

데이터마이닝 with R

전치혁, 이혜선, 이종석, 이영록

2019-07-08

Contents

개요	7
1 데이터마이닝 개요	9
I 1부 - 예측	11
2 회귀분석	13
2.1 필요 R 패키지 설치	13
2.2 다중회귀모형	13
2.3 반응치에 대한 추정 및 예측	14
2.4 지시변수와 회귀모형	19
3 주성분분석	23
3.1 필요 R 패키지 설치	23
3.2 행렬의 분해	23
3.3 주성분 회귀분석	40
4 부분최소자승법	45
4.1 필요 R 패키지 설치	45
4.2 하나의 종속변수의 경우	45
4.3 다수의 종속변수의 경우	54

II 2부 - 분류분석	65
5 분류분석 개요	67
5.1 필요 R 패키지 설치	67
5.2 분류문제 및 분류기법	67
5.3 기본적인 분류기법	68
6 로지스틱 회귀분석	83
6.1 필요 R 패키지 설치	83
6.2 이분 로지스틱 회귀모형	83
6.3 명목 로지스틱 회귀모형	92
6.4 서열 로지스틱 회귀모형	99
7 판별분석	107
7.1 개요	107
7.2 필요 R 패키지 설치	107
7.3 피셔 방법	107
7.4 의사결정론에 의한 선형분류규칙	114
7.5 오분류비용을 고려한 분류규칙	120
7.6 이차판별분석	122
7.7 세 범주 이상의 분류	127
8 트리기반 기법	131
8.1 CART 개요	131
8.2 필요 R package 설치	131
8.3 CART 트리 생성	131
8.4 가지치기 및 최종 트리 선정	138
8.5 R패키지 내 분류 트리 방법	146
9 서포트 벡터 머신	149
9.1 개요	149
9.2 필요 R package 설치	149
9.3 선형 SVM - 분리 가능 경우	149

CONTENTS	5
9.4 선형 SVM - 분리 불가능 경우	155
9.5 비선형 SVM	160
9.6 R패키지 내 SVM	164
10 분류규칙의 성능 평가	169
10.1 필요 R 패키지 설치	169
10.2 분류오류율	169
10.3 정확도, 민감도 및 특이도	170
10.4 ROC 곡선	173
10.5 이익도표	175
III 3부 - 군집분석	181
11 군집분석 개요	183
11.1 필요 R 패키지 설치	183
11.2 군집분석 기법	183
11.3 객체 간의 유사성 척도	184
11.4 범주형 객체의 유사성 척도	190
12 계층적 군집방법	197
12.1 필요 R 패키지 설치	197
12.2 군집 간 거리척도 및 연결법	197
12.3 연결법의 군집 알고리즘	198
12.4 워드 방법	207
12.5 분리적 방법 - 다이아나	218
12.6 군집수의 결정	224
13 비계층적 군집방법	229
13.1 필요 R package 설치	229
13.2 K-means 알고리즘	229
13.3 K-medoids 군집방법	236
13.4 퍼지 K-means 알고리즘	255
13.5 모형기반 군집방법	260

14 군집해의 평가 및 해석	279
14.1 필요 R package 설치	279
14.2 군집해의 평가	279
14.3 군집해의 해석	293
 IV 4부 - 연관규칙	 295
15 연관규칙	297
15.1 필요 R package 설치	297
15.2 연관규칙의 정의 및 성능척도	297
15.3 연관규칙의 탐사	301
15.4 순차적 패턴의 탐사	305
 16 추천시스템	 317
16.1 필요 R package 설치	317
16.2 내용기반 추천시스템	317
16.3 협업 필터링	324
16.4 시장바구니 데이터를 이용한 협업 필터링	328

개요

본 사이트는 전치혁 교수님의 책 을 기반으로 한 R 예제를 제공할 목적으로 만들어졌으며, 지속적으로 업데이트될 예정입니다. 본 사이트의 R 예제들은 R 3.6.0 version에서 수행되었으며, R 프로그램은 CRAN에서 다운로드받아 설치할 수 있습니다.

본 사이트는 R을 이용한 데이터마이닝 수행에 초점을 두고 있으며, 예제 수행을 위해서는 기본적인 R 프로그래밍 지식이 필요합니다. R 프로그래밍에 대한 지식은 아래와 같은 자료들로부터 얻을 수 있습니다.

- R for Data Science (by Hadley Wickham & Garrett Grolemund): <https://r4ds.had.co.nz>
- Advanced R (by Hadley Wickham): <https://adv-r.hadley.nz>

Chapter 1

데이터마이닝 개요

데이터마이닝은 다양한 목적으로 사용될 수 있다. 그러나 기본적인 목적에 따라 분류할 때, 예측(prediction), 분류(classification), 군집화(clustering), 연관규칙(association rule) 추출로 구분할 수 있겠다.

추후 내용 추가

Part I

1부 - 예측

Chapter 2

회귀분석

2.1 필요 R 패키지 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
stats	3.6.0
broom	0.5.2

2.2 다중회귀모형

아래와 같이 n 개의 객체와 k 개의 독립변수 (\mathbf{x})로 이루어지고 하나의 종속변수 (y)로 이루어진 선형 회귀모형을 정의하자.

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + \epsilon_i \quad (2.1)$$

이 때, 오차항 ϵ_i 은 서로 독립이고 동일한 정규분포 $N(0, \sigma^2)$ 을 따른다.

위 회귀모형은 아래와 같이 행렬의 연산으로 표현할 수 있다.

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \quad (2.2)$$

이 때,

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix}$$

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

로 정의되며,

$$E[\boldsymbol{\epsilon}] = \mathbf{0}, \text{Var}[\boldsymbol{\epsilon}] = \sigma^2 \mathbf{I}$$

이다.

2.3 반응치에 대한 추정 및 예측

2.3.1 평균반응치의 추정

2.3.1.1 기본 R 스트립트

다음과 같은 10명의 나이(age), 키(height), 몸무게(weight)에 대한 데이터가 있다.

Table 2.1: 나이, 키, 몸무게 데이터

나이 (age)	키 (height)	몸무게 (weight)
21	170	60
47	167	65
36	173	67
15	165	54
54	168	73
25	177	71
32	169	68
18	172	62
43	171	66
28	175	68

```

train_df <- tribble(
  ~age, ~height, ~weight,
  21, 170, 60,
  47, 167, 65,
  36, 173, 67,
  15, 165, 54,
  54, 168, 73,
  25, 177, 71,
  32, 169, 68,
  18, 172, 62,
  43, 171, 66,
  28, 175, 68
)

knitr::kable(
  train_df, booktabs = TRUE,
  align = c('r', 'r', 'r'),
  col.names = c('나이 (age)', '키 (height)', '몸무게 (weight)'),
  caption = '나이, 키, 몸무게 데이터'
)

```

회귀모형을 아래와 같이 학습한다.

```
lm_fit <- lm(weight ~ age + height, data = train_df)
```

추정된 회귀계수는 아래와 같다.

```
coef(lm_fit)

## (Intercept)      age      height
## -108.1671993   0.3291212   0.9552913
```

추정계수벡터의 분산-공분산 행렬은 아래와 같다.

```
vcov(lm_fit)

##                (Intercept)      age      height
## (Intercept) 1774.3280624 -0.671107283 -10.264885141
## age         -0.6711073  0.004794717  0.003035476
## height       -10.2648851  0.003035476  0.059566804
```

나이가 40, 키가 170인 사람들의 평균 몸무게에 대한 95% 신뢰구간은 아래와 같이 구할 수 있다.

```
predict(lm_fit, newdata = tibble(age = 40, height = 170),
        interval = "confidence", level = 0.95)

##      fit      lwr      upr
## 1 67.39718 65.01701 69.77735
```

2.3.1.2 평균 반응치의 분산 추정

새로운 독립변수에 대한 벡터를 아래와 같이 \mathbf{x}_0 라 하면, 평균반응치의 추정량은 아래와 같이 표현된다.

$$\hat{y}_0 = \mathbf{x}_0^\top \hat{\beta} \quad (2.3)$$

식 (2.3)의 분산은 아래와 같다.

$$Var(\hat{y}_0) = \mathbf{x}_0^\top Var(\hat{\beta}) \mathbf{x}_0 \quad (2.4)$$

$$= \sigma^2 \mathbf{x}_0^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_0 \quad (2.5)$$

위 식 (2.5)에서 σ^2 대신 그 추정값인 MSE (mean squared error)를 대입하여 평균반응치의 분산을 추정한다.

$$\hat{Var}(\hat{y}_0) = MSE \times \left(\mathbf{x}_0^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_0 \right) \quad (2.6)$$

우선 Table 2.1에 대해 회귀모형 추정치 $\hat{\beta}$ 와 MSE 값을 구해보자.

```

n <- nrow(train_df)
X <- cbind(
  intercept = rep(1, n),
  train_df[, c("age", "height")] %>% as.matrix()
)
y <- train_df[, c("weight")] %>% as.matrix()
k <- ncol(X) - 1

# regression coefficient
beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y

# response estimate
y_hat <- X %*% beta_hat

# MSE
MSE <- sum((y - y_hat) ^ 2) / (n - k - 1)

```

이후, 나이가 40, 키가 170인 객체 \mathbf{x}_0 에 대한 평균 반응치 \hat{y}_0 의 95% 신뢰구간을 구해보자.

```

# variance of y_hat on new_x
new_x <- matrix(c(
  intercept = 1,
  age = 40,
  height = 170
), ncol = 1)

new_y_hat <- t(new_x) %*% beta_hat

var_new_y_hat <- MSE * t(new_x) %*% solve(t(X) %*% X) %*% new_x
se_new_y_hat <- sqrt(var_new_y_hat)

lci <- new_y_hat + qt(0.025, n - k - 1) * se_new_y_hat
uci <- new_y_hat + qt(0.975, n - k - 1) * se_new_y_hat

str_glue("({format(lci, nsmall = 3L)}, {format(uci, nsmall = 3L)})")

## (65.01701, 69.77735)

```

2.3.2 미래반응치의 예측

2.3.2.1 기본 R 스트립트

2.3.1 절에서 추정한 회귀모형을 통해, 새로운 독립변수값(나이 = 40, 키 = 170)을 지닌 특정 객체에 대한 몸무게의 예측구간을 구한다. 이는 한 객체의 몸무게의 예측구간으로, 2.3.1 절에서 구한 평균 몸무게의 신뢰구간보다 넓다.

```
predict(lm_fit, newdata = tibble(age = 40, height = 170),
        interval = "prediction", level = 0.95)

##           fit      lwr      upr
## 1 67.39718 60.68745 74.1069
```

2.3.2.2 미래 반응치의 예측구간 추정

독립변수값들에 대응하는 미래반응치인 y_0 의 예측치는 평균반응치의 추정치와 동일하며, 반응치의 예측구간을 구하기 위해서는 예측오차의 분산을 알아야 하는데, 이는 아래와 같이 식 (2.5)보다 σ^2 이 더 크게 된다.

$$Var(y_0 - \hat{y}_0) = Var(y_0) + Var(\hat{y}_0) \quad (2.7)$$

$$= \sigma^2 + \sigma^2 \mathbf{x}_0^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_0 \quad (2.8)$$

$$= \sigma^2 \left(1 + \mathbf{x}_0^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_0 \right) \quad (2.9)$$

위 식 (2.9)에서 σ^2 대신 MSE 를 대입함으로써 예측오차 분산을 추정할 수 있다.

$$\hat{Var}(y_0 - \hat{y}_0) = MSE \times \left(1 + \mathbf{x}_0^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_0 \right) \quad (2.10)$$

앞 절에서 추정한 회귀모형을 이용하여, 나이가 40, 키가 170인 객체 \mathbf{x}_0 에 대한 미래 반응치 y_0 의 95% 예측구간을 구해보자.

```
# variance of y_hat on new_x
new_x <- matrix(c(
  intercept = 1,
  age = 40,
  height = 170
), ncol = 1)

new_y_hat <- t(new_x) %*% beta_hat

var_new_y_pred <- MSE * (1 + t(new_x) %*% solve(t(X) %*% X) %*% new_x)
se_new_y_pred <- sqrt(var_new_y_pred)

lci <- new_y_hat + qt(0.025, n - k - 1) * se_new_y_pred
uci <- new_y_hat + qt(0.975, n - k - 1) * se_new_y_pred

str_glue("({format(lci, nsmall = 3L)}, {format(uci, nsmall = 3L)})")

## (60.68745, 74.1069)
```

2.4 지시변수와 회귀모형

2.4.1 기본 R 스트립트

어떤 열연코일의 인장강도(TS)에 권취온도(CT)가 어떤 영향을 미치는가를 조사하기 위해 TS를 종속변수, CT를 독립변수로 하여 회귀분석을 실시하기로 하였다. 수집된 데이터에는 두 개의 두께 그룹(2mm, 6mm)이 포함되어 있다.

```
train_df <- tribble(
  ~ct, ~thickness, ~ts,
  540, 2, 52.5,
  660, 2, 50.2,
  610, 2, 51.3,
  710, 2, 49.1,
  570, 6, 50.8,
  700, 6, 48.7,
  560, 6, 51.2,
  600, 6, 50.8,
  680, 6, 49.3,
  530, 6, 51.5
) %>%
  mutate(thickness = factor(thickness, levels = c(6, 2)))

str(train_df)

## Classes 'tbl_df', 'tbl' and 'data.frame':    10 obs. of  3 variables:
## $ ct      : num  540 660 610 710 570 700 560 600 680 530
## $ thickness: Factor w/ 2 levels "6","2": 2 2 2 2 1 1 1 1 1 1
## $ ts      : num  52.5 50.2 51.3 49.1 50.8 48.7 51.2 50.8 49.3 51.5
```

두께를 factor로 지정하고 회귀모형을 추정하자.

```
lm_fit <- lm(ts ~ ct + thickness, data = train_df)
```

회귀 계수는 아래와 같이 얻어진다.

```
coef(lm_fit)
```

```
## (Intercept)      ct  thickness2
## 61.10796610 -0.01767797  0.80415254
```

두께 그룹에 따라 CT에 대한 TS의 기울기가 다르다고 예상되면 CT와 두께 간에 교호작용(interaction)이 존재한다고 말하며, 이 때 회귀모형은 다음과 같이 추정한다.

```
lm_interaction_fit <- lm(
  ts ~ ct + thickness + ct:thickness,
  data = train_df
)
```

교호작용이 추가된 회귀 결과는 아래와 같다.

```
broom::tidy(lm_interaction_fit)
```

term	estimate	std.error	statistic	p.value
<chr>	<dbl>	<dbl>	<dbl>	<dbl>
## 1 (Intercept)	60.1	0.750	80.2	2.53e-10
## 2 ct	-0.0161	0.00123	-13.1	1.23e- 5
## 3 thickness2	3.28	1.21	2.71	3.52e- 2
## 4 ct:thickness2	-0.00399	0.00194	-2.05	8.57e- 2

두께에 따른 CT와 TS의 관계를 그래프로 살펴보자.

```
new_df <- crossing(
  ct = seq(500, 750, by = 10),
  thickness = factor(c(6, 2), levels = c(6, 2))
)

new_df %>%
  mutate(ts_hat = predict(lm_interaction_fit, .)) %>%
  ggplot(aes(x = ct, y = ts_hat)) +
  geom_line(aes(color = thickness)) +
  geom_point(aes(x = ct, y = ts, color = thickness), data = train_df) +
  labs(color = "thickness", x = "CT", y = "TS")
```

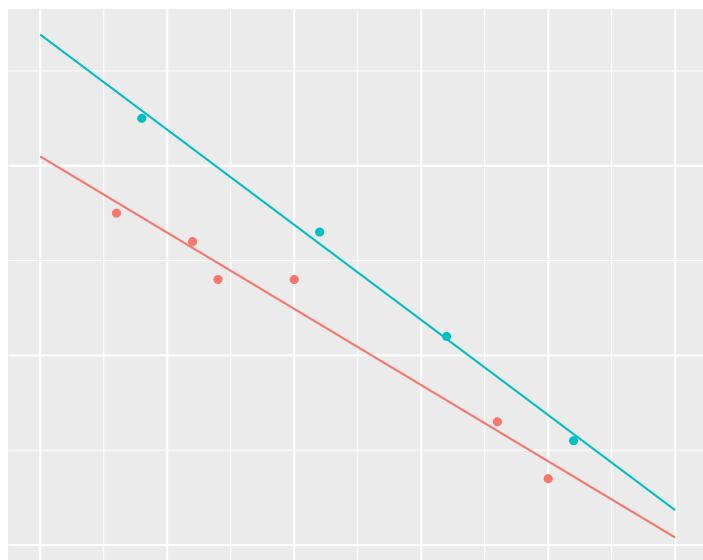


Figure 2.1: 두께에 따른 CT와 TS의 관계

Chapter 3

주성분분석

3.1 필요 R 패키지 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
stats	3.6.0
broom	0.5.2
Matrix	1.2-17
nipals	0.5

3.2 행렬의 분해

3.2.1 기본 R 스트립트

아래 Table 3.1는 국내 18개 증권회사의 주요 재무제표를 나열한 것이다.

```
train_df <- tribble(
  ~company, ~roa, ~roe, ~bis, ~de_ratio, ~turnover,
  "SK증권", 2.43, 11.10, 18.46, 441.67, 0.90,
  "교보증권", 3.09, 9.95, 29.46, 239.43, 0.90,
  "대신증권", 2.22, 6.86, 28.62, 249.36, 0.69,
  "대우증권", 5.76, 23.19, 23.47, 326.09, 1.43,
  "동부증권", 1.60, 5.64, 25.64, 289.98, 1.42,
  "메리츠증권", 3.53, 10.64, 32.25, 210.10, 1.17,
  "미래에셋증권", 4.26, 15.56, 24.40, 309.78, 0.81,
  "부국증권", 3.86, 5.50, 70.74, 41.36, 0.81,
```

```

    "브릿지증권", 4.09, 6.44, 64.38, 55.32, 0.32,
    "삼성증권", 2.73, 10.68, 24.41, 309.59, 0.64,
    "서울증권", 2.03, 4.50, 42.53, 135.12, 0.59,
    "신영증권", 1.96, 8.92, 18.48, 441.19, 1.07,
    "신흥증권", 3.25, 7.96, 40.42, 147.41, 1.19,
    "우리투자증권", 2.01, 10.28, 17.46, 472.78, 1.25,
    "유화증권", 2.28, 3.65, 63.71, 56.96, 0.12,
    "한양증권", 4.51, 7.50, 63.52, 57.44, 0.80,
    "한화증권", 3.29, 12.37, 24.47, 308.63, 0.57,
    "현대증권", 1.73, 7.57, 19.59, 410.45, 1.19
)

knitr::kable(
  train_df, booktabs = TRUE,
  align = rep("r", ncol(train_df)),
  col.names = c(
    "회사",
    "총자본 순이익률 ($x_1$)",
    "자기자본 순이익률 ($x_2$)",
    "자기자본비율 ($x_3$)",
    "부채비율 ($x_4$)",
    "자기자본 회전율 ($x_5$)"
  ),
  caption = "국내 증권회사의 주요 재무제표"
)

```

이에 대하여 R 기본 stats 패키지 내의 prcomp 함수를 이용하여 주성분 분석을 수행할 수 있다.

```
pca_fit <- prcomp(~ roa + roe + bis + de_ratio + turnover,
```

```
           data = train_df, scale = TRUE)
```

```
pca_fit
```

```

## Standard deviations (1, ..., p=5):
## [1] 1.6617648 1.2671437 0.7419994 0.2531070 0.1351235
##
## Rotation (n x k) = (5 x 5):
##          PC1        PC2        PC3        PC4        PC5
## roa      0.07608427 -0.77966993  0.0008915975 -0.140755404  0.60540325
## roe     -0.39463007 -0.56541218 -0.2953216494  0.117644166 -0.65078503
## bis      0.56970191 -0.16228156  0.2412221065 -0.637721889 -0.42921686
## de_ratio -0.55982770  0.19654293 -0.2565972887 -0.748094314  0.14992183
## turnover -0.44778451 -0.08636803  0.8881182665 -0.003668418 -0.05711464

```

Table 3.1: 국내 증권회사의 주요 재무제표

회사	총자본 순이익율 (\$x_1\$)	자기자본 순이익율 (\$x_2\$)	자기자본비율 (\$x_3\$)	부채비율 (\$x_4\$)	자기자본비율 (\$x_5\$)
SK증권	2.43	11.10	18.46	441.67	
교보증권	3.09	9.95	29.46	239.43	
대신증권	2.22	6.86	28.62	249.36	
대우증권	5.76	23.19	23.47	326.09	
동부증권	1.60	5.64	25.64	289.98	
메리츠증권	3.53	10.64	32.25	210.10	
미래에셋증권	4.26	15.56	24.40	309.78	
부국증권	3.86	5.50	70.74	41.36	
브릿지증권	4.09	6.44	64.38	55.32	
삼성증권	2.73	10.68	24.41	309.59	
서울증권	2.03	4.50	42.53	135.12	
신영증권	1.96	8.92	18.48	441.19	
신흥증권	3.25	7.96	40.42	147.41	
우리투자증권	2.01	10.28	17.46	472.78	
유화증권	2.28	3.65	63.71	56.96	
한양증권	4.51	7.50	63.52	57.44	
한화증권	3.29	12.37	24.47	308.63	
현대증권	1.73	7.57	19.59	410.45	

```
summary(pca_fit)
```

```
## Importance of components:
##                 PC1    PC2    PC3    PC4    PC5
## Standard deviation 1.6618 1.2671 0.7420 0.25311 0.13512
## Proportion of Variance 0.5523 0.3211 0.1101 0.01281 0.00365
## Cumulative Proportion 0.5523 0.8734 0.9835 0.99635 1.00000
```

각 주성분에 대한 고유값을 스크리도표로 나타내면 아래 Figure 3.1

```
screeplot(pca_fit, main = NULL)
```

3.2.2 변수의 변동과 제곱합

총 k 개의 독립변수가 있고 각 독립변수에 대하여 n 개의 관측치가 있다고 하자. 이 때, x_{ij} 를 j 번째 독립변수에 대한 i 번째 관측치라 하자. 즉, 관측데이터는 아래와 같은 행렬로 표현할 수 있다.

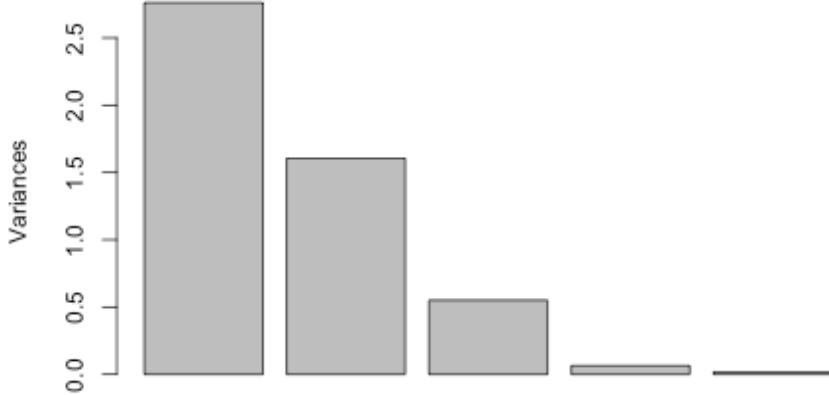


Figure 3.1: 고유치 스크리 도표

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1k} \\ x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nk} \end{bmatrix}$$

주성분분석에서는 통상 원데이터를 그대로 사용하지 않고 적절한 변환을 취하는데, 주로 평균조정 (mean-centered) 데이터를 이용한다. 이는 아래와 같이 독립변수에 대하여 표본평균을 뺌으로써 조정된 변수의 평균이 0이 되도록 하는 것이다.

$$x_{ij} \leftarrow x_{ij} - \frac{1}{n} \sum_{l=1}^n x_{lj} \quad (3.1)$$

이후에 별도의 언급이 없는 한, 행렬 \mathbf{X} 및 변수값 x_{ij} 는 식 (3.1)을 이용하여 평균조정된 것으로 가정한다.

