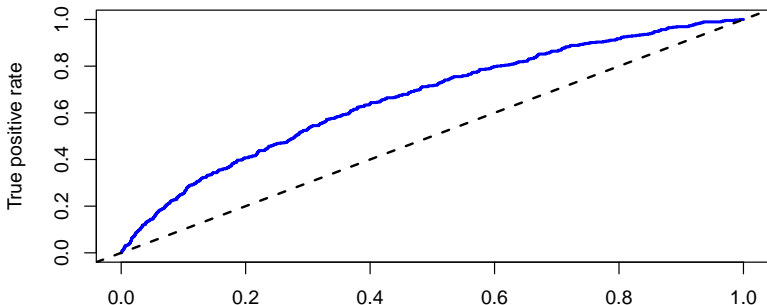
[[1]]
[1] 0.6627131
```

## 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 curve for Train data**  
**AUC: 0.6627**



- 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

- 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
```

- ```
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
```

- ```
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
```







## 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  $\beta$ 의 불편 추정량이지만 분산이 매우 클수가 있다.

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

- Ridge

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

- Lasso

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

- Lasso는 shrinkage뿐만 아니라 변수 선택까지 해준다는 특징이 있다.
- tuning parameter  $\lambda$ 는 cross validation등을 이용해 추정한다.



## 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
LOCG 0.875142e-39 0.452842e-04 0.0700540812
```

## 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
VALUE24
```

```
VALIEF24
```

## 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 AUC를 계산하여 비교.
- 학습자료에서의 절단값을 선택하는 여러가지 기준을 이용하여 예측자료에서 예측 성능 비교.

## 방법들 비교

- 모형별 회귀계수 저장.

[illegible]

- 모형별 AUC 값 비교.

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ▶ ↺ 🔍 ↻

## 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   |





## 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.

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

여기서  $|T|$ 은 나무  $T$ 의 끝마디 개수.

[illegible]

```
library(rpart)
tree.buydata = rpart(as.factor(RESPOND)~., data = train)
tree.buydata

n= 6454
##
node), split, n, loss, yval, (yprob)
* denotes terminal node
##
1) root 6454 486 0 (0.92469786 0.07530214) *
```

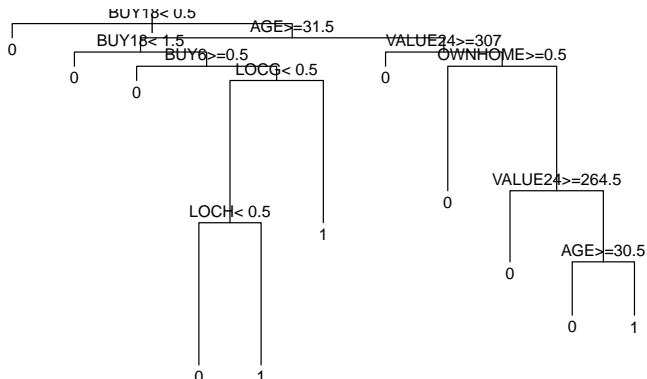
- `rpart.control`

- minbucket: terminal node에서의 최소 관측치의 수.
- cp: Complexity parameter ( $\alpha$ ), 기본값은 0.01로 설정.

```
n= 6454
##
node), split, n, loss, yval, (yprob)
* denotes terminal node
##
1) root 6454 486 0 (0.92469786 0.07530214)
2) BUY18< 0.5 4527 274 0 (0.93947427 0.06052573) *
3) BUY18>=0.5 1927 212 0 (0.88998443 0.11001557)
6) AGE>=31.5 1768 170 0 (0.90384615 0.09615385)
12) BUY18< 1.5 1505 124 0 (0.91760797 0.08239203) *
13) BUY18>=1.5 263 46 0 (0.82509506 0.17490494)
26) BUY6>=0.5 175 19 0 (0.89142857 0.10857143) *
27) BUY6< 0.5 88 27 0 (0.69318182 0.30681818)
54) LOCG< 0.5 78 18 0 (0.76923077 0.23076923)
108) LOCH< 0.5 62 6 0 (0.90322581 0.09677419) *
109) LOCH>=0.5 16 4 1 (0.25000000 0.75000000) *
55) LOCG>=0.5 10 1 1 (0.10000000 0.90000000) *
7) AGE< 31.5 159 42 0 (0.73584906 0.26415094)
14) VALUE24>=307 111 19 0 (0.82882883 0.17117117) *
15) VALUE24< 307 48 23 0 (0.52083333 0.47916667)
30) OWNHOME>=0.5 15 3 0 (0.80000000 0.20000000) *
31) OWNHOME< 0.5 33 13 1 (0.39393939 0.60606061)
62) VALUE24>=264.5 10 3 0 (0.70000000 0.30000000) *
63) VALUE24< 264.5 23 6 1 (0.26086957 0.73913043)
126) AGE>=30.5 7 2 0 (0.71428571 0.28571429) *
127) AGE< 30.5 16 1 1 (0.06250000 0.93750000) *
```

