## 타켓마케팅 실습

로지스틱 회귀 분석, 변수선택, 앙상블

## 목차

- 1. 로지스틱 회귀 분석 로지스틱 회귀 모형 절단값과 오분류율
- 2. 변수선택 및 벌점화 기법 Variable Selection 벌점화 기법 모형 비교
- 3. 앙상블 의사결정나무 배깅 및 랜덤포레스트 부스팅 모형 비교
- 4. Case-control sampling

# 목차

- 1. 로지스틱 회귀 분석 로지스틱 회귀 모형 절단값과 오분류율
- 2. 변수선택 및 벌점화 기법 Variable Selection 벌점화 기법 모형 비교
- 3. 앙상블 의사결정나무 배강 및 랜덤포레스트 부스팅 무형 비교
- 4. Case-control sampling

## 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

### • 자료 불러오기

```
buytest = read.table("buytest.txt", header = T)
dim(buvtest)
## [1] 10000
                26
summary(buytest)
##
            ID
                        RESPOND
                                            AGE
                                                           INCOME
                                                                        SEX
##
   000054889: 1
                     Min.
                            :0.0000
                                      Min. :18.00
                                                      Min. : 15.00
                                                                         : 234
   000219612:
                     1st Qu.:0.0000
                                     1st Qu.:38.00
                                                     1st Qu.: 35.00
                                                                        F:4489
##
##
   001044039:
                     Median :0.0000
                                      Median :44.00
                                                      Median : 50.00
                                                                        M:5277
##
   001079946:
                     Mean
                            :0.0767
                                      Mean
                                              :44.56
                                                       Mean
                                                            : 47.95
##
   001108462:
                     3rd Qu.:0.0000
                                      3rd Qu.:51.00
                                                       3rd Qu.: 61.00
##
   001109024: 1
                     Max.
                            :1.0000
                                      Max.
                                             :75.00
                                                       Max.
                                                             :114.00
                                      NA's
                                              :234
                                                       NA's
                                                              :234
##
    (Other) :9994
##
       MARRIED
                          FICO
                                        OWNHOME
                                                            LOC
   Min.
           :0.0000
                     Min.
                            :577.0
                                     Min.
                                             :0.0000
                                                              :2261
   1st Qu.:0.0000
                     1st Qu.:676.0
                                     1st Qu.:0.0000
                                                              :2168
   Median :1.0000
                     Median :695.0
                                     Median :0.0000
                                                              :1828
##
   Mean
           :0.5842
                     Mean
                            :694.3
                                     Mean
                                             :0.3341
                                                              :1093
   3rd Qu.:1.0000
                     3rd Qu.:714.0
                                     3rd Qu.:1.0000
                                                              : 950
##
   Max.
           :1.0000
                     Max.
                            :800.0
                                     Max.
                                             :1.0000
                                                              : 585
##
           :234
                     NA's
                            :39
                                             :234
   NA's
                                     NA's
                                                       (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
           :20.34
                    Mean
                           :0.1283
                                     Mean
                                             :0.2099
                                                       Mean
                                                              :0.3471
   3rd Qu.:20.00
                    3rd Qu.:0.0000
                                     3rd Qu.:0.0000
                                                       3rd Qu.:1.0000
```

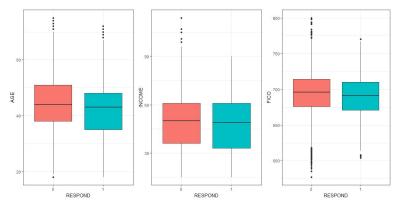
#### • 자료 전처리

```
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
```

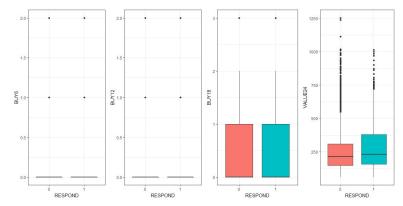
## • 자료 전처리

hea	ad	(buyda	ıta)															
##		RESPO	ND .	AGE	INCO	ME S	SEXM	MARRII	ΞD	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	BUY	6 BI	JY12	BUY:	18 V	ALUE24	OI	RGSRCD	ORGSRCI	ORG	SRCO	ORGSRO	P OR	GSRCR	ORGS	RCU
##	1	0		1	1		1	318		0	(	)	1		0	0		0
	2	0		0	0		0	83		0		)	0		0	1		0
##	3	0		1	1		1	265		1	(	)	0		0	0		0
##	4	0		0	0		1	448		0		)	1		0	0		0
	5	0		0	0		0	161		0	(	)	0		0	1		0
##	6	0		0	0		0	250		0		)	0		0	0		0
##		DISCE	BUY :	RETU														
##			1		0	)	0											
	2		0		0		0											
	3		0		0	)	0											
##	4		1		0		0											
##	5		0		0		0											
##	6		0		1		0											