이 밖에도 다른 변환이 사용되는 경우가 있는데, 특히 단위 등이 서로 상이할 경우에는 평균조정 이후 추가로 각 변수의 분산이 1이 되도록 분산조정을 한다.

$$z_{ij} \leftarrow \frac{x_{ij}}{\sqrt{\frac{1}{n-1} \sum_{l=1}^n x_{lj}^2}}$$

이 때, 식 (3.2.2)에서 분모 부분은 변수의 표본 표준편차로 s_j 로 표현된다.

$$s_j = \sqrt{\frac{1}{n-1} \sum_{l=1}^n x_{lj}^2}$$

이후 분산조정을 이용하는 경우 행렬 \mathbf{Z} 및 변수값 z_{ij} 로 표현한다.

$$\mathbf{Z} = \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1k} \\ z_{21} & z_{22} & \cdots & z_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n1} & z_{n2} & \cdots & z_{nk} \end{bmatrix}$$

변수벡터 $\mathbf{x}_j = [x_{1j} \ x_{2j} \ \cdots \ x_{nj}]^\top$ 에 대한 제곱합의 정의는 아래와 같다.

$$SS(\mathbf{x}_j) = \mathbf{x}_j^\top \mathbf{x}_j = \sum_{i=1}^n x_{ij}^2 \quad (3.2)$$

따라서, 평균조정된 변수에 대해 제곱합과 표본분산은 다음과 같은 관계가 있다.

$$SS(\mathbf{x}_j) = (n-1)s_j^2$$

변수행렬 \mathbf{X} 에 대한 제곱합은 각 변수들의 제곱합의 총합(총변동)으로 정의된다.

$$SS(\mathbf{X}) = \sum_{j=1}^k SS(\mathbf{x}_j) = \sum_{j=1}^k \sum_{i=1}^n x_{ij}^2 \quad (3.3)$$

Table 3.1 데이터에 대하여 각 변수의 제곱합을 계산해보자.

```
train_df %>%
  mutate_if(is.numeric, function(x) x - mean(x)) %>% # mean-centering
  summarize_if(is.numeric, function(x) sum(x ^ 2)) # sum of squares by variable

## # A tibble: 1 x 5
##       roa      roe      bis de_ratio turnover
##     <dbl>    <dbl>    <dbl>    <dbl>
## 1  21.9   355.  5591.  347817.    2.23
```

전체 데이터 행렬에 대한 제곱합은 아래와 같다.

```
train_df %>%
  mutate_if(is.numeric, function(x) x - mean(x)) %>% # mean-centering
  summarize_if(is.numeric, function(x) sum(x ^ 2)) %>% # sum of squares by variable
  {sum(.)}
```

```
## [1] 353786.6
```

위 결과에서 부채비율(de_ratio)의 변동이 총변동의 대부분을 차지하고, 자기자본 회전율(turnover)이 총변동에 미치는 영향은 미미한데, 이는 각 변수들이 측정하는 값의 분포(범위)가 크게 다르기 때문이다. 이러한 경우, 일반적으로 분산조정을 추가로 적용한 뒤 주성분분석을 수행한다.

```
standardized_df <- train_df %>%
  mutate_if(is.numeric, function(x) x - mean(x)) %>% # mean-centering
  mutate_if(is.numeric, function(x) x / sd(x)) # scaling
```

```
standardized_df
```

```
## # A tibble: 18 x 6
##   company      roa     roe     bis de_ratio turnover
##   <chr>       <dbl>   <dbl>   <dbl>    <dbl>    <dbl>
## 1 SK증권     -0.533  0.383  -0.918   1.34    0.0507
## 2 교보증권    0.0484  0.131  -0.312  -0.0749   0.0507
## 3 대신증권   -0.718  -0.545  -0.358  -0.00551  -0.530
## 4 대우증권    2.40    3.03   -0.642   0.531    1.52
## 5 동부증권   -1.26   -0.812  -0.522   0.278    1.49
## 6 메리츠증권  0.436   0.282  -0.158  -0.280    0.797
## 7 미래에셋증권 1.08    1.36   -0.591   0.417   -0.198
## 8 부국증권    0.726   -0.843   1.96   -1.46   -0.198
## 9 브릿지증권  0.929   -0.637   1.61   -1.36   -1.55
## 10 삼성증권   -0.269   0.291  -0.590   0.416   -0.668
## 11 서울증권   -0.885  -1.06    0.409  -0.804   -0.806
## 12 신영증권   -0.947  -0.0942  -0.917   1.34    0.521
## 13 신흥증권    0.189  -0.304   0.293  -0.718    0.852
## 14 우리투자증권 -0.903  0.203  -0.973   1.56    1.02
## 15 유화증권   -0.665  -1.25    1.58   -1.35   -2.11
## 16 한양증권    1.30    -0.405   1.57   -1.35   -0.226
## 17 한화증권    0.225   0.661  -0.587   0.409   -0.861
## 18 현대증권   -1.15   -0.390  -0.856   1.12    0.852
```

분산조정 이후의 각 변수의 제곱합은 모두 $n - 1$ 이 되는데, 이는 각 변수의 표본분산이 모두 1로 조정되기 때문이다.

```
standardized_df %>%
  summarize_if(is.numeric, function(x) sum(x ^ 2)) # sum of squares by variable

## # A tibble: 1 x 5
##   roa    roe    bis de_ratio turnover
##   <dbl> <dbl> <dbl>    <dbl>     <dbl>
## 1    17   17.0   17       17      17.
```

따라서, 분산조정 이후 총변동은 아래와 같다.

```
total_ss <- standardized_df %>%
  summarize_if(is.numeric, function(x) sum(x ^ 2)) %>% # sum of squares by variable
  {sum(.)}

total_ss

## [1] 85
```

3.2.3 주성분의 이해 및 행렬의 분해

주성분분석은 원래의 변수들의 선형조합으로 서로 직교하는 새로운 변수들을 생성하는 것이라 할 수 있다. 이 때, 원래 변수의 수 k 보다 작은 A 개의 새로운 변수들이 원 데이터 행렬 \mathbf{X} 총변동의 대부분을 설명한다고 하면, 해당 새로운 변수들만을 사용하여 여러 가지 분석을 대신할 수 있다는 것이 주성분분석의 개념이라 하겠다.

새로운 변수 $\mathbf{t}_1, \dots, \mathbf{t}_A$ 들은 다음과 같은 형태로 표현된다.

$$\mathbf{t}_a = \sum_{j=1}^k p_{aj} \mathbf{x}_j, \quad a = 1, \dots, A \quad (3.4)$$

결과적으로 주성분분석은 위와 같이 표현되는 새로운 변수를 만들 때 필요한 계수 p_{aj} 를 구하는 것이라 할 수 있겠다. \mathbf{t}_1 이 \mathbf{X} 의 변동을 가장 많이 설명하도록, \mathbf{t}_2 는 \mathbf{t}_1 이 설명하지 못한 변동을 가장 많이 설명하도록 하는 방식으로 A 개의 새로운 변수를 순차적으로 찾아내는 것이 기본적인 원리이다.

Table 3.1 데이터에 대하여 분산조정을 적용한 후 아래 식을 이용하여 새로운 변수를 도출해보자.

$$t_1 = 0.07608427 \times roa - 0.39463007 \times roe + 0.56970191 \times bis - 0.55982770 \times de_ratio - 0.44778451 \times turnover$$

```

new_df <- standardized_df %>%
  mutate(t_1 = 0.07608427 * roa - 0.39463007 * roe
    + 0.56970191 * bis - 0.55982770 * de_ratio
    - 0.44778451 * turnover) %>%
  select(company, t_1) # new variable

print(new_df)

## # A tibble: 18 x 2
##   company      t_1
##   <chr>       <dbl>
## 1 SK증권     -1.49
## 2 교보증권    -0.206
## 3 대신증권     0.197
## 4 대우증권     -2.35
## 5 동부증권     -0.895
## 6 메리츠증권    -0.368
## 7 미래에셋증권 -0.935
## 8 부국증권      2.41
## 9 브릿지증권     2.70
## 10 삼성증권     -0.405
## 11 서울증권      1.40
## 12 신영증권     -1.54
## 13 신흥증권      0.322
## 14 우리투자증권 -2.03
## 15 유화증권      3.04
## 16 한양증권      2.01
## 17 한화증권     -0.421
## 18 현대증권     -1.43

```

이 때, 새로운 변수 t_1 는 분산조정된 행렬 Z 의 총변동의 약 55%를 설명한다.

```

t1_ss <- new_df %>%
  summarize_if(is.numeric, function(x) sum(x ^ 2))

t1_ss / total_ss

##           t_1
## 1 0.5522924

```

위 새로운 변수 t_1 는 실제로 행렬 Z 로부터 얻어지는 첫 번째 주성분이며, 행렬 Z 의 변동에 가장 많이 기여하는 하나의 선형조합이다.

행렬 Z (혹은 X)로부터 주성분을 얻는 방법은 여러 가지가 있으며, 아래에서 하나씩 설명하기로 한다.

3.2.4 특이치분해 (Singular Value Decomposition)

분산조정된 \mathbf{Z} 에 대해 주성분분석을 수행한다고 가정하자. 분산조정을 하지 않고 주성분분석을 수행하는 경우 아래 행렬 \mathbf{Z} 대신 \mathbf{X} 를 사용하면 된다.

임의의 $(n \times k)$ 행렬 \mathbf{Z} 는 다음과 같이 분해된다.

$$\mathbf{Z} = \mathbf{UDV}^\top \quad (3.5)$$

이 때, $r = \min\{n, k\}$ 라 할 때,

- \mathbf{U} : $(n \times r)$ 직교 (orthogonal) 행렬
- \mathbf{D} : $(r \times r)$ 대각 (diagonal) 행렬. rank 수만큼의 비음 대각원소들을 가지며, 각 비음 대각원소를 헝렬 \mathbf{Z} 의 특이치 (singular value)라 하고, 특이치가 내림차순으로 정렬되는 형태로 행렬이 구성된다.
- \mathbf{V} : $(k \times r)$ 직교 (orthogonal) 행렬

아래와 같이, R 함수 `svd`를 이용하여 분해한 행렬들을 곱한 결과가 원래 행렬 \mathbf{Z} 와 동일함을 확인할 수 있다.

```
Z <- as.matrix(standardized_df[, -1])
svd_Z <- svd(Z)
Z_rec <- svd_Z$u %*% diag(svd_Z$d) %*% t(svd_Z$v)
all(near(Z, Z_rec))

## [1] TRUE
```

이 때, 행렬 \mathbf{V} 의 각 열벡터가 각 주성분을 도출하는 선형식의 계수가 된다. 즉, 행렬 \mathbf{V} 의 첫 번째 열이 위에서 살펴본 새로운 변수 t_1 를 도출하는 선형식의 계수이다.

```
svd_Z$v[, 1]
```

```
## [1] 0.07608427 -0.39463007 0.56970191 -0.55982770 -0.44778451
```

특이치는 아래와 같이 추출된다.

```
svd_Z$d
```

```
## [1] 6.8516318 5.2245674 3.0593417 1.0435870 0.5571285
```

이 특이치들은 아래 분광분해에서 살펴볼 고유치의 제곱근이다.

3.2.5 분광분해 (Spectral Decomposition)

임의의 정방행렬 \mathbf{A} 에 대하여

$$\mathbf{Av} = \lambda v$$

가 성립하는 벡터 $v \neq \mathbf{0}$ 과 상수 λ 가 존재할 때, 상수 λ 를 행렬 \mathbf{A} 의 고유치(eigenvalue)라 하며, v 를 이에 대응하는 고유벡터(eigenvector)라 한다. 통상 $v^\top v = 1$ 을 가정한다.

분광분해는 정방행렬을 고유치와 고유벡터의 곱으로 분해하는 방법이다. ($r \times r$) 정방행렬 \mathbf{A} 에 대해 r 개의 고유치 $\lambda_1, \dots, \lambda_r$ 와 고유벡터 v_1, \dots, v_r 이 존재한다고 할 때, 행렬 \mathbf{A} 는 다음과 같이 정리된다.

$$\mathbf{A} [v_1 \dots v_r] = [v_1 \dots v_r] \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & & 0 \\ \vdots & & \ddots & 0 \\ 0 & 0 & \cdots & \lambda_r \end{bmatrix} \mathbf{A} = \mathbf{A} [v_1 \dots v_r] [v_1 \dots v_r]^{-1} = [v_1 \dots v_r] \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix}$$

특히 행렬 \mathbf{A} 가 대칭(symmetric) 행렬인 경우, 고유벡터들은 서로 직교하므로 ($\mathbf{V}\mathbf{V}^\top = \mathbf{I}$), 위 식을 아래와 같이 표현할 수 있다.

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^\top$$

주성분 분석을 위해 정방행렬 $\mathbf{Z}^\top \mathbf{Z}$ 를 분해를 살펴보자. 식 (3.5)로부터,

$$\mathbf{Z}^\top \mathbf{Z} = \mathbf{V}\mathbf{D}^\top \mathbf{U}^\top \mathbf{U}\mathbf{D}\mathbf{V}^\top = \mathbf{V}\mathbf{D}^2\mathbf{V}^\top = \mathbf{V}\Lambda\mathbf{V}^\top$$

즉, 분광분해를 통해 도출된 고유벡터들의 행렬 \mathbf{V} 의 각 열벡터가 각 주성분을 도출하는 선형식의 계수를 나타내며, 대각행렬 Λ 의 각 대각원소값인 고유치는 특이치의 제곱임을 알 수 있다.

R 함수 `eigen`을 이용하여 분광분해를 아래와 같이 수행하여 보자.

```
eig_Z <- eigen(t(Z) %*% Z, symmetric = TRUE)
eig_Z
```

```
## eigen() decomposition
## $values
## [1] 46.9448582 27.2961041  9.3595718  1.0890737  0.3103922
## 
## $vectors
##           [,1]          [,2]          [,3]          [,4]          [,5]
## [1,] -0.07608427 -0.77966993  0.0008915975 -0.140755404  0.60540325
```

```
## [2,]  0.39463007 -0.56541218 -0.2953216494  0.117644166 -0.65078503
## [3,] -0.56970191 -0.16228156  0.2412221065 -0.637721889 -0.42921686
## [4,]  0.55982770  0.19654293 -0.2565972887 -0.748094314  0.14992183
## [5,]  0.44778451 -0.08636803  0.8881182665 -0.003668418 -0.05711464
```

결과에서 `values`는 행렬 $\mathbf{Z}^\top \mathbf{Z}$ 의 고유치 (eigenvalue)들이다. 이들이 앞 장의 특이치 분해에서 얻은 특이치들의 제곱임을 확인하여 보자.

```
all(near(eig_Z$values, svd_Z$d ^ 2))
```

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

또한 분광분해 결과 `vectors`의 각 열은 행렬 $\mathbf{Z}^\top \mathbf{Z}$ 의 고유벡터 (eigenvector)들이다. 이들이 앞 장의 특이치 분해에서 얻은 계수 행렬과 동일함을 확인하여 보자.

```
all(near(eig_Z$vectors, svd_Z$v))
```

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

이 경우 두 행렬이 동일하지 않게 나타날 수 있는데, 그 이유는 경우에 따라 어떤 주성분을 생성하는 선형계수 부호가 정반대인 형태로 얻어질 수 있기 때문이다. 주성분의 설명력은 선형계수의 부호에 영향을 받지 않는다.

두 행렬의 계수 부호가 서로 동일하게 조정한 뒤 행렬을 비교해보자.

```
sign_adjust <- 1 - 2 * ((eig_Z$vectors * svd_Z$v) < 0)
all(near(eig_Z$vectors * sign_adjust, svd_Z$v))
```

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

위 각 고유값들을 고유값들의 총합으로 나누면, 각 고유벡터에 해당하는 주성분이 설명하는 총변동의 비율을 얻을 수 있다.

```
eig_Z$values / sum(eig_Z$values)
```

```
## [1] 0.552292449 0.321130636 0.110112610 0.012812632 0.003651673
```

평균 및 분산 조정된 \mathbf{Z} 의 분산-공분산 행렬은 아래와 같다.

$$\frac{1}{n-1} \mathbf{Z}^\top \mathbf{Z}$$

```
all(near(cov(Z), t(Z) %*% Z / (nrow(Z) - 1)))
```

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

여기에서 위에서 구한 분광분해를 대입하면,

$$\frac{1}{n-1} \mathbf{Z}^\top \mathbf{Z} = \frac{1}{n-1} \mathbf{V} \Lambda \mathbf{V}^\top = \mathbf{V} \left(\frac{1}{n-1} \Lambda \right) \mathbf{V}^\top$$

따라서, \mathbf{Z} 의 분산-공분산 행렬에 대한 분광분해 결과, 고유벡터 행렬 \mathbf{V} 는 앞에서 구한 $\mathbf{Z}^\top \mathbf{Z}$ 의 고유벡터 행렬들과 동일하며, 고유값은 앞에서 구한 $\mathbf{Z}^\top \mathbf{Z}$ 의 고유값을 $(n-1)$ 으로 나눈 값이다.

```
eig_cov_Z <- eigen(cov(Z))
eig_cov_Z
```

```
## eigen() decomposition
## $values
## [1] 2.76146225 1.60565318 0.55056305 0.06406316 0.01825836
##
## $vectors
## [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] -0.07608427 0.77966993 0.0008915975 -0.140755404 0.60540325
## [2,] 0.39463007 0.56541218 -0.2953216494 0.117644166 -0.65078503
## [3,] -0.56970191 0.16228156 0.2412221065 -0.637721889 -0.42921686
## [4,] 0.55982770 -0.19654293 -0.2565972887 -0.748094314 0.14992183
## [5,] 0.44778451 0.08636803 0.8881182665 -0.003668418 -0.05711464
```

```
all(near(eig_cov_Z$values, eig_Z$values / (nrow(Z) - 1)))
```

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

또한 이 결과는 평균 및 분산조정 이전 원 데이터의 상관행렬 (correlation matrix)에 대해 분광분해를 수행한 결과와 동일하다.

```
eig_cor_raw <- eigen(cor(train_df[, -1]))
eig_cor_raw
```

```
## eigen() decomposition
## $values
## [1] 2.76146225 1.60565318 0.55056305 0.06406316 0.01825836
##
```

```

## $vectors
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] -0.07608427  0.77966993  0.0008915975 -0.140755404  0.60540325
## [2,]  0.39463007  0.56541218 -0.2953216494  0.117644166 -0.65078503
## [3,] -0.56970191  0.16228156  0.2412221065 -0.637721889 -0.42921686
## [4,]  0.55982770 -0.19654293 -0.2565972887 -0.748094314  0.14992183
## [5,]  0.44778451  0.08636803  0.8881182665 -0.003668418 -0.05711464

all(near(eig_cov_Z$values, eig_cor_raw$values))

## [1] TRUE

all(near(eig_cov_Z$vectors, eig_cor_raw$vectors))

## [1] TRUE

```

3.2.6 NIPALS 알고리즘

NIPALS(Nonlinear Iterative Parital Least Squares) 알고리즘은 반복적(iterative) 알고리즘을 이용하여 변동 기여율이 가장 큰 주성분부터 가장 작은 주성분까지 순차적으로 고유벡터와 주성분 스코어를 구하는 방법이다.

우선, 특이치 분해에서 사용한 식을 단순화하여, 분산조정된 행렬 \mathbf{Z} 가 아래와 같이 주성분 스코어 행렬 \mathbf{T} 와 고유벡터 행렬 \mathbf{V} 로 분해된다고 하자. (분산조정 대신 평균조정만을 원할 경우 \mathbf{Z} 대신 \mathbf{X} 를 사용)

$$\mathbf{Z} = \mathbf{UDV}^\top = \mathbf{TV}^\top$$

즉, 주성분 스코어 \mathbf{T} 는 아래와 같다.

$$\mathbf{T} = \mathbf{ZV}$$

```

T_mat <- Z %*% svd_Z$v
T_mat

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] -1.4870243  0.6066594 -0.63361774 -0.29625002  0.020293731
## [2,] -0.2063797 -0.0804627 -0.04965017  0.26323513  0.063581473
## [3,]  0.1968538  0.9704605 -0.39507856  0.27123746  0.103351746
## [4,] -2.3542884 -3.5056480  0.16252734  0.02524924 -0.249920974
## [5,] -0.8953707  1.4552899  1.36265905  0.20161775 -0.055517167
## [6,] -0.3682082 -0.5976313  0.65857722  0.27901317  0.060458248

```

```
## [7,] -0.9354306 -1.4144519 -0.82574638 0.07358977 0.095960908
## [8,] 2.4129728 -0.6785064 0.92207607 -0.36161577 -0.062593521
## [9,] 2.6991862 -0.7596591 -0.45091077 -0.21030378 0.168645128
## [10,] -0.4050098 0.2800099 -0.92835441 0.13993488 0.001811118
## [11,] 1.3958199 1.1353513 -0.09819177 0.34335126 -0.094986796
## [12,] -1.5381192 1.1576616 -0.07467334 -0.29404424 0.052430946
## [13,] 0.3217681 -0.2378023 1.10180230 0.28507243 0.030666763
## [14,] -2.0306806 0.9646122 0.20906175 -0.39639758 -0.085778570
## [15,] 3.0389460 0.8841645 -0.77478769 -0.04079854 -0.349688462
## [16,] 2.0064063 -1.2831337 0.64388897 -0.22077705 0.188366871
## [17,] -0.4211779 -0.2987099 -1.20644766 0.11766274 0.068250991
## [18,] -1.4302634 1.4017959 0.37686579 -0.17977686 0.044667568
```

NIPALS 알고리즘은 아래와 같이 주성분 스코어 행렬 \mathbf{T} 의 열과 고유벡터행렬 \mathbf{V} 의 열을 동시에 구한다.