## buydata with decision tree

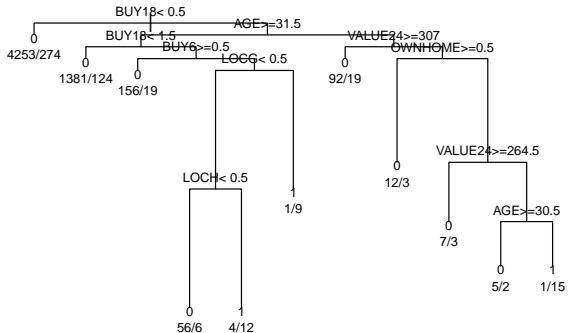
```
plot(tree.buydata)
text(tree.buydata, cex = 0.8)
```



○○○○○●○○○○○○○○○○○○○○○○○○○○○○○○○○○○○○

## buydata with decision tree

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



- cost-complexity을 최소화하는 모형 선택

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ▶ ↺ 🔍 ↻

## Pruning, 가지치기

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

나무 T의 오분류율 = rel error  $\times$  Root node error

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



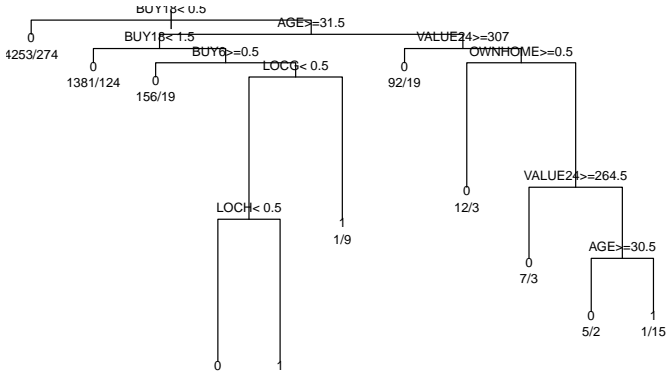
## Pruning, 가지치기

```
prune.buydata = prune(tree.buydata, cp = 0.005)
prune.buydata

n= 6454
##
node), split, n, loss, yval, (yprob)
* denotes terminal node
##
1) root 6454 486 0 (0.92469786 0.07530214)
2) BUY18< 0.5 4527 274 0 (0.93947427 0.06052573) *
3) BUY18>=0.5 1927 212 0 (0.88998443 0.11001557)
6) AGE>=31.5 1768 170 0 (0.90384615 0.09615385)
12) BUY18< 1.5 1505 124 0 (0.91760797 0.08239203) *
13) BUY18>=1.5 263 46 0 (0.82509506 0.17490494)
26) BUY6>=0.5 175 19 0 (0.89142857 0.10857143) *
27) BUY6< 0.5 88 27 0 (0.69318182 0.30681818)
54) LOCG< 0.5 78 18 0 (0.76923077 0.23076923)
108) LOCH< 0.5 62 6 0 (0.90322581 0.09677419) *
109) LOCH>=0.5 16 4 1 (0.25000000 0.75000000) *
55) LOCG>=0.5 10 1 1 (0.10000000 0.90000000) *
7) AGE< 31.5 159 42 0 (0.73584906 0.26415094)
14) VALUE24>=307 111 19 0 (0.82882883 0.17117117) *
15) VALUE24< 307 48 23 0 (0.52083333 0.47916667)
30) OWNHOME>=0.5 15 3 0 (0.80000000 0.20000000) *
31) OWNHOME< 0.5 33 13 1 (0.39393939 0.60606061)
62) VALUE24>=264.5 10 3 0 (0.70000000 0.30000000) *
63) VALUE24< 264.5 23 6 1 (0.26086957 0.73913043)
126) AGE>=30.5 7 2 0 (0.71428571 0.28571429) *
```