### • 자료 분포 살펴보기1



### • 자료 분포 살펴보기2



#### • 모형 적합

• glm(formula, family, data)

```
logit = glm(RESPOND~., data = as.data.frame(buvdata), 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:^^I
                        4915
## Residual Deviance: 4687 ^^IAIC: 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.

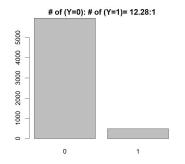
• Buydata training set을 7, test set을 3으로 나눈다.

```
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 = buvdata[train ind, -1]
v_train = buydata[train_ind, 1]
X_test = buydata[-train_ind, -1]
v_test = buydata[-train_ind, 1]
dim(X train)
## [1] 6454
            26
dim(X_test)
## [1] 2767
             26
```

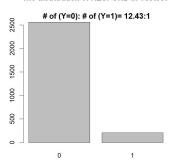
### • 학습자료와 예측자료의 반응변수 분포

### • 학습자료와 예측자료의 반응변수 분포

The distribution of RESPOND in Training set



#### The distribution of RESPOND in Testset



## 모형의 평가

- 연속형 출력변수의 경우 R<sup>2</sup> 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

			예측급	lt .
		0	1	합계
	0	<b>a</b> 00	a <sub>01</sub>	<b>a</b> 0.
실제값	1	1 $a_{10}$ $a_{11}$		$a_{1.}$
	합계	<b>a</b> .0	a.1	a

Table: 분류표

• 정분류율 
$$= (a_{00} + a_{11})/a$$

• 오분류율 = 
$$(a_{10} + a_{01})/a$$

• 특이도 = 
$$a_{00}/a_{0.}$$

## 불균형 자료 모형 평가를 위한 척도

			예측급	it .
		0	1	합계
	0	<b>a</b> 00	a <sub>01</sub>	<b>a</b> 0.
실제값	1	<b>a</b> <sub>10</sub>	a <sub>11</sub>	a <sub>1.</sub>
	합계	a.0	a <sub>.1</sub>	а

Table: 분류표

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

$$precision = \frac{a_{11}}{a_{11} + a_{01}}$$

$$recall = \frac{a_{11}}{a_{11} + a_{10}} \ (= \ 민감도)$$

• F<sub>1</sub> 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.

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

- 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 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$RESPOND, classification(logit, test, 0.5))
##
##
##
    0 2561
   1 206
##
crosstable = function(model, newdata, cutoff){
 table(test$RESPOND,classification(model,newdata,cutoff))
crosstable(logit, test, 1/4)
##
##
##
    0 2547 14
##
    1 203
```

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

## ROC 곡선

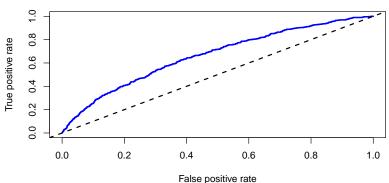
- 절단값에 따라서 특이도와 민감도, 오분류율이 달라진다.
- 절단값을 바꿔가면서 특이도와 민감도를 계산하여 그래프로 나타낸다.
- ROC 곡선
  - x축: 1 특이도
  - y축 : 민감도

### ROC 곡선 및 AUC

```
prob = predict(logit, train, type = 'response')
#### AUC
library(ROCR)
## Loading required package: gplots
##
## Attaching package: 'qplots'
## 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 for Train data AUC: 0.6627



### 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.

#### • 확률 추정 및 절단값 생성

## • 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
##		0.18		0.0988	0.9715	0.1364
##		0.19				
##	[20,]	0.20	0.0875	0.0679	0.9812	0.1046
##		0.21		0.0514	0.9844	0.0828
##	- , -	0.22				
##	[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
##	[26,]	0.26	0.0787			0.0379
##	[27,]	0.27	0.0778	0.0165	0.9960	0.0309

• 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 5967
##
         1 485
```