- [단계 0] 반복알고리즘 수행을 위한 초기화를 한다. $h \leftarrow 1$, $\mathbf{Z}_h \leftarrow \mathbf{Z}$.
- [단계 1] 데이터 행렬 \mathbf{Z}_h 의 임의의 열 하나를 주성분 스코어 벡터 \mathbf{t}_h 로 선정한다.
- [단계 2] 로딩벡터를 구한다. $\mathbf{v}_h \leftarrow \mathbf{Z}_h \mathbf{t}_h / \sqrt{\mathbf{t}_h^\top \mathbf{t}_h}$
- [단계 3] 로딩벡터의 크기가 1이 되도록 한다. $\mathbf{v}_h \leftarrow \mathbf{v}_h / \sqrt{\mathbf{v}_h^\top \mathbf{v}_h}$
- [단계 4] 주성분 스코어 벡터를 로딩벡터에 기반하여 계산한다. $\mathbf{t}_h \leftarrow \mathbf{Z}_h \mathbf{v}_h$
- [단계 5] 주성분 스코어 벡터 \mathbf{t}_h 가 수렴하였으면 [단계 6]으로 진행하고, 그렇지 않으면 [단계 2]로 돌아간다.
- [단계 6] 데이터 행렬 \mathbf{Z}_h 로부터 새로 얻어진 주성분 벡터 \mathbf{t}_h 와 고유벡터 \mathbf{v}_h 가 설명하는 부분을 제거하고 나머지 변동만을 담은 새로운 데이터 행렬 \mathbf{Z}_{h+1} 을 구한다.

$$\mathbf{Z}_{h+1} \leftarrow \mathbf{Z}_h - \mathbf{t}_h \mathbf{v}_h^\top$$

- [단계 7] $h \leftarrow h + 1$ 로 업데이트하고, [단계 1]로 돌아간다. [단계 1] - [단계 7]의 과정을 \mathbf{Z} 의 rank 수만큼의 주성분을 얻을 때까지 반복한다.

위 반복 알고리즘을 수행하는 함수를 아래와 같이 구성해보자. 아래 함수에서 입력변수 \mathbf{X} 는 데이터 행렬으로, 평균조정된 행렬 \mathbf{X} 나 분산조정된 \mathbf{Z} 모두 사용 가능하다. 입력변수 r 은 추출하고자 하는 주성분의 개수이다.

```
nipals_pca <- function(X, r = NULL) {
  if (is_empty(r) || (r > min(dim(X)))) {
    r <- min(dim(X))
  }

  Th <- matrix(NA, nrow = nrow(X), ncol = r)
  Vh <- matrix(NA, nrow = r, ncol = r)
```

```

for (h in seq_len(r)) {
  # 단계 1
  j <- sample(ncol(X), 1)
  Th[, h] <- X[, j]

  while (TRUE) {
    # 단계 2
    Vh[, h] <- t(t(Th[, h]) %*% X / (norm(Th[, h], "2") ^ 2))

    # 단계 3
    Vh[, h] <- Vh[, h] / norm(Vh[, h], "2")

    # 단계 4
    th <- X %*% Vh[, h]

    # 단계 5
    if (all(near(Th[, h], th))) break
    Th[, h] <- th
  }

  #단계 6
  X <- X - Th[, h] %*% t(Vh[, h])
}

return(list(T = Th, V = Vh))
}

nipals_Z <- nipals_pca(Z)
nipals_Z

## $T
##      [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] -1.4870243 -0.6066594 -0.63361774  0.29625002 -0.02029373
## [2,] -0.2063797  0.0804627 -0.04965018 -0.26323513 -0.06358148
## [3,]  0.1968538 -0.9704605 -0.39507856 -0.27123746 -0.10335175
## [4,] -2.3542884  3.5056480  0.16252734 -0.02524925  0.24992097
## [5,] -0.8953707 -1.4552900  1.36265905 -0.20161775  0.05551716
## [6,] -0.3682082  0.5976313  0.65857722 -0.27901317 -0.06045825
## [7,] -0.9354306  1.4144520 -0.82574637 -0.07358976 -0.09596091
## [8,]  2.4129728  0.6785064  0.92207607  0.36161576  0.06259353
## [9,]  2.6991862  0.7596591 -0.45091077  0.21030379 -0.16864512
## [10,] -0.4050098 -0.2800099 -0.92835441 -0.13993488 -0.00181112
## [11,]  1.3958199 -1.1353513 -0.09819178 -0.34335127  0.09498679
## [12,] -1.5381192 -1.1576616 -0.07467334  0.29404424 -0.05243094
## [13,]  0.3217681  0.2378023  1.10180230 -0.28507243 -0.03066677

```

```

## [14,] -2.0306806 -0.9646122  0.20906176  0.39639757  0.08577858
## [15,]  3.0389460 -0.8841645 -0.77478770  0.04079852  0.34968846
## [16,]  2.0064063  1.2831337  0.64388897  0.22077706 -0.18836687
## [17,] -0.4211779  0.2987099 -1.20644766 -0.11766274 -0.06825099
## [18,] -1.4302634 -1.4017959  0.37686579  0.17977686 -0.04466756
##
## $V
##          [,1]          [,2]          [,3]          [,4]          [,5]
## [1,]  0.07608428  0.77966993  0.0008916027  0.140755413 -0.60540325
## [2,] -0.39463007  0.56541218 -0.2953216458 -0.117644174  0.65078502
## [3,]  0.56970191  0.16228156  0.2412221083  0.637721879  0.42921690
## [4,] -0.55982770 -0.19654293 -0.2565972891  0.748094319 -0.14992178
## [5,] -0.44778451  0.08636803  0.8881182671  0.003668408  0.05711464

```

위 분해된 행렬의 곱이 원 데이터 행렬 Z 과 일치하는지 확인해보자.

```
all(near(Z, nipals_Z$T %*% t(nipals_Z$V)))
```

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

R 패키지 `nipals`내의 함수 `nipals`가 이 장에서 설명한 NIPALS 알고리즘에 기반한 주성분 분석을 아래와 같이 제공한다.

```
library(nipals)
nipals(Z, center = FALSE, scale = FALSE)
```

```

## $eig
## [1] 6.8516317 5.2245674 3.0593417 1.0435869 0.5571285
##
## $scores
##          PC1         PC2         PC3         PC4         PC5
## [1,] -0.21705404 -0.11608067 -0.20710543  0.28388475 -0.036368396
## [2,] -0.03011834  0.01540551 -0.01623006 -0.25221776 -0.114174269
## [3,]  0.02869590 -0.18575892 -0.12913406 -0.25987079 -0.185560165
## [4,] -0.34348333  0.67105909  0.05310696 -0.02428657  0.448582844
## [5,] -0.13073246 -0.27851123  0.44541637 -0.19321728  0.099609669
## [6,] -0.05371865  0.11440401  0.21526389 -0.26733867 -0.108571464
## [7,] -0.13647563  0.27074924 -0.26991736 -0.07048155 -0.172255999
## [8,]  0.35219939  0.12981093  0.30139420  0.34648877  0.112419837
## [9,]  0.39397535  0.14532356 -0.14739177  0.20158102 -0.302663550
## [10,] -0.05912155 -0.05359220 -0.30344792 -0.13408884 -0.003277662
## [11,]  0.20367981 -0.21735017 -0.03209051 -0.32904441  0.170427304
## [12,] -0.22453126 -0.22153804 -0.02440174  0.28178254 -0.094052577
## [13,]  0.04697085  0.04551641  0.36014173 -0.27315578 -0.055099430

```

```

## [14,] -0.29641393 -0.18457147  0.06834143  0.37981073  0.154041866
## [15,]  0.44350417 -0.16932257 -0.25324815  0.03896920  0.627670066
## [16,]  0.29288258  0.24554714  0.21046020  0.21162296 -0.338060584
## [17,] -0.06146040  0.05717479 -0.39435062 -0.11272299 -0.122527440
## [18,] -0.20879845 -0.26826541  0.12319285  0.17228468 -0.080140048
##
## $loadings
##          PC1        PC2        PC3        PC4        PC5
## roa      0.07627711  0.7796534  0.0008551484  0.140974596 -0.60534928
## roe     -0.39449021  0.5654941 -0.2953469599 -0.117893972  0.65074198
## bis      0.56974203  0.1621586  0.2412197864  0.637556663  0.42945678
## de_ratio -0.55987629 -0.1964075 -0.2565837179  0.748154680 -0.14963952
## turnover -0.44776314  0.0865197  0.8881144365  0.003640124  0.05711403
##
## $fitted
## NULL
##
## $ncomp
## [1] 5
##
## $R2
## [1] 0.552292435 0.321130644 0.110112610 0.012812631 0.003651673
##
## $iter
## [1] 14 4 4 6 3

```

평균 및 분산조정 이전의 원래 데이터를 입력하고, 파라미터 `center`(평균조정) 및 `scale`(분산조정)의 값을 모두 TRUE로 설정하면 동일한 결과를 얻을 수 있다.

```

library(nipals)
nipals(train_df[, -1], center = TRUE, scale = TRUE)

## $eig
## [1] 6.8516317 5.2245674 3.0593417 1.0435869 0.5571285
##
## $scores
##          PC1        PC2        PC3        PC4        PC5
## [1,] -0.21705404 -0.11608067 -0.20710543  0.28388475 -0.036368396
## [2,] -0.03011834  0.01540551 -0.01623006 -0.25221776 -0.114174269
## [3,]  0.02869590 -0.18575892 -0.12913406 -0.25987079 -0.185560165
## [4,] -0.34348333  0.67105909  0.05310696 -0.02428657  0.448582844
## [5,] -0.13073246 -0.27851123  0.44541637 -0.19321728  0.099609669
## [6,] -0.05371865  0.11440401  0.21526389 -0.26733867 -0.108571464
## [7,] -0.13647563  0.27074924 -0.26991736 -0.07048155 -0.172255999
## [8,]  0.35219939  0.12981093  0.30139420  0.34648877  0.112419837

```

```

## [9,]  0.39397535  0.14532356 -0.14739177  0.20158102 -0.302663550
## [10,] -0.05912155 -0.05359220 -0.30344792 -0.13408884 -0.003277662
## [11,]  0.20367981 -0.21735017 -0.03209051 -0.32904441  0.170427304
## [12,] -0.22453126 -0.22153804 -0.02440174  0.28178254 -0.094052577
## [13,]  0.04697085  0.04551641  0.36014173 -0.27315578 -0.055099430
## [14,] -0.29641393 -0.18457147  0.06834143  0.37981073  0.154041866
## [15,]  0.44350417 -0.16932257 -0.25324815  0.03896920  0.627670066
## [16,]  0.29288258  0.24554714  0.21046020  0.21162296 -0.338060584
## [17,] -0.06146040  0.05717479 -0.39435062 -0.11272299 -0.122527440
## [18,] -0.20879845 -0.26826541  0.12319285  0.17228468 -0.080140048
##
## $loadings
##          PC1        PC2        PC3        PC4        PC5
## roa      0.07627711  0.7796534  0.0008551484  0.140974596 -0.60534928
## roe     -0.39449021  0.5654941 -0.2953469599 -0.117893972  0.65074198
## bis      0.56974203  0.1621586  0.2412197864  0.637556663  0.42945678
## de_ratio -0.55987629 -0.1964075 -0.2565837179  0.748154680 -0.14963952
## turnover -0.44776314  0.0865197  0.8881144365  0.003640124  0.05711403
##
## $fitted
## NULL
##
## $ncomp
## [1] 5
##
## $R2
## [1] 0.552292435 0.321130644 0.110112610 0.012812631 0.003651673
##
## $iter
## [1] 14 4 4 6 3

```

3.3 주성분 회귀분석

2.2 장에서 살펴본 다중회귀모형의 식 (2.2)을 아래에 다시 살펴보자.

$$\mathbf{y} = \mathbf{X}\beta + \epsilon \quad (3.6)$$

여기서, β 와 ϵ 는 각각 회귀계수와 오차항을 나타내는 벡터이며, 독립변수 데이터 행렬 \mathbf{X} 와 종속변수 관측치 벡터 \mathbf{y} 모두 평균조정한 데이터라 간주하자. \mathbf{X} 의 열벡터 간 다중공선성(multicollinearity)이 높으면 최소자승법에 의한 β 의 추정치의 분산이 커지는 문제가 있으며, \mathbf{X} 행렬의 관측수보다 변수 수가 많을 때는 β 추정치를 구할 수 없다. 이 문제를 해결하기 위해 주성분 회귀분석(principal component regression; PCR)에서는 \mathbf{X} 변동 대부분을 설명하는 A 개 ($A \leq \text{rank}(\mathbf{X})$)의 주성분 스코어를 다음과 같이 독립변수로 사용한다.

$$\mathbf{y} = q_1 \mathbf{t}_1 + q_2 \mathbf{t}_2 + \cdots + q_A \mathbf{t}_A + \mathbf{f} \quad (3.7)$$

여기서 \mathbf{f} 는 오차항을 나타내는 벡터이며, q_1, \dots, q_A 는 회귀계수들이다. 이 때, A 개의 주성분 스코어로 구성되는 $(n \times A)$ 주성분행렬을 $\mathbf{T}_A = [\mathbf{t}_1 \cdots \mathbf{t}_A]$ 로, 회귀계수벡터를 $\mathbf{q} = [q_1 \cdots q_A]^\top$ 으로 표기하면, 식 (3.7)의 모형은 다음과 같이 표현된다.

$$\mathbf{y} = \mathbf{T}_A \mathbf{q}_A + \mathbf{f} \quad (3.8)$$

위 모형은 다중회귀모형으로 볼 수 있으므로, 다중회귀모형에 대한 모든 이론이 적용될 수 있다. 또한 위 모형에서 각 주성분 스코어 벡터 $\mathbf{t}_1, \dots, \mathbf{t}_A$ 는 서로 선형 독립적 (linearly independent)이므로, 회귀성 검정이 용이한 측면이 있다.

3.3.1 기본 R 스트립트

3개의 독립변수와 1개의 종속변수 (y)를 관측한 데이터가 아래와 같다고 하자.

```
train_df <- tribble(
  ~x1, ~x2, ~x3, ~y,
  -3, -3, 5, -30,
  -2, -3, 7, -20,
  0, 0, 4, 0,
  1, 2, 0, 5,
  2, 2, -5, 10,
  2, 2, -11, 35
)

knitr::kable(
  train_df, booktabs = TRUE,
  align = rep("r", ncol(train_df)),
  caption = "주성분 회귀분석 예제 데이터"
)
```

3개의 독립변수로 이루어진 데이터에서 2개의 주성분만을 추출하여 회귀모형을 추정하여 보자.

```
pcr_fit <- pls::pcr(y ~ x1 + x2 + x3, data = train_df, ncomp = 2)
coef(pcr_fit, intercept = TRUE)

## , , 2 comps
##
##          y
## (Intercept) 0.000000
```

Table 3.2: 주성분 회귀분석 예제 데이터

x1	x2	x3	y
-3	-3	5	-30
-2	-3	7	-20
0	0	4	0
1	2	0	5
2	2	-5	10
2	2	-11	35

```
## x1      2.130440
## x2      2.721789
## x3     -1.737825
```

위 회귀계수들은 주성분을 이용하여 추정한 회귀 모형을 원래 독립변수를 이용한 회귀식(평균조정 이전)으로 다시 선형변환한 결과이다. 이에 대해서는 다음 절에서 좀 더 자세히 살펴보도록 하자.

```
summary(pcr_fit)
```

```
## Data: X dimension: 6 3
## Y dimension: 6 1
## Fit method: svdpc
## Number of components considered: 2
## TRAINING: % variance explained
##    1 comps  2 comps
## X    94.98    99.79
## y    87.94    91.31
```

위 요약표는 하나의 주성분과 두 개의 주성분을 이용하였을 때 추정된 회귀모형들이 종속 변수의 총 변량을 각각 87.9415591%와 91.3101613% 만큼을 설명함을 알려준다.

3.3.2 주성분 회귀계수 추정

우선 Table 3.2의 세 독립변수에 대해 주성분 분석을 수행하여 두 개의 주성분을 추출하자.

```
pca_fit <- prcomp(train_df[, c("x1", "x2", "x3")], rank. = 2,
                     center = TRUE, scale. = FALSE)
pca_fit$x
```

```
##          PC1         PC2
```

```
## [1,] -6.2346992 1.9880169
## [2,] -7.8320036 0.6817026
## [3,] -3.6996775 -1.5151642
## [4,]  0.8208672 -2.0392493
## [5,]  5.6979984 -0.6940262
## [6,] 11.2475146 1.5787202
```

또한 평균조정된 종속변수 벡터를 계산하자.

```
y_centered <- train_df$y - mean(train_df$y)
y_centered
```

```
## [1] -30 -20   0   5  10  35
```

주성분 스코어와 평균조정된 종속변수를 이용하여 회귀모형을 추정하자. 이 때, intercept 가 없는 모형을 가정한다.

```
pc_lm_fit <- lm(y_centered ~ - 1 + pca_fit$x)
coef(pc_lm_fit)
```

```
## pca_fit$xPC1 pca_fit$xPC2
##      2.918798    -2.539206
```

위 회귀계수 벡터가 식 (3.8)의 회귀계수 벡터 \mathbf{q}_A 의 값이다 ($A = 2$).

3.3.3 회귀계수 선형변환

앞장에서 얻어진 주성분을 이용한 회귀식을 원 데이터에서 관측된 독립변수와 종속변수에 대한 식으로 변환하여 보자.

각 주성분은 평균조정된 독립변수들의 선형조합으로 아래와 같이 얻어진다.

```
pca_fit$rotation
```

```
##          PC1         PC2
## x1  0.2525343 -0.5487321
## x2  0.2841664 -0.7452586
## x3 -0.9249194 -0.3787911
```

따라서, 아래와 같이 주성분에 대한 회귀계수를 원래 독립변수(평균조정 이후)에 대한 회귀계수로 변환할 수 있다.

```
beta_x <- pca_fit$rotation %*% coef(pc_lm_fit)
beta_x

##          [,1]
## x1  2.130440
## x2  2.721789
## x3 -1.737825
```

Intercept는 평균조정 이전 종속변수의 평균에서 위 회귀계수벡터를 평균조정 이전 독립 변수의 평균벡터와 곱한 결과를 뺀 값이다.

```
mean(train_df$y) - colMeans(train_df[, c("x1", "x2", "x3")]) %*% beta_x

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

본 장에서 사용한 Table 3.2는 이미 평균조정이 되어 있어서 Intercept가 0으로 추정된다.

본 장에서 분산조정된 주성분에 대한 회귀계수 변환은 다루지 않았으나, 이 또한 간단하게 변환할 수 있다.

Chapter 4

부분최소자승법

회귀분석에서와 같이 하나의 종속변수에 영향을 주는 k 개의 독립변수가 있다고 하자. 모든 변수는 평균조정되었다고 간주한다. 본 장에서 다루고자 하는 부분최소자승법(partial least squares: PLS)은 앞에서 다룬 주성분 회귀분석(PCR)과 유사하나, 도출되는 새로운 잠재변수들이 다르다.

독립변수와 종속변수간의 관계를 설명하기 위해, 우선 독립변수 행렬과 종속변수 행렬(또는 벡터)가 각각 서로 다른 잠재변수들에 의해 설명된다고 가정한 뒤, 두 잠재변수들간의 관계에 대한 모형을 세운다. 이 때, 본 장에서는 두 잠재변수들간의 관계가 선형인 모형(선형 PLS)만을 살펴본다.

4.1 필요 R 패키지 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
pls	2.7-1

4.2 하나의 종속변수의 경우

4.2.1 기본 R 스크립트

앞 장의 주성분 회귀분석에서 사용했던 데이터에 대해 부분최소자승 회귀분석을 수행해보도록 하자.

```
train_df <- tribble(  
  ~x1, ~x2, ~x3, ~y,
```

Table 4.1: 부분최소자승 회귀분석 예제 데이터

x1	x2	x3	y
-3	-3	5	-30
-2	-3	7	-20
0	0	4	0
1	2	0	5
2	2	-5	10
2	2	-11	35

```

-3, -3, 5, -30,
-2, -3, 7, -20,
0, 0, 4, 0,
1, 2, 0, 5,
2, 2, -5, 10,
2, 2, -11, 35
)

knitr::kable(
  train_df, booktabs = TRUE,
  align = rep("r", ncol(train_df)),
  caption = "부분최소자승 회귀분석 예제 데이터"
)

```

R 패키지 `pls` 내의 함수 `plsr()`을 이용하여 PLS 모형을 아래와 같이 추정할 수 있다.

```
plsr_fit <- pls::plsr(y ~ x1 + x2 + x3, data = train_df, ncomp = 2)
coef(plsr_fit)
```

```

## , , 2 comps
##
##          y
## x1  2.475395
## x2  2.523238
## x3 -1.704636

```

수행결과 object에 `summary()` 함수를 사용하여 학습된 모형의 독립변수 **X** 및 종속변수 **y**의 총변동에 대한 기여율을 확인할 수 있다.