## Pruning, 가지치기

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



## Pruning, 가지치기

- CV를 이용하여 적절한 가지치기 찾기

```
set.seed(1)
K = 10
sample.ind = sample(1:K, size = nrow(train), replace = T)
cp = seq(from = 0.001, to = 0.01, length = 30)
error = matrix(0, nrow = length(cp), K)
for (i in 1:length(cp)){
 for (j in 1:K){
 tmp = rpart(as.factor(RESPOND)~., data = train[sample.ind != j,], cp = cp[i])
 error[i,j] = sum(predict(tmp, train[sample.ind == j,], type = "class")
 != train[sample.ind == j,]$RESPOND)/sum(sample.ind == j)
 }
}
rowMeans(error)

[1] 0.07839427 0.07652141 0.07451454 0.07375332 0.07375332 0.07358573
[7] 0.07358573 0.07357947 0.07264216 0.07280013 0.07280013 0.07309599
[13] 0.07400804 0.07385812 0.07402935 0.07402935 0.07448482 0.07479879
[19] 0.07479879 0.07494985 0.07494985 0.07510684 0.07510684 0.07510684
[25] 0.07510684 0.07510684 0.07510684 0.07510684 0.07510684 0.07510684

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

[1] 0.003482759
```

## Bagging & RandomForest

- Bootstrap aggregating:

Bootstrap 표본을 이용하여 모형을 적합.

적합된 모형을 이용하여 예측값을 구하고 그 평균이나 최빈값을 이용하여 예측.

- Random Forest:

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

100

```
library(randomForest)

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

set.seed(1)
p = dim(as.matrix(train))[2] - 1
bag.fit = randomForest(x= X_train, y = as.factor(y_train),
 mtry = p, ntree = 500, importance = T)
bag.fit

##
Call:
randomForest(x = X_train, y = as.factor(y_train), ntree = 500, mtry = p, importance = T)
Type of random forest: classification
Number of trees: 500
No. of variables tried at each split: 26
OOB estimate of error rate: 7.34%
Confusion matrix:
0 1 class.error
0 5956 12 0.002010724
1 462 24 0.950617284

mean(y_test != predict(bag.fit, X_test))

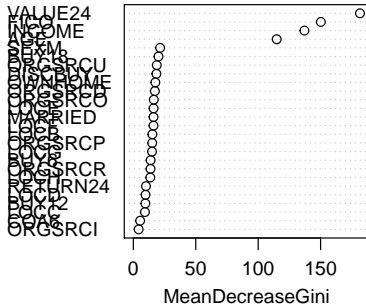
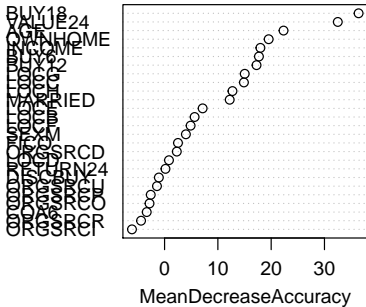
[1] 0.07228045
```