• 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 F1 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
##
```

#### • 예측자료에서의 비교.

# 목차

- 1. 로지스틱 회귀 분석 로지스틱 회귀 모형 절단값과 오분류율
- 변수선택 및 벌점화 기법 Variable Selection 벌점화 기법 모형 비교
- 3. 앙상블 의사결정나무 배강 및 랜덤포레스트 부스팅
- 4. Case-control sampling

### 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*<sup>2</sup>, ...

#### 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:
                       Median
##
       Min
                  10
                                     30
                                              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 *
             0.0228751 0.0144686 1.581 0.113923
## LOCD
## I.OCB
             -0.0131579 0.0085870 -1.532 0.125494
## INCOME
             -0.0003012 0.0002056 -1.465 0.143074
```

### Penalized regression

- 선형회귀모형의 다중공선성과 과대적합을 해결하기 위해 회귀계수에 벌점함수를 줌
- 대표적으로 Ridge와 Lasso라는 방법이 있다.
- Ridge

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

Lasso

$$\hat{\beta} = \operatorname{argmin}_{\beta} \left\{ (y - X\beta)^{\mathsf{T}} (y - X\beta) + \lambda \sum_{i=1}^{p} |\beta_{i}| \right\}$$

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

#### Logistic + Ridge

```
#---- logistic + ridge
library(glmnet)
## Loading required package: Matrix
## Loading required package:
                             foreach
## Loaded glmnet 2.0-10
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"
##
                                     59
                                                  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
             9.875142e-39 8.453843e-04 -0.0729549813
## LOCH
            -2.054345e-39 -1.873398e-04 -0.1873351916
```

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
## ORGSRCD
          -0.0364477620
```

### Logistic + Lasso

```
#---- logistic + lasso
lasso.fit = glmnet(X_train, as.factor(y_train), alpha = 1,
                  familv="binomial")
lasso.fit\frac{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
## 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
## ORGSRCD
```

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
       -0.040834021
## LOCE
## LOCF
## LOCG
## LOCH
## BUY6
## BUY12
           -0.215888169
## BUY18
            0.595990891
## VALUE24
## ORGSRCD
          -0.016573768
```

- 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.

#### • 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)))
```

#### • 모형별 AUC 값 비교.

```
# Model Fualuation
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
## Test set 0.6417832 0.6444236 0.6408866 0.6374084
```

 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,
                                         v train, cutoff can[i])[[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,
                                                  v_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
               0.34 0.0744 0.0049
## Ridge
                                              0.9996 0.0096
                0.35 0.0744
## LASSO
                                  0.000
                                              1.0000 0.0000
```

 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<sub>1</sub> 스코어를 최대로 하는 절단값을 이용하여 예측자료에서 예측 성능 비교.

```
## 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

- 1. 로지스틱 회귀 분석 로지스틱 회귀 모형 절단값과 오분류율
- 2. 변수선택 및 벌점화 기법 Variable Selection 벌점화 기법 모형 비교
- 3. 앙상블 의사결정나무 배깅 및 랜덤포레스트 부스팅 모형 비교
- 4. Case-control sampling

## 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.

cost complexity(a) = 나무 T의 오분류율 + a|T|

여기서 |T|은 나무 T의 끝마디 개수.

# Buytest with decision tree

```
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) *
```

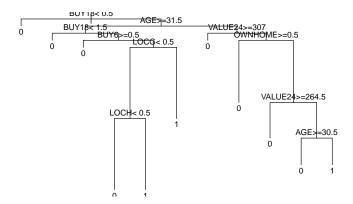
## Buytest with decision tree

- rpart.control
  - minbucket: terminal node에서의 최소 관측치의 수.
  - cp: Complexity parameter (a), 기본값은 0.01로 설정.