```
summary(plsr_fit)
```

```
## Data:      X dimension: 6 3
```

```

## Y dimension: 6 1
## Fit method: kernelpls
## Number of components considered: 2
## TRAINING: % variance explained
##    1 comps  2 comps
## X      94.97   99.78
## y      88.28   91.46

```

위 요약표는 하나의 잠재변수와 두 개의 잠재변수를 이용하였을 때 추정된 회귀모형들이 종속변수의 총 변량을 각각 88.2814529%와 91.4578959% 만큼을 설명함을 알려준다. 이는 앞 장에서 살펴보았던 주성분 회귀모형보다 더 높은 수치이다.

4.2.2 PLS 모형

종속변수가 하나만 존재하는 경우에는 데이터 행렬 \mathbf{X} 와 종속변수 벡터 \mathbf{y} 가 동일한 잠재 변수로 설명된다고 가정할 수 있다. $(n \times k)$ 데이터 행렬 \mathbf{X} 와 종속변수 벡터 \mathbf{y} 에 대하여 동시에 A 개의 잠재변수 벡터 $\mathbf{t}_1, \dots, \mathbf{t}_A$ 로 설명하는 모형을 아래와 같이 기술해보자.

$$\mathbf{X} = \mathbf{t}_1 \mathbf{p}_1^\top + \mathbf{t}_2 \mathbf{p}_2^\top + \dots + \mathbf{t}_A \mathbf{p}_A^\top + \mathbf{E} \quad (4.1)$$

$$\mathbf{y} = \mathbf{t}_1 b_1 + \mathbf{t}_2 b_2 + \dots + \mathbf{t}_A b_A + \mathbf{f} \quad (4.2)$$

여기서 계수벡터 \mathbf{p}_a 는 \mathbf{X} 에 해당하는 로딩 (loading)을, 그리고 계수 b_a 는 \mathbf{y} 에 해당하는 로딩을 나타내며, \mathbf{E} 와 \mathbf{f} 는 각 모형에 해당하는 오차항 (행렬 또는 벡터)이다.

위 모형을 $(n \times A)$ 잠재변수 행렬 $\mathbf{T} = [\mathbf{t}_1 \dots \mathbf{t}_A]$ 와 $(k \times A)$ 로딩행렬 $\mathbf{P} = [\mathbf{p}_1 \dots \mathbf{p}_A]$, 그리고 로딩벡터 $\mathbf{b} = [b_1 \dots b_A]^\top$ 을 이용하여 아래와 같이 행렬식으로 나타낼 수 있다.

$$\mathbf{X} = \mathbf{TP}^\top + \mathbf{E} \quad (4.3)$$

$$\mathbf{y} = \mathbf{ Tb} + \mathbf{f} \quad (4.4)$$

4.2.3 NIPALS 알고리즘

- [단계 0] 반복알고리즘 수행을 위한 초기화를 한다. $a \leftarrow 1$, $\mathbf{X}_a \leftarrow \mathbf{X}$, $\mathbf{y}_a \leftarrow \mathbf{y}$.
- [단계 1] \mathbf{X}_a 을 다중종속변수 행렬으로, \mathbf{y}_a 를 독립변수 벡터로 하는 회귀모형으로부터 기울기 $\mathbf{w}_a = [w_{a1} \dots w_{ak}]^\top$ 를 산출한다.

$$\mathbf{w}_a \leftarrow \mathbf{X}_a^\top \mathbf{y}_a / \mathbf{y}_a^\top \mathbf{y}_a$$

- [단계 2] 기울기 벡터 \mathbf{w}_a 의 크기가 1이 되도록 한다.

$$\mathbf{w}_a \leftarrow \mathbf{w}_a / \sqrt{\mathbf{w}_a^\top \mathbf{w}_a}$$

- [단계 3] 잠재변수 \mathbf{t}_a 를 행렬 \mathbf{X}_a 의 각 열의 가중평균으로 구한다. 이 때, 가중치는 기울기 벡터 \mathbf{w}_a 를 이용한다.

$$\mathbf{t}_a \leftarrow \mathbf{X}_a \mathbf{w}_a$$

- [단계 4] 식 (4.1)와 같이 \mathbf{X}_a 을 다중종속변수 행렬으로, \mathbf{t}_a 를 독립변수 벡터로 하는 회귀모형으로부터 로딩벡터 \mathbf{p}_a 를 구한다.

$$\mathbf{p}_a \leftarrow \mathbf{X}_a^\top \mathbf{t}_a / \mathbf{t}_a^\top \mathbf{t}_a$$

- [단계 5] 로딩벡터 \mathbf{p}_a 의 크기를 1로 조정하고, 잠재변수 벡터 \mathbf{t}_a 와 기울기 벡터 \mathbf{w}_a 의 크기를 그에 따라 보정한다.

$$d \leftarrow \sqrt{\mathbf{p}_a^\top \mathbf{p}_a}, \mathbf{t}_a \leftarrow \mathbf{t}_a d, \mathbf{w}_a \leftarrow \mathbf{w}_a d, \mathbf{p}_a \leftarrow \frac{1}{d} \mathbf{p}_a$$

- [단계 6] 식 (4.2)와 같이 잠재변수 \mathbf{t}_a 를 종속변수 \mathbf{y}_a 에 회귀시킬 때 계수 b_a 를 산출한다.

$$b_a \leftarrow \mathbf{y}_a^\top \mathbf{t}_a / \mathbf{t}_a^\top \mathbf{t}_a$$

- [단계 7] 독립변수 행렬 \mathbf{X}_a 와 종속변수 벡터 \mathbf{y}_a 로부터 새로 얻어진 잠재변수 벡터 \mathbf{t}_a 가 설명하는 부분을 제거하고 나머지 변동만을 담은 독립변수 행렬 \mathbf{X}_{a+1} 과 종속변수 벡터 \mathbf{y}_{a+1} 을 구한다.

$$\mathbf{X}_{a+1} \leftarrow \mathbf{X}_a - \mathbf{t}_a \mathbf{p}_a^\top, \mathbf{y}_{a+1} \leftarrow \mathbf{y}_a - \mathbf{t}_a b_a$$

- [단계 8] $a \leftarrow a + 1$ 로 업데이트하고, [단계 1]로 돌아간다. [단계 1] - [단계 8]의 과정을 A 개의 잠재변수를 얻을 때까지 반복한다.

위 NIPALS 알고리즘을 아래 `nipals_plsr`이라는 함수로 구현해보자. 이 때, 함수의 입력변수는 아래와 같다.

- X: 평균조정된 $(n \times k)$ 행렬
- y: 평균조정된 종속변수 벡터
- A: 잠재변수 개수

```
nipals_plsr <- function(X, y, A = NULL) {
  if (is_empty(A) || (A > min(dim(X)))) {
    A <- min(dim(X))
  }

  Tmat <- matrix(NA, nrow = nrow(X), ncol = A)
  Wmat <- matrix(NA, nrow = ncol(X), ncol = A)
  Pmat <- matrix(NA, nrow = ncol(X), ncol = A)
  b <- vector("numeric", length = A)

  for (a in seq_len(A)) {
    # 단계 1
```

```

Wmat[, a] <- coef(lm(X ~ -1 + y))

# 단계 2
Wmat[, a] <- Wmat[, a] / sqrt(sum(Wmat[, a]^2))

# 단계 3
Tmat[, a] <- X %*% Wmat[, a]

# 단계 4
Pmat[, a] <- coef(lm(X ~ -1 + Tmat[, a]))

# 단계 5
p_size <- sqrt(sum(Pmat[, a]^2))
Tmat[, a] <- Tmat[, a] * p_size
Wmat[, a] <- Wmat[, a] * p_size
Pmat[, a] <- Pmat[, a] / p_size

# 단계 6
b[a] <- coef(lm(y ~ -1 + Tmat[, a]))

# 단계 7
X <- X - Tmat[, a] %*% t(Pmat[, a])
y <- y - Tmat[, a] %*% t(b[a])
}

return(list(T = Tmat, W = Wmat, P = Pmat, b = b))
}

X <- as.matrix(train_df[, c("x1", "x2", "x3")])
y <- train_df$y
nipals_fit <- nipals_plsr(X, y, A = 2)
nipals_fit

## $T
##          [,1]      [,2]
## [1,] -6.3243200 -2.0380766
## [2,] -7.8584372 -0.5965965
## [3,] -3.6317877  1.5429616
## [4,]  0.9079469  1.9745198
## [5,]  5.7294582  0.7158171
## [6,] 11.1771398 -1.5986253
##
## $W
##          [,1]      [,2]
## [1,]  0.2817766  0.6579688

```

```

## [2,]  0.3130851 0.6388931
## [3,] -0.9079469 0.4245052
##
## $P
##           [,1]      [,2]
## [1,]  0.2537679 0.5424154
## [2,]  0.2858180 0.7279599
## [3,] -0.9240724 0.4193565
##
## $b
## [1] 2.924570 2.464658

```

식 (4.3)과 (4.4)에서, 잠재변수 행렬 \mathbf{T} 가 주어졌다 가정할 때 로딩행렬 및 벡터 \mathbf{P} 와 \mathbf{b} 는 아래와 같이 추정된다.

$$\hat{\mathbf{P}}^\top = (\mathbf{T}^\top \mathbf{T})^{-1} \mathbf{T}^\top \mathbf{X} \quad (4.5)$$

$$\hat{\mathbf{b}} = (\mathbf{T}^\top \mathbf{T})^{-1} \mathbf{T}^\top \mathbf{y} \quad (4.6)$$

위 NIPALS 알고리즘 수행 결과에서 이를 확인해보자.

```
P_hat <- t(solve(t(nipals_fit$T) %*% nipals_fit$T) %*% t(nipals_fit$T) %*% X)
all(near(nipals_fit$P, P_hat))
```

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

```
b_hat <- as.vector(t(solve(t(nipals_fit$T) %*% nipals_fit$T) %*%
                           t(nipals_fit$T) %*% as.matrix(y, ncol = 1)))
all(near(nipals_fit$b, b_hat))
```

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

4.2.4 회귀식 변환

위 NIPALS 알고리즘 수행 결과를 원래 독립변수 \mathbf{X} 와 종속변수 \mathbf{y} 에 대한 식으로 변환하는 방법은 아래와 같다.

잠재변수행렬 \mathbf{T} 는 아래와 같이 독립변수 행렬 \mathbf{X} 와 가중치행렬 \mathbf{W} , 그리고 로딩행렬 \mathbf{P} 의 연산으로 표현된다.

$$\mathbf{T} = \mathbf{X}\mathbf{W}(\mathbf{P}^\top \mathbf{W})^{-1} \quad (4.7)$$

이를 식 (4.4)에 대입하면,

$$\begin{aligned} \mathbf{y} &= \mathbf{X}\mathbf{W}(\mathbf{P}^\top\mathbf{W})^{-1}\mathbf{b} + \mathbf{f} \\ &= \mathbf{X}\boldsymbol{\beta}_{PLS} + \mathbf{f} \end{aligned} \quad (4.8)$$

따라서, 원 독립변수 행렬 \mathbf{X} 에 대한 회귀계수는 아래와 같이 정리된다.

$$\boldsymbol{\beta}_{PLS} = \mathbf{W}(\mathbf{P}^\top\mathbf{W})^{-1}\mathbf{b} \quad (4.9)$$

```
beta_pls <- nipals_fit$W %*%
  solve(t(nipals_fit$P) %*% nipals_fit$W) %*%
  as.matrix(nipals_fit$b, ncol = 1L)
beta_pls

##          [,1]
## [1,]  2.475395
## [2,]  2.523238
## [3,] -1.704636
```

4.2.5 제곱합 분해

A 개의 잠재변수를 사용하는 모형에 대하여 모형이 설명하는 \mathbf{y} 의 변동(제곱합)을 SSR , 설명하지 못하는 변동을 SSE 라 할 때, \mathbf{y} 의 전체제곱합(SST)은 다음과 같이 분해된다.

$$SST = SS(\mathbf{y}) = SSR + SSE$$

여기서 $SS()$ 는 제곱합 함수로, 임의의 벡터 \mathbf{x} 에 대해 아래와 같이 정의된다.

$$SS(\mathbf{x}) = \mathbf{x}^\top \mathbf{x}$$

이 때, SSR 은 다음과 같이 산출할 수 있다.

$$\begin{aligned} SSR &= \sum_{a=1}^A SS(b_a \mathbf{t}_a) \\ &= \sum_{a=1}^A b_a^2 SS(\mathbf{t}_a) \end{aligned} \quad (4.10)$$

a 번째 잠재변수 \mathbf{t}_a 가 \mathbf{y} 를 설명하는 회귀제곱합을 $SSR_a = b_a^2 SS(\mathbf{t}_a)$ 라 할 때, SSR 은 아래와 같이 분해된다.

$$SSR = \sum_{a=1}^A SSR_a$$

위 예제에서 2개의 잠재변수가 설명하는 \mathbf{y} 의 총변동을 PLS 결과를 이용하여 계산하면 아래와 같다.

```
SSR_a <- nipals_fit$b ^ 2 * diag(t(nipals_fit$T) %*% nipals_fit$T)
```

```
## [1] 2339.45850 84.17574
```

이 때, 각 주성분이 설명하는 \mathbf{y} 변동의 기여율을 아래와 같이 정의한다.

$$\Delta R_a^2 = \frac{SSR_a}{SST} \quad (4.11)$$

앞 예제에서 각 잠재변수의 \mathbf{y} 변동에 대한 기여율을 계산해보자.

```
SST <- sum(y ^ 2)
delta_Rsq <- SSR_a / SST
delta_Rsq
```

```
## [1] 0.88281453 0.03176443
```

잠재변수 A 개를 이용한 PLS 모형이 설명하는 \mathbf{y} 의 총 변동에 대한 기여도(SSR/SST)은 아래와 같이 각 잠재변수의 기여도의 합으로 산출된다.

$$R^2 = \frac{SSR}{SST} = \frac{\sum_{a=1}^A SSR_a}{SST} = \sum_{a=1}^A \Delta R_a^2$$

따라서, 잠재변수 A 개를 이용한 PLS 모형이 설명하는 \mathbf{y} 의 총 변동에 대한 기여도(SSR/SST)은 아래와 같이 각 잠재변수의 기여도의 합으로 산출된다.

앞 예제에서 잠재변수 2개를 이용한 최종모형이 설명하는 \mathbf{y} 의 변동은 아래와 같다.

```
sum(delta_Rsq)
```

```
## [1] 0.914579
```

이는 앞 4.2.1절에서 R 패키지 `pls`를 이용하여 얻어진 결과와 동일함을 확인할 수 있다.

한편, 잠재변수들이 독립변수행렬 \mathbf{X} 의 변동을 얼마나 설명하는지 동시에 검토할 필요가 있다. 각 잠재변수들의 제곱합 $SS(\mathbf{t}_a)$ 의 \mathbf{X} 의 총변동 ($SS(\mathbf{X})$)에 대한 비율이 그 기여율을 설명한다.

앞 예제에서 각각의 잠재변수의 \mathbf{X} 에 대한 기여율은 아래와 같다.

```
diag(t(nipals_fit$T) %*% nipals_fit$T) / sum(diag(t(X) %*% X))

## [1] 0.94972728 0.04811507

잠재변수 2개를 이용한 PLS 모형의  $\mathbf{X}$ 에 대한 기여율은 아래와 같다.

sum(diag(t(nipals_fit$T) %*% nipals_fit$T)) / sum(diag(t(X) %*% X))

## [1] 0.9978423
```

잠재변수 2개가 독립변수행렬의 대부분의 변동을 설명함을 알 수 있으며, 위 결과는 역시 앞 4.2.1절에서 R 패키지 `pls`를 이용하여 얻어진 결과와 동일함을 확인할 수 있다.

4.2.6 독립변수의 중요도

원래의 각 독립변수가 종속변수를 설명하는 데 얼마나 영향을 미치는지는 공정분석 등에서 매우 중요하다. j 번째 독립변수의 종속변수에 대한 중요도 척도로 VIP (variable importance in projection)를 다음과 같이 정의한다.

$$\begin{aligned} VIP_j &= \sqrt{\frac{k}{SSR} \sum_{a=1}^A SSR_a (w_{aj}/\|\mathbf{w}_a\|)^2} \\ &= \sqrt{\frac{k}{SSR} \sum_{a=1}^A SSR_a \frac{w_{aj}^2}{\mathbf{w}_a^\top \mathbf{w}_a}} \end{aligned} \quad (4.12)$$

위 정의에 의하면 다음이 성립한다.

$$\sum_{j=1}^k VIP_j^2 = k$$

즉, 독립변수당 중요도 제곱의 평균은 1이 된다. 이에 따라, 통상 VIP 가 1보다 큰 독립변수를 의미있는 변수로 간주한다.

앞 예제에 대해 각 변수의 중요도를 계산해보자.

```

k <- ncol(X)
VIP <- sqrt(
  colSums(
    k / sum(SSR_a) * SSR_a *
      (t(nipals_fit$W ^ 2) /
        diag(t(nipals_fit$W) %*% nipals_fit$W))
  )
)
VIP
## [1] 0.5231421 0.5700787 1.5496234

```

즉, x_3 가 가장 영향력 있는 변수라 할 수 있겠다.

4.3 다수의 종속변수의 경우

m 개의 종속변수가 존재하여, 종속변수 데이터가 벡터가 아닌 ($n \times m$) 행렬

$$\mathbf{Y} = [\mathbf{y}_1 \cdots \mathbf{y}_m]$$

으로 표현될 때, 각각의 종속변수에 대해 따로 잠재변수를 산출하기보다는, 여러 종속변수를 설명하는 공통의 잠재변수행렬 \mathbf{T} 를 산출하는 것이 합리적이라 할 수 있다.

4.3.1 기본 R 스크립트

4개의 독립변수 및 2개의 종속변수에 대한 평균조정된 데이터가 다음과 같다.

```

train_df <- tribble(
  ~x1, ~x2, ~x3, ~x4, ~y1, ~y2,
  -1, -0.5, -1, 1, 5.9, -10,
  1, 1.1, -6, -6, -3.7, -2,
  0, 0.3, -5, -2, 1, 11,
  -3, -3.2, -9, 19, 7.7, -22,
  4, 1.2, 14, -12, -7.5, 4,
  -2, -2.6, -2, 9, 2.8, 1,
  1, 3.7, 9, -9, -6.2, 18
)

knitr::kable(
  train_df, booktabs = TRUE,
  align = rep("r", ncol(train_df)),
  caption = "다수의 종속변수에 대한 부분최소자승 회귀분석 예제 데이터"
)

```

Table 4.2: 다수의 종속변수에 대한 부분최소자승 회귀분석 예제 데이터

x1	x2	x3	x4	y1	y2
-1	-0.5	-1	1	5.9	-10
1	1.1	-6	-6	-3.7	-2
0	0.3	-5	-2	1.0	11
-3	-3.2	-9	19	7.7	-22
4	1.2	14	-12	-7.5	4
-2	-2.6	-2	9	2.8	1
1	3.7	9	-9	-6.2	18

앞서 하나의 종속변수를 다루는 경우와 마찬가지로, R 패키지 pls 내의 함수 plsr() 을 이용하여 PLS 모형을 추정할 수 있다. 이 때, formula 입력 시 종속변수의 행렬을 이용한다.

```
X <- as.matrix(train_df[, c("x1", "x2", "x3", "x4")])
Y <- as.matrix(train_df[, c("y1", "y2")])
plsrmulti_fit <- pls::plsr(Y ~ X, ncomp = 3)
```

함수 수행 결과 추정된 PLS 모형으로부터 원 독립변수 x_1, \dots, x_4 와 종속변수 y_1, y_2 간의 선형관계를 함수 coef() 를 적용하여 아래와 같이 얻을 수 있다.

```
coef(plsrmulti_fit)
```

```
## , , 3 comps
##
##          y1           y2
## x1 -0.26509645 -2.8773570
## x2  0.08864959  2.7249194
## x3 -0.14258488  0.3377420
## x4  0.37330470 -0.7406377
```

또한 summary() 함수를 적용하여 잠재변수들이 독립변수행렬 및 각 종속변수의 총변동에 기여하는 비율을 확인할 수 있다.

```
summary(plsrmulti_fit)
```

```
## Data:    X dimension: 7 4
## Y dimension: 7 2
## Fit method: kernelpls
## Number of components considered: 3
## TRAINING: % variance explained
```

```
##      1 comps  2 comps  3 comps
## X     86.59    99.00   99.81
## y1    81.71    81.89   82.19
## y2    53.98    56.50   65.99
```

4.3.2 PLS 모형

앞 4.2.2절의 모형을 일반화하여 아래와 같은 모형을 가정한다.

$$\mathbf{X} = \mathbf{T}\mathbf{P}^\top + \mathbf{E} \quad (4.13)$$

$$\mathbf{Y} = \mathbf{U}\mathbf{Q}^\top + \mathbf{F} \quad (4.14)$$

$$\mathbf{U} = \mathbf{T}\mathbf{B} + \mathbf{H} \quad (4.15)$$

식 (4.13)의 모형은 앞서 하나의 종속변수의 경우에서 살펴본 모형식 (4.3)과 동일하다. 식 (4.14)에서 $(n \times A)$ 행렬 \mathbf{U} 는 \mathbf{Y} 를 설명하는 A 개의 잠재변수를 나타내는 행렬이며, $(m \times A)$ 행렬 \mathbf{Q} 는 종속변수행렬 \mathbf{Y} 와 잠재변수행렬 \mathbf{U} 간의 선형관계를 나타내는 로딩 행렬이다. 또한 식 (4.15)은 잠재변수행렬 \mathbf{T} 와 \mathbf{U} 간의 선형관계를 나타내는데, 특히 \mathbf{B} 는 $(A \times A)$ 대각행렬(diagonal matrix)로써, \mathbf{U} 와 \mathbf{T} 간에는 서로 대응하는 열 간에만 관계가 성립하며, 그 관계는 아래와 같다.

$$\mathbf{u}_a = b_a \mathbf{t}_a + \mathbf{h}_a, \quad a = 1, \dots, A$$

이 때, b_a 는 행렬 \mathbf{B} 의 a 번째 대각 원소를 나타낸다.

$$\mathbf{B} = \begin{bmatrix} b_1 & 0 & \cdots & 0 \\ 0 & b_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & b_A \end{bmatrix}$$

행렬 $\mathbf{E}, \mathbf{F}, \mathbf{H}$ 는 오차항에 해당하는 행렬이다.

4.3.3 NIPALS 알고리즘

다수의 종속변수가 존재하는 경우에도 NIPALS 알고리즘을 이용하여 모형을 추정한다. 이때는 각 잠재변수 \mathbf{t}_a 를 추출할 때 추출한 잠재변수의 수렴 여부를 확인할 필요가 없었던 위 4.2.3절의 경우와는 달리, 각 잠재변수 \mathbf{t}_a 와 \mathbf{u}_a 를 추출하는 과정에서 반복적인 (iterative) 기법으로 두 잠재변수 벡터들을 업데이트하며 수렴 여부를 확인하여야 한다.

- [단계 0] 반복알고리즘 수행을 위한 초기화를 한다. $a \leftarrow 1, \mathbf{X}_a \leftarrow \mathbf{X}, \mathbf{Y}_a \leftarrow \mathbf{Y}$.
- [단계 1] 종속변수 행렬 \mathbf{Y}_a 의 임의의 열 하나를 잠재변수 벡터 \mathbf{u}_a 로 선정한다.

- [단계 2] \mathbf{X}_a 을 다중종속변수 행렬으로, 잠재변수 \mathbf{u}_a 를 독립변수 벡터로 하는 회귀 모형으로부터 기울기 $\mathbf{w}_a = [w_{a1} \cdots w_{ak}]^\top$ 를 산출한다.

$$\mathbf{w}_a \leftarrow \mathbf{X}_a^\top \mathbf{u}_a / \mathbf{u}_a^\top \mathbf{u}_a$$

- [단계 3] 기울기 벡터 \mathbf{w}_a 의 크기가 1이 되도록 한다.

$$\mathbf{w}_a \leftarrow \mathbf{w}_a / \sqrt{\mathbf{w}_a^\top \mathbf{w}_a}$$

- [단계 4] 잠재변수 \mathbf{t}_a 를 행렬 \mathbf{X}_a 의 각 열의 가중평균으로 구한다. 이 때, 가중치는 기울기 벡터 \mathbf{w}_a 를 이용한다.

$$\mathbf{t}_a \leftarrow \mathbf{X}_a \mathbf{w}_a$$

- [단계 5] \mathbf{Y}_a 을 다중종속변수 행렬으로, 잠재변수 \mathbf{t}_a 를 독립변수 벡터로 하는 회귀 모형으로부터 기울기 (로딩벡터) $\mathbf{q}_a = [q_{a1} \cdots q_{am}]^\top$ 를 산출한다.

$$\mathbf{q}_a \leftarrow \mathbf{Y}_a^\top \mathbf{t}_a / \mathbf{t}_a^\top \mathbf{t}_a$$

- [단계 6] 기울기 벡터 \mathbf{q}_a 의 크기가 1이 되도록 한다.

$$\mathbf{q}_a \leftarrow \mathbf{q}_a / \sqrt{\mathbf{q}_a^\top \mathbf{q}_a}$$

- [단계 7] 잠재변수 \mathbf{u}_a 를 행렬 \mathbf{Y}_a 의 각 열의 가중평균으로 구한다. 이 때, 가중치는 기울기 벡터 \mathbf{q}_a 를 이용한다.

$$\mathbf{u}_a \leftarrow \mathbf{Y}_a \mathbf{q}_a$$

- [단계 8] (수렴 확인) [단계 2]에서 [단계 7]까지의 과정을 잠재변수 벡터 \mathbf{u}_a 의 모든 원소값이 수렴할 때까지 반복한다. 수렴이 확인되면 [단계 9]로 진행한다.

- [단계 9] \mathbf{t}_a 를 \mathbf{X}_a 에 회귀시켜, \mathbf{X}_a 을 다중종속변수 행렬으로, \mathbf{t}_a 를 독립변수 벡터로 하는 회귀모형으로부터 로딩벡터 \mathbf{p}_a 를 구한다.

$$\mathbf{p}_a \leftarrow \mathbf{X}_a^\top \mathbf{t}_a / \mathbf{t}_a^\top \mathbf{t}_a$$

- [단계 10] 로딩벡터 \mathbf{p}_a 의 크기를 1로 조정하고, 잠재변수 벡터 \mathbf{t}_a 와 기울기 벡터 \mathbf{w}_a 의 크기를 그에 따라 보정한다.

$$d \leftarrow \sqrt{\mathbf{p}_a^\top \mathbf{p}_a}, \mathbf{t}_a \leftarrow \mathbf{t}_a d, \mathbf{w}_a \leftarrow \mathbf{w}_a d, \mathbf{p}_a \leftarrow \frac{1}{d} \mathbf{p}_a$$

- [단계 11] 잠재변수 벡터 \mathbf{u}_a 와 \mathbf{t}_a 간의 내부관계 계수 b_a 를 산출한다.

$$b_a \leftarrow \mathbf{u}_a^\top \mathbf{t}_a / \mathbf{t}_a^\top \mathbf{t}_a$$

- [단계 12] 독립변수행렬 \mathbf{X}_a 와 종속변수행렬 \mathbf{Y}_a 로부터 새로 얻어진 잠재변수 벡터 \mathbf{t}_a 가 설명하는 부분을 제거하고 나머지 변동만을 담은 독립변수행렬 \mathbf{X}_{a+1} 과 종속변수행렬 \mathbf{Y}_{a+1} 을 구한다.

$$\mathbf{X}_{a+1} \leftarrow \mathbf{X}_a - \mathbf{t}_a \mathbf{p}_a^\top, \mathbf{Y}_{a+1} \leftarrow \mathbf{Y}_a - b_a \mathbf{t}_a \mathbf{q}_a^\top$$

- [단계 13] $a \leftarrow a + 1$ 로 업데이트하고, [단계 1]로 돌아간다. [단계 1] - [단계 13]의 과정을 A 개의 잠재변수를 얻을 때까지 반복한다.

위 알고리즘을 아래 `nipals_plsr2`이라는 함수로 구현해보자. 이 때, 함수의 입력변수는 아래와 같다.

- X: 평균조정된 $(n \times k)$ 독립변수행렬
- Y: 평균조정된 $(n \times m)$ 종속변수행렬
- A: 잠재변수 개수