- 변수의 층

- 변수의 중요도 평가

```
varImpPlot(bag.fit)
```

bag.fit



## Bagging with R

- 변수의 중요도 평가

```
importance(bag.fit)
```

| ##         |            | 0             | 1 | MeanDecreaseAccuracy | MeanDecreaseGini |
|------------|------------|---------------|---|----------------------|------------------|
| ## AGE     | 18.9826688 | 14.6246518770 |   | 22.3223230           | 114.837041       |
| ## INCOME  | 18.3851637 | 0.0002270112  |   | 17.9788551           | 137.058983       |
| ## SEXM    | 4.5603124  | -1.8420525126 |   | 3.9993910            | 21.444837        |
| ## MARRIED | 12.6504742 | -2.1593780914 |   | 12.2197085           | 16.153212        |
| ## FICO    | 2.2196027  | 1.3041227799  |   | 2.5152368            | 150.237245       |
| ## OWNHOME | 18.7644969 | 5.7492548241  |   | 19.5325807           | 17.629495        |
| ## LOCB    | 5.2300338  | 2.2530010684  |   | 5.6379442            | 15.255700        |
| ## LOCC    | 11.8695461 | 10.5873086116 |   | 14.8870146           | 9.303287         |
| ## LOCD    | -0.6570701 | 5.4801448448  |   | 0.8261815            | 9.798880         |
| ## LOCE    | 6.1223763  | 5.4735396583  |   | 7.1441480            | 16.190131        |
| ## LOCF    | 4.1955880  | 2.6770841613  |   | 4.8834447            | 16.136970        |
| ## LOCG    | 13.5552580 | 7.1892750615  |   | 15.0403042           | 14.968393        |
| ## LOCH    | 8.6116017  | 13.6523608078 |   | 12.7375233           | 13.569721        |
| ## BUY6    | 18.2756634 | -1.0401812334 |   | 17.7041941           | 13.832496        |
| ## BUY12   | 18.3548041 | -9.8893877817 |   | 17.2530306           | 9.635502         |
| ## BUY18   | 33.3774354 | 10.7827845382 |   | 36.4030778           | 20.155567        |
| ## VALUE24 | 32.9547277 | -8.5112638027 |   | 32.5006302           | 181.532130       |
| ## ORGSRCD | 1.7016260  | 2.0740379226  |   | 2.2927006            | 17.266872        |
| ## ORGSRCI | -6.2061533 | -0.6357188717 |   | -6.1153098           | 4.241807         |
| ## ORGSRCO | -2.1487036 | -2.5343330529 |   | -2.8238635           | 16.411465        |
| ## ORGSRCP | -2.5942259 | -0.1941494487 |   | -2.5937897           | 15.120350        |
| ## ORGSRCR | -5.7943054 | 3.4525476575  |   | -4.4135749           | 13.758309        |
| ## ORGSRCU | -2.2865741 | 2.5669121751  |   | -1.4271582           | 18.800321        |

## RandomForest with R

```
set.seed(1)
rf.fit = randomForest(x= X_train, y = as.factor(y_train),
 mtry = floor(sqrt(p)), ntree = 500, importance = T)

rf.fit

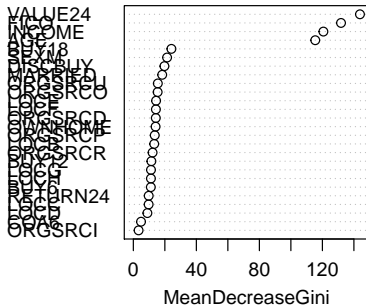
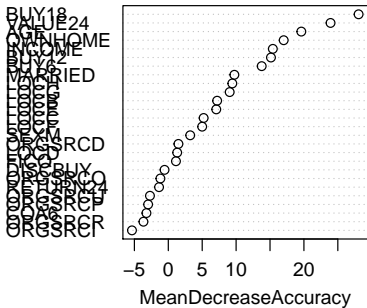
##
Call:
randomForest(x = X_train, y = as.factor(y_train), ntree = 500, mtry = floor(sqrt(p))
Type of random forest: classification
Number of trees: 500
No. of variables tried at each split: 5
##
OOB estimate of error rate: 7.56%
Confusion matrix:
0 1 class.error
0 5965 3 0.000502681
1 485 1 0.997942387

mean(y_test != predict(rf.fit, X_test))

[1] 0.07372606
```