```
tree.buydata = rpart(as.factor(RESPOND)~., data = train, control = rpart.control(cp = 0.005))
tree.buydata
## n= 6454
##
## node), split, n, loss, vval, (vprob)
##
        * 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) *
##
             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) *
```

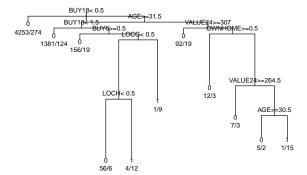
# buydata with decision tree

```
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

```
tree.buydata = rpart(as.factor(RESPOND)~., data = train, control = rpart.control(cp = 0.00
printcp(tree.buvdata)
##
## Classification tree:
## rpart(formula = as.factor(RESPOND) ~ ., data = train, control = rpart.control(cp = 0.00
##
## Variables actually used in tree construction:
## [1] AGE
               BUY18 BUY6 FICO
                                     INCOME LOCB LOCC
                                                          LOCE
                                                                    LOCG
## [10] LOCH
               OWNHOME VALUE24
##
## Root node error: 486/6454 = 0.075302
##
## n = 6454
##
##
           CP nsplit rel error xerror xstd
## 1 0.0059156
                  0 1.00000 1.00000 0.043620
## 2 0.0041152
                  10 0.93827 0.99177 0.043454
## 3 0.0024691
                 12 0.93004 0.99794 0.043578
## 4 0.0013717
                 17 0.91770 1.01440 0.043907
## 5 0.0012346
                  20 0.91358 1.04321 0.044474
## 6 0.0010288
                  31 0.89918 1.05144 0.044634
## 7 0.0010000
                  39
                      0.89095 1.04938 0.044594
```

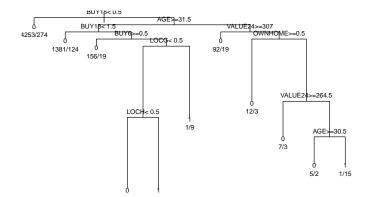
- CP : complexity parameter
- nsplit : |T| 1
- rel error: RSS(|T|)/RSS(1), RSS(k): residual sum of squares for the tree with k terminal nodes.

나무 T의 오분류율 = rel error × Root node error

- xerror × Root node error : cross-validation error rate (10-fold CV)
- xstd: the standard error of xerror

```
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) *
##
##
             ##
        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) *
```

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



### • CV을 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 == i,]$RESPOND)/sum(sample.ind == i)
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

- 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

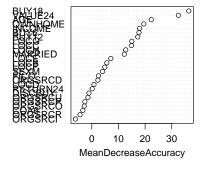
```
library(randomForest)
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
set.seed(1)
p = dim(as.matrix(train))[2] - 1
bag.fit = randomForest(x= X_train, y = as.factor(y_train),
                      mtrv = p, ntree = 500, importance = T)
bag.fit
##
## Call:
   randomForest(x = X_train, y = as.factor(y_train), ntree = 500, mtry = p, importan
##
                  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
```

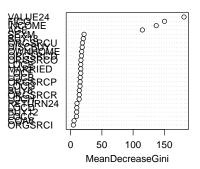
## Bagging with R

• 변수의 중요도 평가

varImpPlot(bag.fit)

bag.fit





## Bagging with R

## • 변수의 중요도 평가

<pre>importance(bag.fit)</pre>								
##		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			
##	DISCBUY	-0.8863171	-0.7357533385	-1.0713226	18.475145			
шш	DETIIDMO4	1 0110600	2 0400025500	0.1600003	10 115050			



#### RandomForest with R

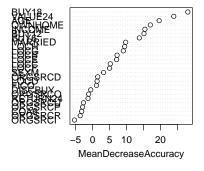
```
set.seed(1)
rf.fit = randomForest(x= X_train, y = as.factor(y_train),
                      mtry = floor(sqrt(p)), ntree = 500, importance = T)
rf.fit.
##
## Call:
  randomForest(x = X_train, y = as.factor(y_train), ntree = 500, mtry = floor(sqrt(
##
                  Type of random forest: classification
##
                        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
```

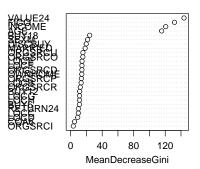
### RandomForest with R

• 변수의 중요도 평가

varImpPlot(rf.fit)

rf.fit





### RandomForest with R

### • 변수의 중요도 평가

imp	importance(rf.fit)									
##		^	4	W D	M D					
	100	0		MeanDecreaseAccuracy						
	AGE		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						
##	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					
##	DISCBUY	-0.3249249	-0.6881814	-0.5528266	19.727107					
шш	DETUDNO4	1 0046307	0.4066704	1 2640627	0.751060					