```

nipals_plsr2 <- function(X, Y, A = NULL) {
  if (is.vector(Y)) {
    Y <- as.matrix(Y, ncol = 1L)
  }

  if (nrow(X) != nrow(Y)) stop("X and Y must have the same numbers of observations.")

  if (is_empty(A) || (A > min(dim(X)))) {
    A <- min(dim(X))
  }

  Tmat <- matrix(NA, nrow = nrow(X), ncol = A)
  Umat <- matrix(NA, nrow = nrow(X), ncol = A)
  Wmat <- matrix(NA, nrow = ncol(X), ncol = A)
  Pmat <- matrix(NA, nrow = ncol(X), ncol = A)
  Qmat <- matrix(NA, nrow = ncol(Y), ncol = A)
  Bmat <- diag(nrow = A)

  for (a in seq_len(A)) {
    # 단계 1
    j <- sample.int(ncol(Y), size = 1L)
    Umat[, a] <- Y[, j]

    while (TRUE) {
      # 단계 2
      Wmat[, a] <- coef(lm(X ~ -1 + Umat[, a]))

      # 단계 3
      Wmat[, a] <- Wmat[, a] / sqrt(sum(Wmat[, a]^2))

      # 단계 4
      Tmat[, a] <- X %*% Wmat[, a]

      # 단계 5
      Qmat[, a] <- coef(lm(Y ~ -1 + Tmat[, a]))
    }
  }
}

```

```

# 단계 6
Qmat[, a] <- Qmat[, a] / sqrt(sum(Qmat[, a]^ 2))

# 단계 7
u_new <- Y %*% Qmat[, a]

# 단계 8
if (all(near(u_new, Umat[, a]))) break

Umat[, a] <- u_new
}

# 단계 9
Pmat[, a] <- coef(lm(X ~ -1 + Tmat[, a]))

# 단계 10
p_size <- sqrt(sum(Pmat[, a]^ 2))
Tmat[, a] <- Tmat[, a] * p_size
Wmat[, a] <- Wmat[, a] * p_size
Pmat[, a] <- Pmat[, a] / p_size

# 단계 11
Bmat[a, a] <- coef(lm(Umat[, a] ~ -1 + Tmat[, a]))

# 단계 12
X <- X - Tmat[, a] %*% t(Pmat[, a])
Y <- Y - Bmat[a, a] * Tmat[, a] %*% t(Qmat[, a])
}

return(list(T = Tmat, W = Wmat, P = Pmat,
           U = Umat, Q = Qmat, B = Bmat))
}

nipals_fit2 <- nipals_plsr2(X, Y, A = 3)
nipals_fit2

```

```

## $T
##          [,1]      [,2]      [,3]
## [1,]  1.5777938  0.4117531  0.44707709
## [2,] -2.2993972  8.0194451 -0.79803681
## [3,]  0.8145895  5.1398023 -0.30016399
## [4,] 21.3867331 -3.2065797 -0.01083136
## [5,] -17.8784038 -5.7770058 -1.68928119
## [6,]  9.2701258 -3.6198261 -0.09042238

```

```

## [7,] -12.8714412 -0.9675888 2.44165865
##
## $W
## [,1]      [,2]      [,3]
## [1,] -0.1476019 -0.4020251 -0.6921766
## [2,] -0.1848598  0.4862964  0.5027723
## [3,] -0.5065102 -0.8236460  0.4768770
## [4,]  0.8312518 -0.4651155  0.2794808
##
## $P
## [,1]      [,2]      [,3]
## [1,] -0.1648502  0.002634502 -0.5484068
## [2,] -0.1588038  0.130002426  0.6241044
## [3,] -0.5486716 -0.859489687  0.4559029
## [4,]  0.8040928 -0.494337847  0.3192118
##
## $U
## [,1]      [,2]      [,3]
## [1,] 11.60599122 -9.401433 -8.559602
## [2,] -0.03696152 -3.340983 -7.252566
## [3,] -9.14720168 11.437784  9.474719
## [4,] 22.98172912 -6.021677 -4.914480
## [5,] -7.12619591 -9.125035 -6.794076
## [6,]  0.47754875  7.914057  9.259791
## [7,] -18.75490997  8.537286  8.786214
##
## $Q
## [,1]      [,2]      [,3]
## [1,] 0.4832295 -0.1202142  0.07948906
## [2,] -0.8754937  0.9927480  0.99683574
##
## $B
## [,1]      [,2]      [,3]
## [1,] 0.8443757  0.0000000  0.0000000
## [2,] 0.0000000  0.4255916  0.0000000
## [3,] 0.0000000  0.0000000  3.206299

```

4.3.4 회귀식 변환

위 NIPALS 알고리즘 수행 결과를 원래 독립변수 \mathbf{X} 와 종속변수 \mathbf{Y} 에 대한 식으로 변환하는 방법은 아래와 같다.

잠재변수행렬 \mathbf{T} 는 하나의 종속변수일 때 살펴봤던 바와 같이 독립변수행렬 \mathbf{X} 와 가중치 행렬 \mathbf{W} , 그리고 로딩행렬 \mathbf{P} 의 연산으로 표현된다.

$$\mathbf{T} = \mathbf{XW} (\mathbf{P}^\top \mathbf{W})^{-1} \quad (4.16)$$

이를 식 (4.15)에 대입하면,

$$\mathbf{U} = \mathbf{XW} (\mathbf{P}^\top \mathbf{W})^{-1} \mathbf{B} + \mathbf{H} \quad (4.17)$$

이를 다시 식 (4.14)에 대입하면,

$$\begin{aligned} \mathbf{Y} &= \mathbf{XW} (\mathbf{P}^\top \mathbf{W})^{-1} \mathbf{BQ}^\top + \mathbf{HQ}^\top + \mathbf{F} \\ &= \mathbf{XB}_{PLS} + \mathbf{G} \end{aligned} \quad (4.18)$$

여기에서 $\mathbf{G} = \mathbf{HQ}^\top + \mathbf{F}$ 는 독립변수 \mathbf{X} 를 종속변수 \mathbf{Y} 에 회귀시킨 뒤 남은 오차항 행렬이다. 따라서, PLS 모형을 원 독립변수 행렬 \mathbf{X} 에 대한 모형으로 변환할 때의 회귀계수는 아래와 같이 정리된다.

$$\mathbf{B}_{PLS} = \mathbf{W} (\mathbf{P}^\top \mathbf{W})^{-1} \mathbf{BQ}^\top \quad (4.19)$$

```
beta_pls2 <- nipals_fit2$W %*%
  solve(t(nipals_fit2$P) %*% nipals_fit2$W) %*%
  nipals_fit2$B %*% t(nipals_fit2$Q)
beta_pls2
```

```
##          [,1]      [,2]
## [1,] -0.26509645 -2.8773570
## [2,]  0.08864959  2.7249194
## [3,] -0.14258488  0.3377420
## [4,]  0.37330470 -0.7406377
```

이는 앞 4.3.1 절에서 R 패키지 `pls`를 통해 얻어진 결과와 동일함을 확인할 수 있다.

```
all(near(beta_pls2, coef(pls_r_multi_fit)[, , 1]))
```

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

4.3.5 제곱합 분해

\mathbf{Y} 의 전체제곱합은

$$SST = SSR + SSE$$

로 분해되며, 여기서 SSR 은 다음과 같이 산출된다.

$$\begin{aligned} SSR &= \sum_{a=1}^A SSR_a \\ &= \sum_{a=1}^A SS(b_a \mathbf{t}_a \mathbf{q}_a^\top) \\ &= \sum_{a=1}^A b_a^2 SS(\mathbf{t}_a) \end{aligned} \tag{4.20}$$

이 때, SSR_a 는 잠재변수 \mathbf{t}_a 에 의해 설명되는 \mathbf{Y} 의 변동을 나타낸다.

```
SSR_a <- diag(
  t(nipals_fit2$T %*% nipals_fit2$B) %*%
  (nipals_fit2$T %*% nipals_fit2$B)
)
SSR_a
```

```
## [1] 739.40663 26.91455 100.23860
```

SSR_a 를 전체제곱합 SST 로 나누면 각 잠재변수가 \mathbf{Y} 의 변동에 기여하는 비율을 산출할 수 있다.

```
SST <- sum(Y ^ 2)
SSR_a / SST
```

```
## [1] 0.58621653 0.02133840 0.07947119
```

또한, 잠재변수 \mathbf{t}_a 가 설명하는 종속변수행렬 \mathbf{Y} 의 j 번째 열의 변동을 SSR_{aj} 라 할 때, 이는 다음과 같이 산출된다.

$$SSR_{aj} = q_{ja}^2 SSR_a \tag{4.21}$$

```
SSR_aj <- diag(SSR_a) %*% t(nipals_fit2$Q ^ 2)
SSR_aj
```

```
##          [,1]      [,2]
## [1,] 172.6593685 566.74726
## [2,]  0.3889542 26.52560
## [3,]  0.6333587 99.60524
```

이렇게 산출된 SSR_{aj} 를 j 번째 종속변수 y_j 의 제곱합 $SS(y_j)$ 로 나누면, y_j 에 대한 t_a 의 기여도를 얻을 수 있다.

```
SS_j <- colSums(Y ^ 2)
SSR_aj %*% diag(1 / SS_j)
```

```
##          [,1]      [,2]
## [1,] 0.817051715 0.53975930
## [2,] 0.001840593 0.02526248
## [3,] 0.002997154 0.09486213
```

위의 결과에서 y_1 의 변동은 잠재변수 t_1 으로 대부분 설명되는 반면, y_2 의 변동은 잠재변수 t_2 및 t_3 에 의해서도 설명됨을 볼 수 있다.

Part II

2부 - 분류분석

Chapter 5

분류분석 개요

분류분석(classification analysis)은 다수의 속성(attribute) 또는 변수를 갖는 객체(object)를 사전에 정해진 그룹 또는 범주(class, category) 중의 하나로 분류하는 것이다. 예를 들어, 기업의 3개의 재무제표를 기준으로 우량 또는 불량으로 분류하는 것은 범주수가 2이고 변수수가 3인 분류분석 문제가 될 것이다. 이를 위해서는 이미 범주(우량 또는 불량)가 알려진 여러 기업에 대하여 3개의 재무제표 데이터를 수집한 후 효율적인 분류규칙(classification rule)을 만들어야 할 것이다. 여기서 효율적이라 함은 기존 객체를 잘 분류할 뿐만 아니라 새로운 객체 역시 잘 분류함을 의미한다. 분류규칙을 만들기 위해서는 기존의 범주가 알려진 객체 데이터를 수집하여야 하며, 이를 학습표본(learning sample)이라 한다.

5.1 필요 R 패키지 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
stats	3.6.0
class	7.3-15

5.2 분류문제 및 분류기법

분류문제를 설명하기 위하여 N 개의 객체로 이루어진 학습데이터 $\{(\mathbf{x}_i, y_i)\}_{i=1,\dots,N}$ 를 아래와 같이 정의하자.

- \mathbf{x}_i : p 개의 독립변수로 이루어진 i 번째 객체의 변수벡터 ($\mathbf{x}_i = [x_{i1} \ x_{i2} \ \cdots \ x_{ip}]^\top$)
- J : 총 범주 수
- y_i : i 번째 객체의 범주 변수; $y_i \in \{1, 2, \dots, J\}$

이 때 학습표본을 다음과 같이 나타낼 수 있다.

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$$

분류문제는 새로운 객체를 범주 중의 하나로 분류하기 위하여 학습표본을 바탕으로 분류 규칙을 만드는 것이다. 이 때 분류규칙은 객체의 변수벡터의 함수로 도출되므로 이를 $r(\mathbf{x})$ 로 나타낸다. 이 때, $r(\mathbf{x})$ 는 $1, \dots, J$ 중 하나의 값을 가지며, 이를 분류기(classifier)라 부르기도 한다. 분류규칙의 성능을 관찰하기 위하여 우선 학습표본에 적용하여 실제범주와 추정된 범주를 비교한다. 즉, $r(\mathbf{x}_i)$ 와 y_i 를 비교하여 오분류율 등을 분석한다. 다시 말하면, $r(\mathbf{x}_i) = y_i$ 이면 올바르게 분류된 것이나, 그렇지 않으면 잘못 분류된 것이다. 학습표본에 있는 전체 객체는 서로 배타적인 J 개의 집합으로 나누어진다. 분류규칙의 성능평가에 대한 보다 자세한 설명은 이후 10장에서 하기로 한다.

분류를 위한 방법론은 무수하게 많은데, 크게 아래와 같이 대별된다.

1. 통계적 방법: 로지스틱 회귀분석, 반별분석 등 다변량 통계이론에 바탕을 둔 방법
2. 트리기반 기법: CART, C4.5, CHAID 등 트리 형태의 분지방법을 이용하는 기법
3. 비선형 최적화 기법: 서포트 벡터 머신(support vector machine; SVM) 등
4. 기계학습 기법: 신경망(neural network) 등의 블랙박스 형태의 기법

6장에서는 로지스틱 회귀분석을, 7장에서는 판별분석에 의한 분류분석을, 8장에서는 트리 기반 기법을 다루며, 9장에서는 서포트 벡터 머신을 다루고자 한다.

5.3 기본적인 분류기법

본 절에서는 위에서 언급하지 않은 기본적인 몇 가지 분류기법에 대하여 설명하고자 한다.

5.3.1 인접객체법

인접객체법(nearest neighbor classification)은 학습 데이터를 활용하지만 규칙을 도출하는 기법은 아니다. 분류하고자 하는 새로운 객체에 대하여 학습 데이터에 있는 가장 가까운 몇 개의 객체들을 찾은 후 이를 인접객체들의 다수 범주로 분류하는 기법이다. k 개의 인접객체를 고려할 때, k -인접객체법(k -nearest neighbor method)이라 한다. 가까운 정도의 척도는 유사성 척도 또는 유clidean 거리 등의 비유사성 척도가 사용되는데, 이들에 대한 자세한 설명은 11장에서 이루어진다.

5.3.1.1 기본 R 스트립트

다음과 같은 7개의 객체에 대한 학습표본이 있다.

Table 5.1: 인접객체법 학습표본

객체번호	\$x_1\$	\$x_2\$	범주
1	5	7	1
2	4	3	2
3	7	8	2
4	8	6	2
5	3	6	1
6	2	5	1
7	9	6	2

```

train_df <- tribble(
  ~id, ~x1, ~x2, ~y,
  1, 5, 7, 1,
  2, 4, 3, 2,
  3, 7, 8, 2,
  4, 8, 6, 2,
  5, 3, 6, 1,
  6, 2, 5, 1,
  7, 9, 6, 2
) %>%
  mutate(y = factor(y, levels = c(1, 2)))

knitr::kable(
  train_df, booktabs = TRUE,
  align = c('r', 'r', 'r', 'r'),
  col.names = c('객체번호', '$x_1$', '$x_2$', '범주'),
  caption = '인접객체법 학습표본'
)

```

class 패키지의 knn.cv 함수는 학습표본의 각각의 객체에 대해 그 객체를 제외한 나머지 학습표본 중 객체에서 가장 가까운(유클리드 거리 기반) k 개의 객체의 범주값을 이용하여 대상 학습표본의 범주값을 추정하는 leave-one-out cross validation을 수행한다. 아래 스크립트는 Table 5.1의 학습표본 데이터에 대해 3-인접객체 leave-one-out cross validation 결과 추정된 범주값을 산출한다.

```

y_hat <- class::knn.cv(
  train = train_df[, c("x1", "x2")],
  cl = train_df$y,
  k = 3
)

train_df %>%

```

Table 5.2: 인접객체법 추정범주 - 학습데이터

객체번호	\$x_1\$	\$x_2\$	실제범주	추정범주
1	5	7	1	2
2	4	3	2	1
3	7	8	2	2
4	8	6	2	2
5	3	6	1	1
6	2	5	1	1
7	9	6	2	2

```
mutate(y_hat = y_hat) %>%
knitr::kable(
  booktabs = TRUE,
  align = c('r', 'r', 'r', 'r', 'r'),
  col.names = c('객체번호', '$x_1$', '$x_2$', '실제범주', '추정범주'),
  caption = '인접객체법 추정범주 - 학습데이터'
)
```

`class` 패키지의 `knn` 함수는 새로운 객체에 대해 인접한 학습데이터를 이용하여 범주를 추정하는 함수이다. 아래 스크립트는 두 개의 새로운 객체 $(6, 7)^T$ 과 $(4, 2)^T$ 에 대해 3-인근객체법으로 추정범주를 구하는 스크립트이다.

```
test_df <- tribble(
  ~id, ~x1, ~x2,
  8, 6, 7,
  9, 4, 2
)

y_hat <- class::knn(
  train = train_df %>% select(x1, x2),
  test = test_df %>% select(x1, x2),
  cl = train_df$y,
  k = 3
)

test_df %>%
  mutate(y_hat = y_hat) %>%
knitr::kable(
  booktabs = TRUE,
  align = c('r', 'r', 'r', 'r'),
  col.names = c('객체번호', '$x_1$', '$x_2$', '추정범주'),
  caption = '인접객체법 추정범주 - 새로운 객체'
```

Table 5.3: 인접객체법 추정범주 - 새로운 객체

객체번호	\$x_1\$	\$x_2\$	추정범주
8	6	7	2
9	4	2	1

)

5.3.1.2 인접객체법 알고리즘

k -인접객체법의 알고리즘은 다음과 같다.

- [단계 1] k 값을 정한다.
- [단계 2] 분류하고자 하는 새로운 객체 \mathbf{z} 에 대하여
 - 2-1. 학습표본에 있는 각 객체 \mathbf{x}_i 와의 거리 $d(\mathbf{z}, \mathbf{x}_i)$ 를 산출한다.
 - 2-2. 위의 거리가 짧은 순으로 k 개의 객체를 선정한다.
 - 2-3. k 개의 인근객체가 취하는 범주 중 최빈값을 새로운 객체 \mathbf{z} 의 범주로 정한다.

위 알고리즘을 학습표본 Table 5.1와 두 새로운 객체 $(6, 7)^\top$ 및 $(4, 2)^\top$ 에 적용해보자.

[단계 1] 우선 각 학습표본 객체에 대해 k 값을 변화시키며 인접객체법으로 분류해보자. 이 때, 각 객체 스스로는 인접객체에 포함되지 않는다.

우선 아래 스크립트는 각 학습 객체간 유클리드 거리를 구한다.

```
train_pairwise_dist <- dist(train_df[, c("x1", "x2")], upper = TRUE) %>%
  broom::tidy()

train_pairwise_dist

## # A tibble: 42 x 3
##   item1 item2 distance
##   <int> <int>     <dbl>
## 1     2     1     4.12
## 2     3     1     2.24
## 3     4     1     3.16
## 4     5     1     2.24
## 5     6     1     3.61
## 6     7     1     4.12
## 7     8     1     4.12
## 8     3     2     5.83
```

```
## 9      4      2      5
## 10     5      2     3.16
## # ... with 32 more rows
```

각 객체별로 가장 인접한 객체 순으로 순서(rank)를 구한다.

```
train_nn_rank <- train_pairwise_dist %>%
  group_by(item1) %>%
  mutate(nn_rank = rank(distance, ties.method = "random")) %>%
  ungroup() %>%
  arrange(item1, nn_rank)

train_nn_rank
```

```
## # A tibble: 42 x 4
##   item1 item2 distance nn_rank
##   <int> <int>    <dbl>   <int>
## 1     1     5     2.24     1
## 2     1     3     2.24     2
## 3     1     4     3.16     3
## 4     1     6     3.61     4
## 5     1     2     4.12     5
## 6     1     7     4.12     6
## 7     2     6     2.83     1
## 8     2     5     3.16     2
## 9     2     1     4.12     3
## 10    2     4     5         4
## # ... with 32 more rows
```

이후 각 k 값에 대하여 각 객체 대해 k -인접객체법에 대한 추정범주를 구해보자.

```
loo_cv <- bind_cols(
  train_nn_rank %>% select(item1, nn_rank),
  map2_dfr(
    train_nn_rank$item1,
    train_nn_rank$nn_rank,
    function(.x, .y, df, y) {
      df %>%
        filter(
          item1 == .x,
          nn_rank <= .y
        ) %>%
        mutate(y = y[item2]) %>%
        count(y) %>%
```

```

        slice(which.max(n))
    },
    df = train_nn_rank,
    y = train_df$y
)
) %>%
  rename(k = nn_rank, y_hat = y) %>%
  mutate(y = train_df$y[item1])

loo_cv

```

A tibble: 42 x 5
item1 k y_hat n y
<int> <int> <fct> <int> <fct>
1 1 1 1 1 1 1
2 1 2 2 1 1 1
3 1 3 3 2 2 1
4 1 4 4 1 2 1
5 1 5 5 2 3 1
6 1 6 6 2 4 1
7 2 1 1 1 1 2
8 2 2 2 1 2 2
9 2 3 3 1 3 2
10 2 4 4 1 3 2
... with 32 more rows

학습객체들의 k -인접객체법 추정범주와 실제범주가 같은 비율을 정확도라 하여, 각 k 값에 대해 정확도를 계산해보자.