- 변수의 중요도 평가



- 변수의 중요도 평가

- 변수의 중요도 평가

| ##         |            | 0          | 1 | MeanDecreaseAccuracy | MeanDecreaseGini |
|------------|------------|------------|---|----------------------|------------------|
| ## AGE     | 16.3672154 | 13.6951214 |   | 19.6153828           | 115.466646       |
| ## INCOME  | 15.7512494 | 0.8339830  |   | 15.4144256           | 120.628491       |
| ## SEXM    | 3.2516800  | 0.1948610  |   | 3.2312540            | 21.421611        |
| ## MARRIED | 9.9829210  | 0.1232558  |   | 9.7424958            | 18.320485        |
| ## FICO    | 0.7161066  | 1.6229367  |   | 1.1280775            | 131.817719       |
| ## OWNHOME | 15.7906106 | 6.4855615  |   | 17.0166659           | 14.118126        |
| ## LOCB    | 7.0494952  | 1.8047641  |   | 7.2034364            | 13.249472        |
| ## LOCC    | 2.8787144  | 8.5306199  |   | 5.2094749            | 9.711090         |
| ## LOCD    | 0.3096278  | 3.5760325  |   | 1.3093823            | 8.915793         |
| ## LOCE    | 6.6207543  | 2.4146180  |   | 7.0729732            | 14.442517        |
| ## LOCF    | 4.2829281  | 3.1402135  |   | 4.9956141            | 14.326898        |
| ## LOCG    | 7.5276433  | 5.5971058  |   | 9.0524397            | 11.202337        |
| ## LOCH    | 8.7072556  | 3.8341445  |   | 9.4403879            | 11.169681        |
| ## BUY6    | 13.9196678 | -2.8318499 |   | 13.7730200           | 11.108509        |
| ## BUY12   | 15.4747366 | -8.0371611 |   | 15.1383148           | 11.443821        |
| ## BUY18   | 25.5911486 | 11.0810982 |   | 28.0615451           | 24.159079        |
| ## VALUE24 | 24.5793044 | -6.5161863 |   | 23.9248679           | 143.842073       |
| ## ORGSRCD | 0.5875620  | 3.0738652  |   | 1.4927652            | 14.266742        |
| ## ORGSRCI | -5.0862227 | -1.7150918 |   | -5.3562680           | 3.335523         |
| ## ORGSRCO | -1.5530348 | 1.1668484  |   | -1.1657044           | 15.425394        |
| ## ORGSRCP | -3.1005284 | -0.1356451 |   | -2.9760362           | 13.581794        |
| ## ORGSRCR | -3.6472426 | -0.3798578 |   | -3.6672000           | 12.208561        |
| ## ORGSRCU | -2.7697821 | -0.1755594 |   | -2.7032758           | 15.538865        |

## 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

```
install.packages("xgboost")
```

## Gradient Boosting with xgboost

```
training
library(xgboost)

boost.fit = xgboost(data = X_train, label = y_train, max.depth = 2,
 eta = 0.1, nround = 2, objective = "binary:logistic")

[1] train-error:0.075302
[2] train-error:0.075302

testing
pred = predict(boost.fit, X_test)
mean(y_test != round(pred))

[1] 0.07444886
```

## Gradient Boosting with xgboost

- Validation for nround

```
validation set
set.seed(123)
val.ind = sample(1:nrow(X_train), size = floor(nrow(X_train)*0.3))
val.err = c()
candidates = seq(from = 50, to = 250, by = 20)
for (i in candidates){
 boost.val = xgboost(data = X_train[-val.ind,], label = y_train[-val.ind], max.depth = 2,
 eta = 0.1, nround = i, objective = "binary:logistic", verbose = 0)
 pred.val = predict(boost.val, X_train[val.ind,])
 val.err = c(val.err, mean(y_train[val.ind] != round(pred.val)))
}
val.err

[1] 0.06869835 0.06869835 0.06869835 0.06869835 0.06869835 0.06818182
[7] 0.06818182 0.06818182 0.06818182 0.06818182 0.06818182

which.min(val.err)

[1] 6
```

## Gradient Boosting with xgboost

```
boost.fit = xgboost(data = X_train, label = y_train, max.depth = 2,
 eta = 0.1, nround = candidates[which.min(val.err)], objective = "binary:logistic")
pred = predict(boost.fit, X_test)
mean(y_test != round(pred))

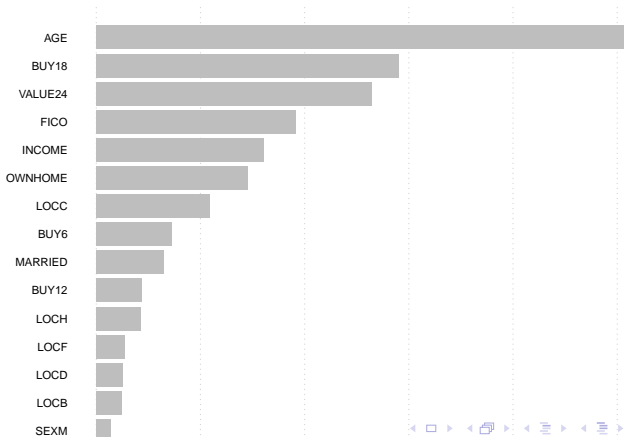
[1] 0.07408746

relative influence
import_mat = xgb.importance(colnames(X_train), model = boost.fit)
print(import_mat)
```