# **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.

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

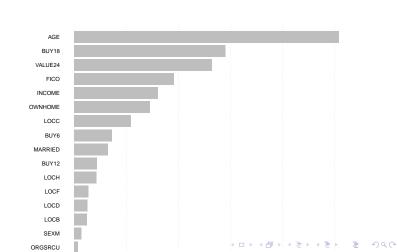
```
install.packages("xgboost")
```

#### Validation for nround

```
# validataion 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
```

```
boost.fit = xgboost(data = X_train, label = y_train, max.depth = 2,
                     eta = 0.1, nround = candidates[which.min(val.err)], objective = "bi-
nary:logistic", verbose = 0)
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
                                           Frequency
                                   Cover
##
           AGE 0.2539029501 0.2204086531 0.210526316
##
         BUY18 0.1449728584 0.1774062417 0.091533181
   3: VALUE24 0.1320686033 0.1112637172 0.151029748
          FICO 0.0956610425 0.1311882671 0.151029748
##
        INCOME 0.0801702893 0.0911149501 0.100686499
##
   6: OWNHOME 0.0726177020 0.0909539245 0.054919908
##
   7:
         LOCC 0.0544839982 0.0517878235 0.043478261
##
          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
         LOCF 0.0137657579 0.0054418662 0.016018307
## 12:
## 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: DRGSRCU 0.0034697666 0.0131104784 0.011441648
```

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

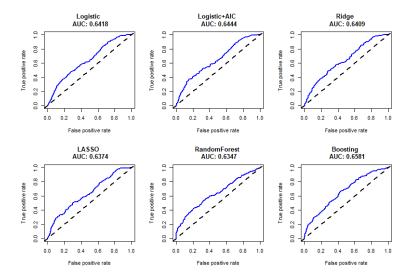


비교

### • 모형별 ROC curve 및 AUC 값 비교.

```
roc.plot = function(pred_prob, y, model_name = NULL){
  AUC = performance(prediction(pred_prob , y) , "auc")
  ROC = performance(prediction(pred_prob ,y) , "tpr","fpr")
  plot(ROC , main = paste(model_name,"\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)
pred_probs_test = sapply(1:4, function(i) 1/(1+exp(-cbind(1, X_test))**beta_hat[,i]))
pred_probs_test = cbind(pred_probs_test, predict(rf.fit, X_test, type = 'prob')[,2],
                        predict(boost.fit, X_test))
model_names = c('Logistic','Logistic+AIC', 'Ridge', 'LASSO',
                'RandomForest', 'Boosting')
par(mfrow = c(2.3))
for (i in 1:6){
 roc.plot(pred_probs_test[,i], y_test, model_names[i])
```

### • 모형별 ROC curve 및 AUC 값 비교.



# 山교

 학습자료에서의 오분류율을 최소화하는 절단값을 이용하여 예측자료에서 예측 성능 비교.

```
cut sel = matrix(0, nrow = 6, ncol = 3)
pred_probs_train = sapply(1:4, function(i) 1/(1+exp(-cbind(1, X_train))**\begin{align*}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_
 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_prob
      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 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
## RandomForest 0.59 0.0744 0.0000
                                      1.0000 0.0000
## Boosting 0.09
                   0.2519
                            0.3932
                                      0.7766
                                            0.1886
```

 학습자료에서의 F<sub>1</sub> 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
## RandomForest 0.28 0.0763
                             0.0922
                                       0.9906 0.1526
## Boosting
             0.12
                  0.1493
                              0.2573
                                       0.8985 0.2042
```

# 목차

- 1. 로지스틱 회귀 분석 로지스틱 회귀 모형 절단값과 오분류율
- 2. 변수선택 및 벌점화 기법 Variable Selection 벌점화 기법 모형 비교
- 3. 앙상블 의사결정나무 배강 및 랜덤포레스트 부스팅
- 4. Case-control sampling

- 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  $S_{\text{max}}$ 와 비정상 자료  $S_{\text{min}}$ 로 분리한다.
  - 2. 정상 자료 중  $|S_{\min}|$ 개 만큼의 자료를 랜덤하게 추출한다.
  - 3. 2에서 추출한 자료와 Smin 을 이용하여 분류모형을 생성한다.
  - 4. 2-3을 충분히 반복한 후, 반복을 통해 만들어진 여러 모형을 합쳐서 최종 앙상블 모형을 생성한다.

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

• buytest 자료에서 "RESPOND=0"인 자료를 "RESPOND=1"인 자료수의 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(v train == 0).
                size = 2*n1, replace = F)
  cc_data = rbind(train[y_train == 1,],
                train[v_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)
```

• 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.