```

loo_cv %>%
  mutate(is_correct = (y == y_hat)) %>%
  group_by(k) %>%
  summarize(accuracy = mean(is_correct)) %>%
  arrange(desc(accuracy))

```

A tibble: 6 x 2
k accuracy
<int> <dbl>
1 1 0.714
2 2 0.714
3 3 0.714
4 4 0.714
5 5 0.286
6 6 0

위의 결과에 기반하여, 정확도가 가장 높은 경우의 k 들 중 가장 큰 값인 $k = 3$ 을 최적 k 값으로 선정하자.

[단계 2] 두 새로운 객체에 대한 3-인접객체법 추정범주를 구해보자.

우선 새로운 객체들과 기존 학습표본 객체들간의 거리를 구해보자.

```
test_df <- tribble(
  ~id, ~x1, ~x2,
  8, 6, 7,
  9, 4, 2
)

test_train_dist <- flexclust::dist2(
  test_df %>% select(x1, x2),
  train_df %>% select(x1, x2)
) %>%
  as_tibble() %>%
  `names<-`(seq_len(nrow(train_df))) %>%
  mutate(item1 = seq_len(nrow(test_df))) %>%
  gather(key = "item2", value = "distance", -item1) %>%
  mutate(item2 = as.numeric(item2))

test_train_dist

## # A tibble: 14 x 3
##   item1 item2 distance
##   <int> <dbl>     <dbl>
## 1     1     1       1
## 2     2     2       5.10
## 3     3     1       4.47
## 4     4     2       1
## 5     5     1       1.41
## 6     6     2       6.71
## 7     7     1       2.24
## 8     8     2       5.66
## 9     9     1       3.16
## 10    10    2       4.12
## 11    11    1       4.47
## 12    12    2       3.61
## 13    13    1       3.16
## 14    14    2       6.40
```

각 새로운 객체에 대하여 가장 인접한 3개의 학습표본만 남긴다.

```

test_nn <- test_train_dist %>%
  group_by(item1) %>%
  arrange(distance) %>%
  mutate(nn_rank = row_number()) %>%
  filter(nn_rank <= 3) %>%
  ungroup()

test_nn

```

A tibble: 6 x 4
item1 item2 distance nn_rank
<int> <dbl> <dbl> <int>
1 1 1 1 1
2 2 2 1 1
3 1 3 1.41 2
4 1 4 2.24 3
5 2 6 3.61 2
6 2 5 4.12 3

해당 인접 학습표본들의 범주값을 관측하여, 가장 자주 발견되는 범주값을 새로운 객체의 범주값으로 추정한다.

```

test_yhat <- test_nn %>%
  mutate(
    id = test_df$id[item1],
    y = train_df$y[item2]
  ) %>%
  group_by(id, y) %>%
  summarize(n = n()) %>%
  arrange(desc(n)) %>%
  slice(1) %>%
  ungroup() %>%
  rename(y_hat = y)

test_yhat

```

A tibble: 2 x 3
id y_hat n
<dbl> <fct> <int>
1 8 2 2
2 9 1 2

위 결과, 객체 $(6, 7)^T$ 는 범주 2로, 객체 $(4, 2)^T$ 는 범주 1로 분류된다.

5.3.2 나이브 베이지안 분류법

나이브 베이지안(Naive Bayesian) 분류법이란 속성변수들과 범주변수가 확률분포를 따른다고 간주하여 베이즈 정리와 조건부 독립성을 활용한 분류기법이다. 속성변수들이 범주형일 때 주로 사용되나, 연속형인 경우에도 확률분포의 형태를 가정하여 사용할 수 있다. 본 장에서는 범주형 변수인 경우를 설명한다.

5.3.2.1 기본 R 스크립트

아래와 같은 9명의 고객에 대한 학습표본이 있다.

```
train_df <- tribble(
  ~id, ~x1, ~x2, ~y,
  1, "남", "20대", 1,
  2, "남", "20대", 2,
  3, "남", "30대", 1,
  4, "남", "40대", 1,
  5, "여", "10대", 1,
  6, "여", "20대", 2,
  7, "여", "20대", 1,
  8, "여", "30대", 2,
  9, "여", "40대", 2
) %>%
  mutate(y = factor(y, levels = c(1, 2)))

knitr::kable(
  train_df, booktabs = TRUE,
  align = c('r', 'r', 'r', 'r'),
  col.names = c('고객번호', '성별 ($x_1$)', '나이 ($x_2$)', '범주 ($y$)'),
  caption = '나이브 베이지안 분류법 학습표본'
)
```

e1071 패키지의 `naiveBayes` 함수를 이용하면, 객체가 각 범주에 속할 조건부 확률분포 모델을 추정할 수 있다.

```
nb_fit <- e1071::naiveBayes(formula = y ~ x1 + x2, data = train_df)

print(nb_fit)

## 
## Naive Bayes Classifier for Discrete Predictors
## 
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
```

Table 5.4: 나이브 베이지안 분류법 학습표본

고객번호	성별 (\$x_1\$)	나이 (\$x_2\$)	범주 (\$y\$)
1	남	20대	1
2	남	20대	2
3	남	30대	1
4	남	40대	1
5	여	10대	1
6	여	20대	2
7	여	20대	1
8	여	30대	2
9	여	40대	2

```
##  
## A-priori probabilities:  
## Y  
##      1      2  
## 0.5555556 0.4444444  
##  
## Conditional probabilities:  
## x1  
## Y    남   여  
## 1 0.60 0.40  
## 2 0.25 0.75  
##  
## x2  
## Y    10대 20대 30대 40대  
## 1 0.20 0.40 0.20 0.20  
## 2 0.00 0.50 0.25 0.25
```

추정된 모델을 학습표본에 적용하여 범주를 추정해보자.

```
# 범주 추정값  
y_hat <- predict(nb_fit, train_df)  
  
# 사후확률 추정값  
nb_posterior <- predict(nb_fit, train_df, type = "raw") %>%  
  as_tibble() %>%  
  `colnames<-`(str_c("p", levels(train_df$y)))  
  
train_df %>%  
  mutate(y_hat = y_hat) %>%  
  bind_cols(nb_posterior) %>%  
  knitr::kable()
```

Table 5.5: 나이브 베이지안 분류법에 의한 추정 범주

고객번호	성별 (\$x_1\$)	나이 (\$x_2\$)	실제범주 (\$y\$)	추정범주 (\$\hat{y}\$)	사후확률 (\$y = 1\$)
1	남	20대	1	1	0.7058824
2	남	20대	2	1	0.7058824
3	남	30대	1	1	0.7058824
4	남	40대	1	1	0.7058824
5	여	10대	1	1	0.9925558
6	여	20대	2	2	0.3478261
7	여	20대	1	2	0.3478261
8	여	30대	2	2	0.3478261
9	여	40대	2	2	0.3478261

```
booktabs = TRUE,
align = c('r', 'r', 'r', 'r', 'r'),
col.names = c('고객번호', '성별 ($x_1$)', '나이 ($x_2$)',
              '실제범주 ($y$)', '추정범주 ($\hat{y}$)'),
str_c('사후확률 ($y = ', levels(train_df$y), ')')),
caption = '나이브 베이지안 분류법에 의한 추정 범주'
)
```

또한, 학습표본에 포함되지 않은 10대 남자인 새로운 고객에 대한 범주가 아래와 같이 추정된다.

```
predict(nb_fit, tibble(x1 = "남", x2 = "10대"))
```

```
## [1] 1
## Levels: 1 2
```

5.3.2.2 알고리즘

어떤 객체 \mathbf{x} 에 대해 범주가 y 일 조건부 확률분포는 베이즈 정리에 의하여 다음과 같이 표현된다.

$$P(y | \mathbf{x}) \propto P(y)P(\mathbf{x} | y), y = 1, \dots, J \quad (5.1)$$

여기서 $P(y)$ 는 임의의 객체가 범주 y 에 속할 사전확률을 의미하며, $P(\mathbf{x} | y)$ 는 객체 속 성변수 \mathbf{x} 의 관측값에 따른 범주 y 의 사후확률을 나타낸다. 그리고 $P(\mathbf{x} | y)$ 는 범주 y 에 속한 객체들의 속성변수 분포를 나타낸다.

나이브 베이지안 분류법에서는 속성변수들의 조건부 결합확률분포 $P(\mathbf{x} | y)$ 에 대한 조건부 독립성을 가정하여, p 개의 변수로 이루어진 객체 속성변수 벡터 $\mathbf{x} = (x_1, x_2, \dots, x_p)$ 에 대하여 다음이 성립한다고 가정한다.

$$P(x_a | x_{a+1}, x_{a+2}, \dots, x_p, y) = P(x_a | y)$$

이 때, 식 (5.1)는 아래와 같이 표현될 수 있다.

$$P(y | \mathbf{x}) \propto P(y) \prod_{a=1}^p P(x_a | y), \quad y = 1, \dots, J \quad (5.2)$$

우선, 학습표본 5.4을 이용하여 범주의 사전확률 $P(y)$ 를 추정해보자.

```
prior_prob <- train_df %>%
  group_by(y) %>%
  summarize(n = n()) %>%
  mutate(prior = n / sum(n)) %>%
  select(-n)
```

```
prior_prob
```

```
## # A tibble: 2 x 2
##   y     prior
##   <fct> <dbl>
## 1 1     0.556
## 2 2     0.444
```

또한, 학습표본 5.4에 대해 각 변수의 조건부 확률 $P(x_a | y)$ 를 추정해보자.

```
condition_prob <- train_df %>%
  gather(key = "variable", value = "value", x1, x2) %>%
  group_by(y, variable, value) %>%
  summarize(n = n()) %>%
  mutate(cond_prob = n / sum(n)) %>%
  select(-n) %>%
  ungroup() %>%
  complete(y, nesting(variable, value), fill = list(cond_prob = 0))

condition_prob
```

```
## # A tibble: 12 x 4
##   y     variable value cond_prob
##   <fct> <chr>    <chr>    <dbl>
## 1 1     x1       남        0.6
## 2 1     x1       여        0.4
## 3 1     x2       10대     0.2
```

```

##  4 1      x2      20대      0.4
##  5 1      x2      30대      0.2
##  6 1      x2      40대      0.2
##  7 2      x1      남        0.25
##  8 2      x1      여        0.75
##  9 2      x2      10대      0
## 10 2     x2      20대      0.5
## 11 2     x2      30대      0.25
## 12 2     x2      40대      0.25

```

추정된 확률을 식 (5.2)에 적용하여, 각 학습데이터에 대한 범주의 사후확률을 구해보자.

```

posterior_prob <- train_df %>%
  select(-y) %>%
  gather(key = "variable", value = "value", x1, x2) %>%
  inner_join(condition_prob, by = c("variable", "value")) %>%
  group_by(id, y) %>%
  summarize(cond_prob = reduce(cond_prob, `*`)) %>%
  inner_join(prior_prob, by = "y") %>%
  mutate(posterior_unadjust = prior * cond_prob) %>%
  mutate(posterior = posterior_unadjust / sum(posterior_unadjust)) %>%
  select(id, y, posterior) %>%
  ungroup()

posterior_prob %>%
  spread(key = y, value = posterior)

```

```

## # A tibble: 9 x 3
##       id     `1`     `2`
##   <dbl> <dbl> <dbl>
## 1     1  0.706  0.294
## 2     2  0.706  0.294
## 3     3  0.706  0.294
## 4     4  0.706  0.294
## 5     5  1        0
## 6     6  0.348  0.652
## 7     7  0.348  0.652
## 8     8  0.348  0.652
## 9     9  0.348  0.652

```

추정범주는 사후확률이 가장 큰 범주를 선택한다.

```

posterior_prob %>%
  group_by(id) %>%

```

```
top_n(1, posterior) %>%
  slice(1)

## # A tibble: 9 x 3
## # Groups:   id [9]
##       id y     posterior
##   <dbl> <fct>    <dbl>
## 1     1 1     0.706
## 2     2 1     0.706
## 3     3 1     0.706
## 4     4 1     0.706
## 5     5 1     1
## 6     6 2     0.652
## 7     7 2     0.652
## 8     8 2     0.652
## 9     9 2     0.652
```

5.3.2.3 R 패키지 내 나이브 베이지안 분류법

위 5.3.2.1절에서 살펴본 바와 같이 e1071 패키지 내의 `naiveBayes` 함수를 이용하여 분류 모델을 추정할 수 있다.

```
nb_fit <- e1071::naiveBayes(formula = y ~ x1 + x2, data = train_df)
```

위 `naiveBayes` 모델 객체의 component 중 `apriori`는 객체가 각 범주에 속할 사전분포를 나타내는 `table` 형태의 객체로, 본 예에서 학습표본 중 각 범주에 속한 객체 수를 나타낸다.

```
str(nb_fit$apriori)
```

```
## 'table' int [1:2(1d)] 5 4
## - attr(*, "dimnames")=List of 1
##   ..$ Y: chr [1:2] "1" "2"
```

아래와 같이, 각 범주에 속한 객체 수를 전체 객체 수로 나눔으로써 추정된 사전분포(prior distribution)을 확인할 수 있다.

```
nb_fit$apriori %>%
  broom::tidy() %>%
  mutate(p = n / sum(n))
```

```
## # A tibble: 2 x 3
##   Y      n     p
##   <chr> <int> <dbl>
## 1 1       5 0.556
## 2 2       4 0.444
```

각 변수별 조건부 확률은 `tables`라는 리스트 객체에서 변수별로 확인할 수 있다.

```
nb_fit$tables
```

```
## $x1
##   x1
## Y   남 여
## 1 0.60 0.40
## 2 0.25 0.75
##
## $x2
##   x2
## Y   10대 20대 30대 40대
## 1 0.20 0.40 0.20 0.20
## 2 0.00 0.50 0.25 0.25
```

`predict` 함수를 이용하여 사후확률을 구할 때, `threshold` 파라미터값을 이용하여 최소 사후확률값을 지정할 수 있다. 기본값은 0.001로, 추정 사후확률값이 최소 0.1%보다 커야한다는 것을 의미한다.

```
predict(nb_fit, newdata = train_df[5, ], type = "raw")
```

```
##           1           2
## [1,] 0.9925558 0.007444169
```

해당 파라미터값을 0.01으로 지정할 경우, 위에서 범주 2에 속할 사후확률이 보다 크게 얻어짐을 확인할 수 있다.

```
predict(nb_fit, newdata = train_df[5, ], type = "raw", threshold = 0.01)
```

```
##           1           2
## [1,] 0.9302326 0.06976744
```

Chapter 6

로지스틱 회귀분석

로지스틱 회귀분석(logistic regression)은 종속변수가 통상 2개의 범주(있음/없음, 불량/양호, 합격/불합격 등)를 다루는 모형을 지칭하나, 3개 이상의 범주를 다루기도 한다. 후자의 경우는 다시 서열형(ordinal) 데이터와 명목형(nominal) 데이터인 경우에 따라서 다른 모형이 사용된다.

6.1 필요 R 패키지 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
stats	3.6.0
nnet	7.3-12
MASS	7.3-51.4
VGAM	1.1-1

6.2 이분 로지스틱 회귀모형

6.2.1 기본 R 스크립트

```
train_df <- tribble(
  ~id, ~x1, ~x2, ~x3, ~y,
  1, 0, 8, 2, "우수",
  2, 1, 7, 1, "우수",
  3, 0, 9, 0, "우수",
  4, 1, 6, 4, "우수",
```

Table 6.1: 우수/보통 학생에 대한 설문조사 결과

객체번호	아침식사여부(\$x_1\$)	수면시간(\$x_2\$)	서클활동시간(\$x_3\$)	범주(y)
1	0	8	2	우수
2	1	7	1	우수
3	0	9	0	우수
4	1	6	4	우수
5	1	8	2	우수
6	0	7	3	우수
7	0	7	0	보통
8	1	6	1	보통
9	0	7	2	보통
10	0	8	1	보통
11	0	5	2	보통
12	1	8	0	보통
13	0	6	3	보통
14	1	7	2	보통
15	0	6	1	보통

```

5, 1, 8, 2, "우수",
6, 0, 7, 3, "우수",
7, 0, 7, 0, "보통",
8, 1, 6, 1, "보통",
9, 0, 7, 2, "보통",
10, 0, 8, 1, "보통",
11, 0, 5, 2, "보통",
12, 1, 8, 0, "보통",
13, 0, 6, 3, "보통",
14, 1, 7, 2, "보통",
15, 0, 6, 1, "보통"
) %>%
  mutate(y = factor(y, levels = c("보통", "우수")))

knitr::kable(train_df, booktabs = TRUE,
             align = c('r', 'r', 'r', 'r', 'r'),
             col.names = c('객체번호', '아침식사여부($x_1$)', '수면시간($x_2$)', '서클활동시간($x_3$)', '범주(y)'),
             caption = '우수/보통 학생에 대한 설문조사 결과')

```

Table 6.1와 같이 세 개의 독립변수 x_1, x_2, x_3 와 이분형 종속변수 y 의 관측값(보통 = 0, 우수 = 1)으로 이루어진 15개의 학습표본을 `train_df`라는 data frame에 저장한다.

아래와 같이 `glm` 함수를 이용하여 로지스틱 회귀모형을 간편하게 추정할 수 있다.

Table 6.2: 위 우수/보통 학생 설문조사 데이터에 대한 Logistic Regression 결과

term	estimate	std.error	statistic	p.value
(Intercept)	-30.510836	18.018256	-1.693329	0.0903929
x1	2.031278	1.983692	1.023989	0.3058406
x2	3.470671	2.074978	1.672631	0.0944000
x3	2.414387	1.396372	1.729043	0.0838015

```
glm_fit <- glm(y ~ x1 + x2 + x3, family = binomial(link = "logit"), data = train_df)

knitr::kable(
  glm_fit %>% broom::tidy(),
  booktabs = TRUE,
  caption = "위 우수/보통 학생 설문조사 데이터에 대한 Logistic Regression 결과"
#  caption = "Table \@ref(tab:binary-logistic-reg-train-data)에 대한 Logistic Regression 결과"
)
```

Table 6.2은 추정된 회귀계수 추정치 `estmate`과 그 표준오차 `std.error`, 표준화(standardized) 된 회귀계수값 `statistic` (= `estmate / std.error`), 그리고 귀무가설 $H_0: \text{statistic} = 0$ 에 대한 유의확률 `p.value`를 보여준다.

6.2.2 회귀모형

이분 로지스틱 회귀모형은 종속변수가 2가지 범주를 취하는 경우에 사용된다.

N 개의 객체로 이루어진 학습데이터 $\{(x_i, y_i)\}_{i=1, \dots, N}$ 를 아래와 같이 정의하자.

- $\mathbf{x}_i \in \mathbb{R}^p$: p 개의 독립변수로 이루어진 벡터 ($\mathbf{x}_i = [x_{i1} \ x_{i2} \ \cdots \ x_{ip}]^\top$)
- y_i : 0 혹은 1의 값을 갖는 이분형 지시변수 (indicator variable)

\mathbf{x}_i 관측값을 이용하여 y_i 의 기대값 P_i 을 추정하는 모형을 아래와 같이 로지스틱 함수로 정의하자.

$$P_i = P(y_i = 1 | \mathbf{x}_i) \quad (6.1)$$

$$= E[y_i | \mathbf{x}_i] \quad (6.2)$$

$$= \frac{\exp(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)}{1 + \exp(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)} \quad (6.3)$$

여기에서 $\boldsymbol{\beta} \in \mathbb{R}^p$ 는 \mathbf{x}_i 와 동일한 차원의 벡터이다 ($\boldsymbol{\beta} = [\beta_1 \ \beta_2 \ \cdots \ \beta_p]^\top$).

식 (6.3)는 모든 \mathbf{x}_i 값에 대해 0에서 1 사이의 값을 갖게 되므로 각 범주에 속할 확률을 추정하는 데 적합한 반면, 변수 \mathbf{x} 및 계수들에 대해 선형이 아니므로 추정이 어렵다. 그러나 아래와 같이 로짓(logit) 변환을 통해 선형회귀식으로 변환할 수 있다.

$$\text{logit}(P_i) = \ln \left[\frac{P_i}{1 - P_i} \right] \quad (6.4)$$

$$= \ln(\exp(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)) \quad (6.5)$$

$$= \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i \quad (6.6)$$

식 (6.6)에서 확률 P_i 는 직접적으로 관측되는 것이 아니고 0 또는 1을 갖는 y_i 가 관측되므로, P_i 를 일종의 잠재변수(latent variable)로 해석할 수 있다.

$$y_i = \begin{cases} 1 & \text{if } \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i + \varepsilon_i > 0 \\ 0 & \text{otherwise} \end{cases} \quad (6.7)$$

식 (6.7)에서 ε_i 는 표준로지스틱분포(standard logistic distribution)을 따른다.

6.2.3 회귀계수 추정

로지스틱 모형에서 회귀계수의 추정을 위해서 주로 최우추정법(maximum likelihood estimation)이 사용된다. N 개의 객체로 이루어진 학습데이터에 대해 우도함수는 다음과 같다.

$$L = \prod_{i=1}^N P_i^{y_i} (1 - P_i)^{1-y_i}$$

그리고 우도함수에 자연로그를 취하면 아래와 같이 전개된다.

$$\log L = \sum_{i=1}^N y_i \log P_i + \sum_{i=1}^N (1 - y_i) \log(1 - P_i) \quad (6.8)$$

$$= \sum_{i=1}^N y_i \log \frac{P_i}{1 - P_i} + \sum_{i=1}^N \log(1 - P_i) \quad (6.9)$$

$$= \sum_{i=1}^N y_i (\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i) - \sum_{i=1}^N \log(1 + \exp(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)) \quad (6.10)$$

$$= \sum_{i=1}^N y_i \left(\beta_0 + \sum_{j=1}^p \beta_j x_{ij} \right) - \sum_{i=1}^N \log \left(1 + \exp \left(\beta_0 + \sum_{j=1}^p \beta_j x_{ij} \right) \right) \quad (6.11)$$

식 (eq:binary-logistic-reg-loglik)을 각 회귀계수 $\beta_0, \beta_1, \dots, \beta_p$ 에 대해 편미분하여 최적해를 얻는다. 이를 위해 주로 뉴턴-랩슨 알고리즘(Newton-Raphson algorithm)이나 quasi-Newton 알고리즘이 사용되나 (전치혁, 2012), 본 장에서는 우선 안정성은 떨어지지만 보다 간편한 방법으로 경사하강법(gradient descent)을 소개한다.

6.2.3.1 경사하강법

식 (6.3)과 $P(y_i = 0 | \mathbf{x}_i) = 1 - P_i$, 그리고

$$\frac{\exp(z)}{1 + \exp(z)} = \frac{1}{1 + \exp(-z)}$$

임을 고려하면 아래와 같이 범주학률모형을 정의할 수 있다.

$$P(y = y_i | \mathbf{x}_i, \beta_0, \boldsymbol{\beta}) = \frac{1}{1 + \exp((1 - 2y_i)(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}))}$$

이에 따라 로그우도함수 (6.11)는 아래와 같이 정리된다.

$$\log \prod_{i=1}^N P(y = y_i | \mathbf{x}_i, \beta_0, \boldsymbol{\beta}) = - \sum_{i=1}^N \log \left(1 + \exp \left((1 - 2y_i)(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}) \right) \right)$$

위 로그우도함수를 최대화하는 문제는 아래 함수를 최소화하는 문제와 동일하다.

$$f(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^N \log \left(1 + \exp \left((1 - 2y_i)(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}) \right) \right) \quad (6.12)$$

경사하강법에 따라 아래의 과정을 통해 회귀계수를 추정할 수 있다.

1. 임의의 값으로 $\beta_0, \beta_1, \dots, \beta_j$ 의 초기 추정값을 설정한다.
2. 식 (6.12)을 각 회귀변수에 대해 편미분한 미분값을 구한다.
3. 2의 값에 학습률(step size)을 곱한 만큼 회귀계수 추정값을 이동시킨다. 방향은 미분값의 반대방향.
4. 수렴할 때까지 2-3의 과정을 반복한다.

여기에서 식 (6.12)의 각 회귀변수에 대한 편미분식은 아래와 같다.

$$\begin{aligned}\frac{\partial f}{\partial \beta_0} &= \sum_{i=1}^N (1 - 2y_i) \frac{\exp((1 - 2y_i)(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}))}{1 + \exp((1 - 2y_i)(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}))} \\ &= \sum_{i=1}^N \frac{1 - 2y_i}{1 + \exp((2y_i - 1)(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}))}\end{aligned}$$

$$\begin{aligned}\frac{\partial f}{\partial \beta_j} &= \sum_{i=1}^N (1 - 2y_i) x_{ij} \frac{\exp((1 - 2y_i)(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}))}{1 + \exp((1 - 2y_i)(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}))} \\ &= \sum_{i=1}^N \frac{(1 - 2y_i) x_{ij}}{1 + \exp((2y_i - 1)(\beta_0 + \sum_{j=1}^p \beta_j x_{ij}))}\end{aligned}$$

따라서, 회귀계수 추정값을 이동시키는 함수 `update_beta`를 아래와 같이 구현할 수 있다.