| ##     | Feature | Gain         | Cover        | Frequency   |
|--------|---------|--------------|--------------|-------------|
| ## 1:  | AGE     | 0.2539029501 | 0.2204086531 | 0.210526316 |
| ## 2:  | BUY18   | 0.1449728584 | 0.1774062417 | 0.091533181 |
| ## 3:  | VALUE24 | 0.1320686033 | 0.1112637172 | 0.151029748 |
| ## 4:  | FICO    | 0.0956610425 | 0.1311882671 | 0.151029748 |
| ## 5:  | INCOME  | 0.0801702893 | 0.0911149501 | 0.100686499 |
| ## 6:  | OWNHOME | 0.0726177020 | 0.0909539245 | 0.054919908 |
| ## 7:  | LOCC    | 0.0544839982 | 0.0517878235 | 0.043478261 |
| ## 8:  | BUY6    | 0.0362888650 | 0.0028422337 | 0.025171625 |
| ## 9:  | MARRIED | 0.0325740174 | 0.0401201337 | 0.036613272 |
| ## 10: | BUY12   | 0.0219128876 | 0.0100172614 | 0.020594966 |
| ## 11: | LOCH    | 0.0212842253 | 0.0065609941 | 0.022883295 |
| ## 12: | LOCF    | 0.0137657579 | 0.0054418662 | 0.016018307 |
| ## 13: | LOCD    | 0.0127332803 | 0.0193484966 | 0.022883295 |
| ## 14: | LOCB    | 0.0120800714 | 0.0168521233 | 0.022883295 |
| ## 15: | SEXM    | 0.0068540885 | 0.0043614005 | 0.009153318 |
| ## 16: | ORGSRCU | 0.0034697666 | 0.0131104784 | 0.011441648 |

## Gradient Boosting with xgboost

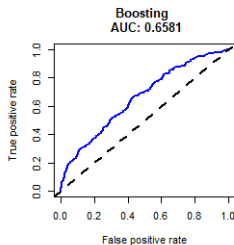
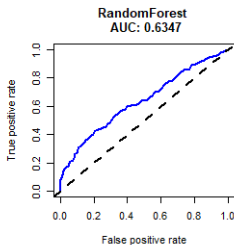
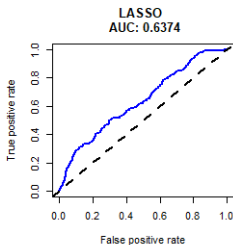
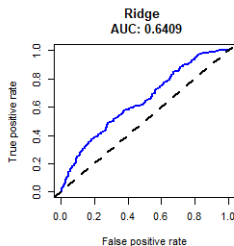
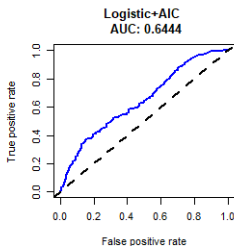
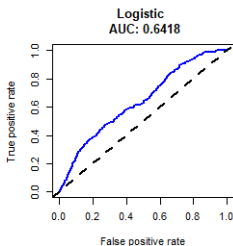
```
library(Ckmeans.1d.dp)
xgb.plot.importance(importance_matrix = import_mat)
```







- 모형별 ROC curve 및 AUC 값 비교.



- ```
cut_sel = matrix(0, nrow = 6, ncol = 3)
pred_probs_train = sapply(1:4, function(i) 1/(1+exp(-cbind(1, X_train)%*%beta_hat[,i])))
pred_probs_train = cbind(pred_probs_train, predict(rf.fit, X_train, type = 'prob')[,2],
                          predict(boost.fit, X_train))

for (i in 1:6){
  cutoff_out = t(sapply(1:length(cutoff_can),
                        function(j) cutoff_res(newx = X_train, response = y_train,
                                                cutoff = cutoff_can[j], pred_prob = pred_probs_train[,j]))))
  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(1:6, function(i) cutoff_res(newx = X_test, response = y_test,
                                             cutoff = cut_sel[i, 1], pred_prob = pred_probs_test[,i])))),
       nrow = 6,
       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 |
| ## RandomForest | 0.28 | 0.0763 | 0.0922 | 0.9906 | 0.1526 |
| ## Boosting | 0.28 | 0.0752 | 0.0485 | 0.9953 | 0.0877 |

- 학습자료에서의 F_1 스코어를 최대화 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
## RandomForest	0.28	0.0763	0.0922	0.9906	0.1526
## Boosting	0.12	0.1493	0.2573	0.8985	0.2042

4. Case-control sampling

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.