```
update_beta <- function(x, y, beta0 = 0, beta = rep(0, dim(x)[2]), alpha = 0.01) {
  # 변미분식의 분모
  denominator <- 1 + exp((2 * y - 1) * (beta0 + (x %*% beta)))

  # intercept 이동량 계산
  beta0_numerator <- 1 - 2 * y
  beta0_update = sum(beta0_numerator / denominator)

  # intercept 외 회귀계수 이동량 계산
  beta_numerator <- sweep(x, MARGIN = 1, STATS = 1 - 2 * y, FUN = "*")
  beta_update = apply(beta_numerator, MARGIN = 2,
    function(x) sum(x / denominator))

  # 회귀계수 이동
  beta0 <- beta0 - alpha * beta0_update
  beta <- beta - alpha * beta_update

  return(list(beta0 = beta0, beta = beta))
}
```

위의 함수를 이용하여 아래 `estimate_beta`처럼 수렴할 때까지 회귀계수 추정값을 계속 이동시킨다. 본 경사하강법은 학습률 파라미터 `alpha`값에 따라 민감한 단점이 있으며, 특히 `alpha`값을 크게 설정할 경우에는 추정값이 수렴하지 않고 오히려 실제값에서 계속 멀어지는 현상이 발생하기도 한다. 이러한 단점을 보완하기 위한 여러 방법이 있으나, 본 장에서 자세한 설명은 생략하기로 한다.

```

calculate_loglik <- function(x, y,
                           beta0 = 0,
                           beta = rep(0, dim(x)[2])) {
  sum(y * (beta0 + (x %*% beta))) -
  sum(log(1 + exp(beta0 + (x %*% beta))))
}

estimate_beta <- function(x, y,
                           beta0 = 0,
                           beta = rep(0, dim(x)[2]),
                           alpha = 0.01,
                           conv_threshold = 1e-5,
                           max_iter = 1e+5) {
  new_beta0 <- beta0
  new_beta <- beta
  conv <- FALSE

  i_iter <- 0
  while(i_iter < max_iter) {
    res <- update_beta(x, y, new_beta0, new_beta, alpha)

    if(abs(calculate_loglik(x, y, beta0, beta) -
           calculate_loglik(x, y, res$beta0, res$beta)) < conv_threshold) {
      conv <- TRUE
      break
    }

    new_beta0 <- res$beta0
    new_beta <- res$beta

    i_iter <- i_iter + 1
  }

  return(list(conv = conv, beta0 = new_beta0, beta = new_beta))
}

```

위에서 정의한 함수를 이용하여 Table 6.1의 학습표본에 대한 로지스틱 회귀모형을 추정해보자.

```

res <- estimate_beta(train_df[, c("x1", "x2", "x3")] %>% as.matrix(),
                      train_df$y %>% as.numeric() - 1,
                      alpha = 0.015,
                      conv_threshold = 1e-6)

```

```
print(res)

## $conv
## [1] FALSE
##
## $beta0
## [1] -30.39189
##
## $beta
##      x1        x2        x3
## 2.022808 3.457019 2.405969
```

위 회귀계수 추정값은 R 함수 `glm`을 이용한 추정값(Table 6.2)과 유사함을 볼 수 있다.

6.2.3.2 반복재가중최소제곱법

R의 `glm` 함수는 반복재가중최소제곱법(iteratively reweighted least squares; IRLS 혹은 IWLS)을 사용한다. 이는 선형회귀식

$$\text{logit}(y_i) = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i + \varepsilon_i$$

을 추정하는 방법인데, 여기에서 y_i 는 0 혹은 1이므로, $\text{logit}(y_i)$ 는 $-\infty$ 혹은 ∞ 가 되어 회귀식을 추정할 수 없다. 따라서, 식 (6.3)에 설명된 로지스틱 함수

$$P = \frac{\exp(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x})}{1 + \exp(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x})}$$

와 테일러 급수(Taylor series)를 이용하여 $\text{logit}(y)$ 에 대한 근사함수를 아래와 같이 얻는다.

$$\begin{aligned} g(y) &= \text{logit}(P) + (y - P) \frac{\partial \text{logit}(P)}{\partial P} \\ &= \log \frac{P}{1 - P} + (y - P) \left(\frac{1}{P} + \frac{1}{1 - P} \right) \end{aligned}$$

그리고 아래 선형회귀식을 추정한다.

$$g(y_i) = \beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i + \varepsilon_i$$

여기에서 오차항 ε_i 의 분산은 추정된 확률 P_i 에 따라 다르므로, 통상적 최소자승법(ordinary least squares; OLS) 대신 오차항의 분산이 동일해지도록 객체마다 가중치를

부여하는 가중최소자승법(weighted least squares; WLS)을 사용한다. 로지스틱 회귀 모형에서 각 객체의 가중치는

$$w_i = P_i(1 - P_i)$$

가중치와 회귀계수 추정값은 상호 영향을 미치므로, 수렴할 때까지 반복적으로 가중치와 회귀계수 추정값을 변화시키면서 최종 추정값을 찾아가는 방법이다.

우선 회귀계수 추정값이 주어졌을 때 각 객체에 대한 확률값 P_i 와 가중치 w_i 를 구하는 함수 `calculate_weight`를 아래와 같이 구현해보자.

```
calculate_weight <- function(x, beta0 = 0,
                                beta = rep(0, dim(x)[2])) {
  # 각 객체의 y값이 1일 확률
  P <- (1 + exp(-beta0 - (x %*% beta)))^( -1) %>% drop()

  # 가중치 계산
  w <- P * (1 - P)

  return(list(P = P, w = w))
}
```

그리고 확률추정값과 가중치가 주어졌을 때 회귀계수를 구하는 함수 `calculate_beta`를 아래와 같이 구현해보자. 여기서 회귀계수를 구하는 부분은 R의 선형회귀분석함수 `lm`을 사용한다.

```
calculate_beta <- function(x, y, P, w) {
  # 추정확률값이 0이나 1인 경우
  # 여전히 logit 함수가 정의되지 않으므로 회귀계수 결정에서 제외
  logit_derivative <- 1/P + 1/(1 - P)
  is_good <- !is.nan(logit_derivative)

  # 모든 객체에 대한 추정확률이 0이나 1인 경우 회귀계수 추정 불가능
  if(all(!is_good)) return(NULL)

  # 테일러 급수 계산
  g_y <- log(P[is_good]) -
    log(1 - P[is_good]) +
    (y[is_good] - P[is_good]) * logit_derivative

  # 가중치최소자승법을 이용한 추정
  df <- bind_cols(as_tibble(x) %>%
    `colnames<-`(`colnames(x)),
    tibble(g_y = g_y))
  lm(g_y ~ ., data = df, subset = is_good, weights = w)
}
```

위에서 정의한 두 함수 `calculate_weight`과 `calculate_beta`를 반복적으로 사용하여 Table 6.1의 학습표본에 대한 로지스틱 회귀모형을 추정해보자. 모든 객체의 가중치 변화량이 1/10000 보다 작을 경우 모형추정이 수렴한 것으로 간주하도록 하자.

```
X <- train_df[, c("x1", "x2", "x3")] %>% as.matrix()
y <- train_df$y %>% as.numeric() - 1

weight <- calculate_weight(X)
for(i in 1:10) {
  wls_fit <- calculate_beta(X, y, weight$P, weight$w)

  if(is.null(wls_fit)) {break}

  new_weight <- calculate_weight(x = X,
                                   beta0 = coef(wls_fit)[1],
                                   beta = coef(wls_fit)[-1])

  if(max(abs(new_weight$w - weight$w)) < 1e-4) {break}

  weight <- new_weight
}

coef(wls_fit)

## (Intercept)          x1          x2          x3
## -30.510837    2.031278   3.470671   2.414387
```

위 스크립트를 실행시킨 결과 7번째 반복수행에서 결과가 수렴하였으며, 해당 결과는 `glm` 함수를 사용하였을 때의 결과 (Table 6.2) 과 매우 근사함을 확인할 수 있다.

6.3 명목 로지스틱 회귀모형

6.3.1 기본 R 스크립트

```
train_df <- tribble(
  ~id, ~x1, ~x2, ~y,
  1, 0.09, 5.02, 1,
  2, 0.1, 5.01, 1,
  3, 0.12, 4.94, 1,
  4, 0.12, 5.12, 1,
  5, 0.12, 5.03, 1,
  6, 0.12, 4.94, 2,
```

Table 6.3: 공정변수-불량 종류 데이터

객체번호	\$x_1\$	\$x_2\$	불량범주(\$y\$)
1	0.09	5.02	1
2	0.10	5.01	1
3	0.12	4.94	1
4	0.12	5.12	1
5	0.12	5.03	1
6	0.12	4.94	2
7	0.10	5.13	2
8	0.10	4.87	1
9	0.10	5.13	2
10	0.11	4.94	3
11	0.11	4.93	3
12	0.09	5.02	3
13	0.10	5.01	3
14	0.09	4.94	3
15	0.10	5.12	2
16	0.12	4.93	2
17	0.10	5.00	1
18	0.09	5.01	3

```

7, 0.1, 5.13, 2,
8, 0.1, 4.87, 1,
9, 0.1, 5.13, 2,
10, 0.11, 4.94, 3,
11, 0.11, 4.93, 3,
12, 0.09, 5.02, 3,
13, 0.1, 5.01, 3,
14, 0.09, 4.94, 3,
15, 0.1, 5.12, 2,
16, 0.12, 4.93, 2,
17, 0.1, 5, 1,
18, 0.09, 5.01, 3
) %>%
  mutate(y = factor(y, levels = c(1, 2, 3)))

knitr::kable(train_df, booktabs = TRUE,
             align = c('r', 'r', 'r', 'r'),
             col.names = c('객체번호', '$x_1$', '$x_2$', '불량범주($y$)'),
             caption = '공정변수-불량 종류 데이터')

```

Table 6.3와 같이 두 개의 독립변수 x_1, x_2 에 따라 세 종류의 불량 ($y = 1, 2, 3$)이 발생

Table 6.4: 위 공정변수-불량종류 데이터에 대한 Logistic Regression 결과

y.level	term	estimate	std.error	statistic	p.value
2	(Intercept)	-53.924661	45.402992	-1.1876896	0.2349557
2	x1	31.545749	62.162158	0.5074751	0.6118215
2	x2	9.992311	8.479305	1.1784352	0.2386231
3	(Intercept)	64.191145	57.282533	1.1206059	0.2624556
3	x1	-101.817613	65.516143	-1.5540844	0.1201643
3	x2	-10.810865	10.915881	-0.9903795	0.3219887

함을 알았다면, 아래와 같이 `nnet` 패키지의 `multinom` 함수를 이용하여 공정변수에 따른 불량 종류를 분류하기 위한 로지스틱 회귀모형을 간편하게 추정할 수 있다.

```
multinom_fit <- nnet::multinom(
  y ~ x1 + x2,
  data = train_df,
  maxit = 1000)

## # weights:  12 (6 variable)
## initial value 19.775021
## iter  10 value 17.825831
## iter  20 value 16.489924
## iter  30 value 16.116751
## iter  40 value 16.044223
## iter  50 value 16.025259
## iter  60 value 16.024629
## iter  70 value 16.016395
## final value 16.015185
## converged

knitr::kable(
  multinom_fit %>%
    broom::tidy(exponentiate = FALSE),
  booktabs = TRUE,
  caption = '위 공정변수-불량종류 데이터에 대한 Logistic Regression 결과',
#  caption = 'Table \\\\@ref(tab:nominal-logistic-reg-train-data)에 대한 Logistic Regression 결과'
)
```

`multinom` 함수는 범주 `y`의 값 1, 2, 3중 첫번째 값인 1을 기준범주(reference category)로 사용한다.

6.3.2 기준범주 로짓모형

종속변수가 세 이상의 범주를 갖고 있으나 자연스러운 순서가 없는 경우, 기준범주 로짓모형이 널리 사용된다.

N 개의 객체로 이루어진 학습데이터 $\{(\mathbf{x}_i, y_i)\}_{i=1,\dots,N}$ 를 아래와 같이 정의하자.

- $\mathbf{x}_i \in \mathbb{R}^p$: p 개의 독립변수로 이루어진 벡터 ($\mathbf{x}_i = [x_{i1} \ x_{i2} \ \cdots \ x_{ip}]^\top$)
- J : 범주 수
- y_i : 객체 i 에 대한 종속변수값 $\in \{1, 2, \dots, J\}$

각 객체 i 가 각 범주에 해당할 확률을 π_{ij} 라 하자.

$$\pi_{ij} = P(y_i = j \mid \mathbf{x}_i), \quad j = 1, \dots, J$$

이 때, 모든 i 에 대하여

$$\sum_{j=1}^J \pi_{ij} = 1$$

이 성립한다. 여기에서 범주 1을 기준 범주(reference category 혹은 baseline category)로 간주하여 범주별로 다음과 같은 회귀모형을 정의한다 (교재 (전치혁, 2012)에는 범주 J 를 기준 범주로 간주).

$$\log \left(\frac{\pi_{ij}}{\pi_{i1}} \right) = \beta_{0,j} + \boldsymbol{\beta}_j^\top \mathbf{x}_i, \quad j = 2, \dots, J$$

이를 π_{ij} 에 대해 풀면, 아래와 같은 해가 얻어진다 (Czepiel, 2002).

$$\begin{aligned} \pi_{ij} &= \frac{\exp \left(\beta_{0,j} + \boldsymbol{\beta}_j^\top \mathbf{x}_i \right)}{1 + \sum_{j=2}^J \exp \left(\beta_{0,j} + \boldsymbol{\beta}_j^\top \mathbf{x}_i \right)}, \quad j = 2, \dots, J \\ \pi_{i1} &= \frac{1}{1 + \sum_{j=2}^J \exp \left(\beta_{0,j} + \boldsymbol{\beta}_j^\top \mathbf{x}_i \right)} \end{aligned} \tag{6.13}$$

위 모수 추정을 위해 최우추정법을 사용해보자. 우선, 종속변수를 변환한 지시변수를 아래와 같이 정의한다.

$$v_{ij} = \begin{cases} 1 & \text{if } y_i = j \\ 0 & \text{otherwise} \end{cases}$$

이를 이용해 우도 함수를

$$\begin{aligned}
L &= \prod_{i=1}^n \prod_{j=1}^J (\pi_{ij})^{v_{ij}} \\
&= \prod_{i=1}^n \pi_{i1}^{1-\sum_{j=2}^J v_{ij}} \prod_{j=2}^J (\pi_{ij})^{v_{ij}} \\
&= \prod_{i=1}^n \frac{\pi_{i1}}{\pi_{i1}^{\sum_{j=2}^J v_{ij}}} \prod_{j=2}^J (\pi_{ij})^{v_{ij}} \\
&= \prod_{i=1}^n \frac{\pi_{i1}}{\prod_{j=2}^J \pi_{i1}^{v_{ij}}} \prod_{j=2}^J (\pi_{ij})^{v_{ij}} \\
&= \prod_{i=1}^n \pi_{i1} \prod_{j=2}^J \left(\frac{\pi_{ij}}{\pi_{i1}} \right)^{v_{ij}}
\end{aligned}$$

와 같이 표현할 수 있으며, 여기에 식 (6.13)을 이용하면 아래와 같이 정리할 수 있다.

$$\begin{aligned}
L &= \prod_{i=1}^n \frac{1}{1 + \sum_{j=2}^J \exp(\beta_{0,j} + \beta_j^\top \mathbf{x}_i)} \prod_{j=2}^J \left(\exp(\beta_{0,j} + \beta_j^\top \mathbf{x}_i) \right)^{v_{ij}} \\
&= \prod_{i=1}^n \left(1 + \sum_{j=2}^J \exp(\beta_{0,j} + \beta_j^\top \mathbf{x}_i) \right)^{-1} \prod_{j=2}^J \left(\exp(\beta_{0,j} + \beta_j^\top \mathbf{x}_i) \right)^{v_{ij}}
\end{aligned}$$

이에 따라 로그 우도함수는 다음과 같이 정의된다.

$$\log L = \sum_{i=1}^n \left(-\log \left(1 + \sum_{j=2}^J \exp(\beta_{0,j} + \beta_j^\top \mathbf{x}_i) \right) + \sum_{j=2}^J v_{ij} (\beta_{0,j} + \beta_j^\top \mathbf{x}_i) \right) \quad (6.14)$$

식 (6.14)을 각 계수에 대해 미분하면 아래와 같이 정리된다.

$$\begin{aligned}
\frac{\partial \log L}{\partial \beta_{0,j}} &= \sum_{i=1}^n v_{ij} - \frac{\exp(\beta_{0,j} + \beta_j^\top \mathbf{x}_i)}{1 + \sum_{j=2}^J \exp(\beta_{0,j} + \beta_j^\top \mathbf{x}_i)} \\
\frac{\partial \log L}{\partial \beta_{k,j}} &= \sum_{i=1}^n v_{ij} x_{ik} - \frac{x_{ik} \exp(\beta_{0,j} + \beta_j^\top \mathbf{x}_i)}{1 + \sum_{j=2}^J \exp(\beta_{0,j} + \beta_j^\top \mathbf{x}_i)}, \quad k = 1, \dots, p
\end{aligned} \quad (6.15)$$

따라서, 명목 로지스틱 회귀분석은 식 (6.15)이 표현하는 $(J - 1) \times (p + 1)$ 개의 미분식을 모두 0으로 만드는 계수값을 찾는 문제가 된다. 이에 대한 closed form solution은 존재하지 않으므로, 각종 알고리즘을 이용하여 해를 찾아야 한다. Newton-Raphson method에 의해 해를 찾는 방법은 Czepiel (2002)에 보다 자세하게 설명되어 있다.

Table 6.3의 학습데이터에 대해 명목 로지스틱 회귀모형을 학습하여 범주를 추정한 결과는 아래와 같다.

```
multinom_fit <- nnet::multinom(
  y ~ x1 + x2,
  data = train_df,
  maxit = 1000)

## # weights: 12 (6 variable)
## initial value 19.775021
## iter 10 value 17.825831
## iter 20 value 16.489924
## iter 30 value 16.116751
## iter 40 value 16.044223
## iter 50 value 16.025259
## iter 60 value 16.024629
## iter 70 value 16.016395
## final value 16.015185
## converged

predict_df <- predict(multinom_fit, train_df, type = "probs") %>%
  as_data_frame() %>%
  `colnames<-`(`c("p1", "p2", "p3")`) %>%
  mutate(pred_class = predict(multinom_fit, train_df, type = "class"))

bind_cols(train_df, predict_df) %>%
  select(-x1, -x2) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r', 'r', 'r', 'r'),
    col.names = c('객체번호', '불량범주 $y_i$', '',
                 '$\\pi_{i1}$', '$\\pi_{i2}$', '$\\pi_{i3}$', '추정범주 $\\hat{y}_i$'),
    caption = '명목 로지스틱 회귀모형 범주 추정 결과',
    digits = 3
  )

```

`nnet` 패키지 외에도 `glmnet`, `mlogit`, `VGAM` 등의 R 패키지들을 사용해 명목형 로지스틱 회귀모형을 추정할 수 있다.

Table 6.5: 명목 로지스틱 회귀모형 범주 추정 결과

객체번호	불량범주 \$y_i\$	\$\pi_{i1}\$	\$\pi_{i2}\$	\$\pi_{i3}\$	추정범주 \$\hat{y}_i\$
1	1	0.283	0.112	0.604	3
2	1	0.425	0.209	0.365	1
3	1	0.589	0.271	0.141	1
4	1	0.262	0.729	0.009	2
5	1	0.450	0.509	0.041	2
6	2	0.589	0.271	0.141	1
7	2	0.349	0.570	0.082	2
8	1	0.199	0.024	0.777	3
9	2	0.349	0.570	0.082	2
10	3	0.501	0.168	0.331	1
11	3	0.490	0.149	0.361	1
12	3	0.283	0.112	0.604	3
13	3	0.425	0.209	0.365	1
14	3	0.160	0.029	0.811	3
15	2	0.365	0.540	0.095	2
16	2	0.595	0.247	0.158	1
17	1	0.416	0.186	0.398	1
18	3	0.268	0.096	0.636	3

```

VGAM::vglm(y ~ x1 + x2,
            data = train_df,
            family = VGAM::multinomial)

## 
## Call:
## VGAM::vglm(formula = y ~ x1 + x2, family = VGAM::multinomial,
##            data = train_df)
## 
## 
## Coefficients:
## (Intercept):1 (Intercept):2          x1:1          x1:2          x2:1
##      -64.56378     -118.15274     102.65063     133.29235     10.86829
##           x2:2
##      20.81307
## 
## Degrees of Freedom: 36 Total; 30 Residual
## Residual deviance: 32.03007
## Log-likelihood: -16.01503
## 
```

```
## This is a multinomial logit model with 3 levels
```

6.4 서열 로지스틱 회귀모형

본 장에서는 종속변수가 3개 이상의 범주를 가지며, 각 범주 간에 서열이 있는 경우에 대한 로지스틱 회귀모형을 소개한다.

6.4.1 기본 R 스크립트

```
train_df <- tribble(
  ~N, ~L, ~y,
  25, 5, 3,
  25, 10, 3,
  25, 20, 2,
  25, 30, 1,
  32, 5, 3,
  32, 10, 3,
  32, 20, 2,
  32, 30, 1,
  42, 5, 1,
  42, 10, 3,
  42, 20, 1,
  42, 30, 1
) %>%
  mutate(y = as.ordered(y))

knitr::kable(train_df, booktabs = TRUE,
             align = c('r', 'r', 'r'),
             col.names = c('잡음(N)', '손실(L)', '만족도($y$)'),
             caption = '성능변수에 따른 통신 만족도')
```

Table 6.6은 벨 연구소에서 한 통신장치에 대하여 실시한 조사 결과를 나타낸 것이다. 주요 성능변수인 회선잡음(circuit noise: N)과 소리크기 손실(loudness loss: L)이 이용자의 주관적인 만족도에 미치는 영향을 분석하기 위한 것이다. 만족도는 원결과(Cavanaugh et al., 1976)를 가공하여 다음과 같이 3가지로 분류하였다.

$$y = \begin{cases} 1 & \text{good} \\ 2 & \text{fair} \\ 3 & \text{poor} \end{cases}$$

본 장에서는 두 가지 모형을 다룬다. 우선 누적 로짓모형(cumulative logit model)은 아래와 같이 MASS 패키지의 `polr` 함수를 사용하여 추정할 수 있다.

Table 6.6: 성능변수에 따른 통신 만족도

잡음(N)	손실(L)	만족도(\$y\$)
25	5	3
25	10	3
25	20	2
25	30	1
32	5	3
32	10	3
32	20	2
32	30	1
42	5	1
42	10	3
42	20	1
42	30	1

```
MASS::polr(y ~ N + L, data = train_df) %>%
  broom::tidy()
```

```
## # A tibble: 4 x 5
##   term estimate std.error statistic coefficient_type
##   <chr>    <dbl>     <dbl>     <dbl> <chr>
## 1 N        -0.224    0.146    -1.53 coefficient
## 2 L        -0.300    0.137    -2.19 coefficient
## 3 1|2      -13.0     6.46     -2.02 zeta
## 4 2|3      -11.4     6.17     -1.85 zeta
```

인근범주 로짓모형 (adjacent-categories logit model)은 아래와 같이 VGAM 패키지의 `vgglm` 함수를 이용하여 추정할 수 있다.

```
VGAM::vgglm(y ~ N + L,
             data = train_df,
             family = VGAM::acat(reverse = TRUE))
```

```
##
## Call:
## VGAM::vgglm(formula = y ~ N + L, family = VGAM::acat(reverse = TRUE),
##             data = train_df)
##
## 
## Coefficients:
## (Intercept):1 (Intercept):2           N:1           N:2           L:1
```

```

## -12.94227208 -6.10597713 0.31431599 0.04643039 0.17500408
##          L:2
## 0.29578067
##
## Degrees of Freedom: 24 Total; 18 Residual
## Residual deviance: 11.67769
## Log-likelihood: -5.838847
##
## This is an adjacent categories model with 3 levels

```

6.4.2 누적 로짓모형

객체 i 가 범주 j 이하에 속할 누적확률을 κ_{ij} 라 하자.

$$\kappa_{ij} = P(y_i \leq j | \mathbf{x}_i), j = 1, \dots, J$$

누적 로짓모형은 범주 누적확률의 로짓변환에 대한 선형 회귀모형이다.

$$\log \left(\frac{\kappa_{ij}}{1 - \kappa_{ij}} \right) = \beta_{0,j} + \boldsymbol{\beta}^\top \mathbf{x}_i, j = 1, \dots, J-1 \quad (6.16)$$

식 (6.16)은 독립변수에 대한 계수 $\boldsymbol{\beta}$ 가 모든 범주에 대해 동일하며 절편(intercept) $\beta_{0,j}$ 만 범주에 따라 다른 비례 승산 모형(proportional odds model)이다. 즉, 범주에 관계 없이 각 독립변수가 한 단위 증가할 때마다 로그 승산비는 동일하게 증가한다.

모형의 추정은 6.3.2절과 유사하게 다항분포를 사용한 최우추정법을 사용할 수 있다. 각 객체 i 가 범주 j 에 속할 확률은 아래와 같다.

$$\begin{aligned} \pi_{ij} &= \kappa_{ij} - \kappa_{i,j-1} \\ &= \frac{\exp(\beta_{0,j} + \boldsymbol{\beta}^\top \mathbf{x}_i)}{1 + \exp(\beta_{0,j} + \boldsymbol{\beta}^\top \mathbf{x}_i)} - \frac{\exp(\beta_{0,j-1} + \boldsymbol{\beta}^\top \mathbf{x}_i)}{1 + \exp(\beta_{0,j-1} + \boldsymbol{\beta}^\top \mathbf{x}_i)}, j = 2, \dots, J-1 \end{aligned}$$

$$\begin{aligned} \pi_{i1} &= \kappa_{i1} \\ &= \frac{\exp(\beta_{0,1} + \boldsymbol{\beta}^\top \mathbf{x}_i)}{1 + \exp(\beta_{0,1} + \boldsymbol{\beta}^\top \mathbf{x}_i)} \end{aligned}$$

$$\begin{aligned} \pi_{iJ} &= 1 - \kappa_{i,J-1} \\ &= 1 - \frac{\exp(\beta_{0,J-1} + \boldsymbol{\beta}^\top \mathbf{x}_i)}{1 + \exp(\beta_{0,J-1} + \boldsymbol{\beta}^\top \mathbf{x}_i)} \end{aligned}$$

로그 우도함수는

$$\sum_{i=1}^n \sum_{j=1}^J \log \pi_{ij}$$

이며, 이에 위에서 정리한 π_{ij} 식을 대입하여 전개할 수 있다. 이 로그 우도함수는 concave 함수이므로(Pratt, 1981), 각 계수에 대해 편미분하여 0이 되도록 하는 값을 구하는 방식으로 회귀모형을 추정할 수 있다.

```
polr_fit <- MASS::polr(y ~ N + L, data = train_df)
print(polr_fit)
```

```
## Call:
## MASS::polr(formula = y ~ N + L, data = train_df)
##
## Coefficients:
##             N          L
## -0.2236292 -0.2998833
##
## Intercepts:
##      1|2      2|3
## -13.03527 -11.39902
##
## Residual Deviance: 12.8825
## AIC: 20.8825
```

위와 같이 `polr` 함수 실행 시 얻어지는 각 변수들에 대한 계수들의 부호는 교재(전치혁, 2012)의 내용과 반대인데, 이는 `polr` 함수는 아래와 같은 모형을 추정하기 때문이다.

$$\log \left(\frac{\kappa_{ij}}{1 - \kappa_{ij}} \right) = \beta_{0,j} - \boldsymbol{\beta}^\top \mathbf{x}_i, j = 1, \dots, J - 1$$

위 모형에서 `polr` 함수 실행 결과 추정된 절편값은 $\beta_{0,1} = -13.0352721$, $\beta_{0,2} = -11.3990207$ 이며, 두 변수 N , L 에 대한 회귀계수는 각각 -0.2236292 , -0.2998833 로 추정된다.

추정된 회귀계수를 식 (6.4.2)에 대입하면 각 객체 i 가 각 범주 j 에 속할 확률을 Table 6.7와 같이 얻을 수 있다. 아래 R 스크립트에서 사용한 `predict`라는 함수가 해당 계산을 수행한다.

```
predict_df <- predict(polr_fit, train_df, type = "probs") %>%
  as_data_frame() %>%
  `colnames<-`(`c("p1", "p2", "p3")`) %>%
  mutate(pred_class = predict(polr_fit, train_df, type = "class"))
```

Table 6.7: 위 통신 만족도 데이터에 대한 누적 로짓모형의 추정범주

잡음(N)	손실(L)	실제범주 \$y_i\$	\$\pi_{i1}\$	\$\pi_{i2}\$	\$\pi_{i3}\$	추정범주 \$\hat{y}_i\$
25	5	3	0.0026	0.0107	0.9867	3
25	10	3	0.0116	0.0452	0.9432	3
25	20	2	0.1905	0.3567	0.4528	3
25	30	1	0.8252	0.1352	0.0396	1
32	5	3	0.0124	0.0481	0.9395	3
32	10	3	0.0531	0.1706	0.7763	3
32	20	2	0.5296	0.3230	0.1474	1
32	30	1	0.9576	0.0338	0.0085	1
42	5	1	0.1049	0.2709	0.6241	3
42	10	3	0.3443	0.3852	0.2705	2
42	20	1	0.9133	0.0685	0.0181	1
42	30	1	0.9953	0.0038	0.0009	1

```

bind_cols(train_df, predict_df) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r', 'r', 'r', 'r', 'r'),
    col.names = c('잡음(N)', '손실(L)', '실제범주 $y_i$', '$\pi_{i1}$', '$\pi_{i2}$', '$\pi_{i3}$', '추정범주 $\hat{y}_i$'),
    caption = '위 통신 만족도 데이터에 대한 누적 로짓모형의 추정범주',
    #   caption = 'Table \@ref(tab:ordinal-logistic-reg-train-data)에 대한 누적 로짓모형의 추정범주',
    digits = 4
  )

```

6.4.3 인근범주 로짓모형

인근범주 로짓모형은 아래와 같이 인접한 두 범주의 확률 비율에 대한 회귀모형이다.

$$\log \left(\frac{\pi_{ij}}{\pi_{i,j+1}} \right) = \beta_{0,j} + \boldsymbol{\beta}^\top \mathbf{x}_i, \quad j = 1, \dots, J-1 \quad (6.17)$$

따라서, π_{ij} 간에 다음과 같은 관계식이 성립한다.

$$\begin{aligned}\pi_{ij} &= \exp(\beta_{0,j} + \boldsymbol{\beta}^\top \mathbf{x}_i) \pi_{i,j+1} \\ &= \pi_{iJ} \exp\left(\sum_{k=j}^{J-1} \beta_{0,k} + (J-j)\boldsymbol{\beta}^\top \mathbf{x}_i\right), j = 1, \dots, J-1 \\ \sum_{j=1}^J \pi_{ij} &= 1\end{aligned}$$

o]를 정리하면

$$\begin{aligned}\pi_{ij} &= \frac{\exp\left(\sum_{l=j}^{J-1} \beta_{0,l} + (J-j)\boldsymbol{\beta}^\top \mathbf{x}_i\right)}{1 + \sum_{k=1}^{J-1} \exp\left(\sum_{l=k}^{J-1} \beta_{0,l} + (J-k)\boldsymbol{\beta}^\top \mathbf{x}_i\right)}, j = 1, \dots, J-1 \\ \pi_{iJ} &= \frac{1}{1 + \sum_{k=1}^{J-1} \exp\left(\sum_{l=k}^{J-1} \beta_{0,l} + (J-k)\boldsymbol{\beta}^\top \mathbf{x}_i\right)}\end{aligned}\tag{6.18}$$

와 같다. 이는 6.3.2절에서 살펴보았던 명목형 로지스틱 회귀모형에 비해 다소 복잡하지만 비슷한 형태이며, 역시 최우추정법을 이용하여 모형을 추정할 수 있다.

R에서는 VGAM 패키지의 `vglm` 함수를 이용할 때 파라미터 `family`의 값을 VGAM 패키지의 `acat` 함수를 설정함으로써 인근범주 로짓모형을 추정할 수 있다. 이 때 `acat` 함수의 `parallel` 파라미터값을 `TRUE`로 설정함으로써 식 (6.17)에서와 같이 비례 승산 모형 (proportional odds model)을 정의한다.

```
vglm_fit <- VGAM::vglm(
  y ~ N + L,
  data = train_df,
  family = VGAM::acat(reverse = TRUE, parallel = TRUE)
)

print(coef(vglm_fit))

## (Intercept):1 (Intercept):2           N           L
##      -9.0658976     -8.9018134     0.1725867    0.2082167
```

추정된 모형을 위 식 (6.18)에 대입하면 각 샘플 i 가 각 범주 j 에 속할 확률을 추정할 수 있다. VGAM 패키지의 `predictvglm` 함수가 해당 계산을 수행한다.

```
predict_df <- VGAM::predictvglm(vglm_fit, train_df, "response") %>%
  as_data_frame() %>%
```

Table 6.8: 위 통신 만족도 데이터에 대한 인근법주 로짓모형의 추정범주

잡음(N)	손실(L)	실제범주 \$y_i\$	\$\pi_{i1}\$	\$\pi_{i2}\$	\$\pi_{i3}\$	추정범주 \$\hat{y}_i\$
25	5	3	0.0007	0.0280	0.9713	3
25	10	3	0.0052	0.0751	0.9197	3
25	20	2	0.1804	0.3244	0.4952	3
25	30	1	0.7894	0.1770	0.0337	1
32	5	3	0.0072	0.0874	0.9054	3
32	10	3	0.0474	0.2045	0.7480	3
32	20	2	0.5611	0.3015	0.1375	1
32	30	1	0.9339	0.0626	0.0036	1
42	5	1	0.1393	0.3026	0.5581	3
42	10	3	0.4411	0.3385	0.2204	1
42	20	1	0.9063	0.0867	0.0070	1
42	30	1	0.9881	0.0118	0.0001	1

```

`colnames<-`c("p1", "p2", "p3")) %>%
  mutate(pred_class = ordered(apply(., 1, function(x) which.max(x)))) 

bind_cols(train_df, predict_df) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r', 'r', 'r', 'r', 'r'),
    col.names = c('잡음(N)', '손실(L)', '실제범주 $y_i$', 
                 '$\pi_{i1}$', '$\pi_{i2}$', '$\pi_{i3}$', '추정범주 $\hat{y}_i$'),
    caption = '위 통신 만족도 데이터에 대한 인근법주 로짓모형의 추정범주',
    #   caption = 'Table \@ref(tab:ordinal-logistic-reg-train-data)에 대한 인근법주 로짓모형의 추정범주',
    digits = 4
  )

```

비례 승산 모형(proportional odds model)이 아닌 인근법주 로짓모형은 아래와 같다.

$$\log \left(\frac{\pi_{ij}}{\pi_{i,j+1}} \right) = \beta_{0,j} + \boldsymbol{\beta}_j^\top \mathbf{x}_i, \quad j = 1, \dots, J-1 \quad (6.19)$$

해당 모형은 acat 함수의 parallel 파라미터 값을 FALSE로 설정함으로써 추정할 수 있다.

```

vglm_fit <- VGAM::vglm(
  y ~ N + L,
  data = train_df,
  family = VGAM::acat(reverse = TRUE, parallel = FALSE)

```

Table 6.9: 위 통신 만족도 데이터에 대한 인근법주 로짓모형의 추정법주 (비례 승산 모형이 아닌 경우)

잡음(N)	손실(L)	실제법주 $\$y_i\$$	$\$\\pi_{i1}\$$	$\$\\pi_{i2}\$$	$\$\\pi_{i3}\$$	추정법주 $\$\\hat{y}_i\$$
25	5	3	0.0004	0.0303	0.9693	3
25	10	3	0.0043	0.1200	0.8757	3
25	20	2	0.1295	0.6313	0.2392	2
25	30	1	0.5365	0.4546	0.0089	1
32	5	3	0.0055	0.0412	0.9533	3
32	10	3	0.0488	0.1517	0.7995	3
32	20	2	0.5924	0.3200	0.0876	1
32	30	1	0.9131	0.0857	0.0012	1
42	5	1	0.1667	0.0536	0.7797	3
42	10	3	0.6335	0.0850	0.2815	1
42	20	1	0.9734	0.0227	0.0039	1
42	30	1	0.9959	0.0040	0.0000	1

```
)
print(coef(vglm_fit))

## (Intercept):1 (Intercept):2           N:1           N:2           L:1
## -12.94227208   -6.10597713    0.31431599    0.04643039    0.17500408
##          L:2
##    0.29578067

predict_df <- VGAM::predictvglm(vglm_fit, train_df, "response") %>%
  as_data_frame() %>%
  `colnames<-`(`c("p1", "p2", "p3")) %>%
  mutate(pred_class = ordered(apply(., 1, function(x) which.max(x)))) 

bind_cols(train_df, predict_df) %>%
knitr::kable(
  booktabs = TRUE,
  align = c('r', 'r', 'r', 'r', 'r', 'r', 'r'),
  col.names = c('잡음(N)', '손실(L)', '실제법주  $\$y\_i\$$ ',
               ' $\$\\pi_{i1}\$$ ', ' $\$\\pi_{i2}\$$ ', ' $\$\\pi_{i3}\$$ ', '추정법주  $\$\\hat{y}\_i\$$ '),
  caption = '위 통신 만족도 데이터에 대한 인근법주 로짓모형의 추정법주 (비례 승산 모형이 아닌 경우)',
#  caption = 'Table \\ref{tab:ordinal-logistic-reg-train-data}에 대한 인근법주 로짓모형의 추
  digits = 4
)
```

Chapter 7

판별분석

7.1 개요

판별분석 (discriminant analysis)은 범주들을 가장 잘 구별하는 변수들의 하나 또는 다수의 함수를 도출하여 이를 기반으로 분류규칙을 제시한다. 본 장에서는 변수의 분포에 대한 가정이 필요 없는 피셔 (Fisher) 방법과 다변량 정규분포를 가정하는 선형 및 비선형 판별분석을 설명한다.

7.2 필요 R 패키지 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
MASS	7.3-51.4
mvtnorm	1.0-10

7.3 피셔 방법

7.3.1 기본 R 스크립트

```
train_df <- tibble(
  id = c(1:9),
  x1 = c(5, 4, 7, 8, 3, 2, 6, 9, 5),
  x2 = c(7, 3, 8, 6, 6, 5, 6, 6, 4),
  class = factor(c(1, 2, 2, 2, 1, 1, 2, 2), levels = c(1, 2))
```

Table 7.1: 판별분석 학습표본 데이터

객체번호	\$x_1\$	\$x_2\$	범주
1	5	7	1
2	4	3	2
3	7	8	2
4	8	6	2
5	3	6	1
6	2	5	1
7	6	6	1
8	9	6	2
9	5	4	2

```
)
knitr:::kable(train_df, booktabs = TRUE,
               align = c('r', 'r', 'r', 'r'),
               col.names = c('객체번호', '$x_1$', '$x_2$', '범주'),
               caption = '판별분석 학습표본 데이터')
```

Table 7.1와 같이 두 독립변수 x_1, x_2 와 이분형 종속변수 $class$ 의 관측값으로 이루어진 9 개의 학습표본을 $train_df$ 라는 data frame에 저장한다.

```
fisher_da <- MASS::lda(class ~ x1 + x2, train_df)

print(fisher_da)
```

```
## Call:
## lda(class ~ x1 + x2, data = train_df)
##
## Prior probabilities of groups:
##          1          2
## 0.4444444 0.5555556
##
## Group means:
##      x1  x2
## 1 4.0 6.0
## 2 6.6 5.4
##
## Coefficients of linear discriminants:
##                 LD1
## x1  0.6850490
```

```
## x2 -0.7003859
```

7.3.2 피셔 판별함수

각 객체는 변수벡터 $\mathbf{x} \in \mathbb{R}^p$ 와 범주 $y \in \{1, 2\}$ 로 이루어진다고 하자. 아래는 변수 \mathbf{x} 의 기대치와 분산-공분산행렬(variance-covariance matrix)을 나타낸다.

$$\begin{aligned}\boldsymbol{\mu}_1 &= E(\mathbf{x}|y=1) \\ \boldsymbol{\mu}_2 &= E(\mathbf{x}|y=2) \\ \boldsymbol{\Sigma} &= Var(\mathbf{x}|y=1) = Var(\mathbf{x}|y=2)\end{aligned}$$

다음과 같이 변수들의 선형조합으로 새로운 변수 z 를 형성하는 함수를 피셔 판별함수(Fisher's discriminant function)라 한다.

$$z = \mathbf{w}^\top \mathbf{x} \quad (7.1)$$

여기서 계수벡터 $\mathbf{w} \in \mathbb{R}^p$ 는 통상 아래와 같이 변수 z 의 범주간 평균 차이 대 변수 z 의 분산의 비율을 최대화하는 것으로 결정한다.

$$\operatorname{argmin}_{\mathbf{w}} \frac{\mathbf{w}^\top (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)}{\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}} \quad (7.2)$$

위 식 (7.2)의 해는

$$\mathbf{w} \propto \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

의 조건을 만족하며, 편의상 비례상수를 1로 두면 아래와 같은 해가 얻어진다.

$$\mathbf{w} = \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \quad (7.3)$$

실제 모집단의 평균 및 분산을 알지 못하는 경우, 학습표본으로부터 $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}$ 의 추정치를 얻어 식 (7.3)에 대입하는 방식으로 판별계수를 추정한다. 자세한 내용은 교재(전치혁, 2012) 참조.

Table 7.1에 주어진 학습표본을 이용하여 피셔 판별함수를 구해보도록 하자. 우선 각 범주별 평균벡터 $\hat{\boldsymbol{\mu}}_1, \hat{\boldsymbol{\mu}}_2$ 를 아래와 같이 구한다.

```
mu_hat <- train_df %>%
  group_by(class) %>%
  summarize(x1 = mean(x1),
            x2 = mean(x2)) %>%
  arrange(class)

print(mu_hat)
```

```
## # A tibble: 2 x 3
##   class     x1     x2
##   <fct> <dbl> <dbl>
## 1 1         4      6
## 2 2        6.6    5.4
```

또한 범주별 표본 분산-공분산행렬 $\mathbf{S}_1, \mathbf{S}_2$ 를 다음과 같이 구한다. 리스트 $S_{\text{within_group}}$ 의 첫번째 원소는 범주 1의 분산-공분산행렬 \mathbf{S}_1 , 두번째 원소는 범주 2의 분산-공분산행렬 \mathbf{S}_2 를 나타낸다.

```
S_within_group <- lapply(
  unique(train_df$class) %>% sort(), function(x) {
    train_df %>% filter(class == x) %>% select(x1, x2) %>% var()
  }
)

print(S_within_group)

## [[1]]
##           x1          x2
## x1 3.333333 1.0000000
## x2 1.000000 0.6666667
##
## [[2]]
##           x1          x2
## x1 4.30 2.95
## x2 2.95 3.80
```

위에서 얻은 범주별 표본 분산-공분산행렬을 이용하여 합동 분산-공분산행렬을 아래와 같이 추정한다.

$$\hat{\Sigma} = \mathbf{S}_p = \frac{(n_1 - 1)\mathbf{S}_1 + (n_2 - 1)\mathbf{S}_2}{n_1 + n_2 - 2}$$

이 때 n_1, n_2 는 각각 범주 1, 2에 속한 학습표본 객체의 수를 나타낸다. 아래 R 스크립트에서는 임의의 범주 표본수 벡터 \mathbf{n} 과 범주별 표본 분산-공분산행렬 리스트 \mathbf{S} 에 대해 합동 분산-공분산행렬을 구하는 함수 `pooled_variance`를 정의하고, 주어진 학습표본에 대한 입력값을 대입하여 합동 분산-공분산행렬 추정치 `Sigma_hat`을 구한다.

```
pooled_variance <- function(n, S) {
  lapply(1:length(n), function(i) (n[i] - 1)*S[[i]]) %>%
    Reduce(`+`, .) %>%
    `/` (sum(n) - length(n))
}
```

```

n_obs <- train_df %>%
  group_by(class) %>%
  count() %>%
  ungroup() %>%
  mutate(pi = n / sum(n)) %>%
  arrange(class)

Sigma_hat <- pooled_variance(n_obs$n, S_within_group)

print(Sigma_hat)

##           x1          x2
## x1 3.885714 2.114286
## x2 2.114286 2.457143

```

위에서 구한 추정치들을 이용하여 아래와 같이 판별함수 계수 추정치 \hat{w} 를 구한다.

$$\hat{w} = \hat{\Sigma}^{-1}(\hat{\mu}_1 - \hat{\mu}_2)$$

```

w_hat <- solve(Sigma_hat) %*% t(mu_hat[1, c('x1', 'x2')] - mu_hat[2, c('x1', 'x2')])

print(w_hat)

##           [,1]
## x1 -1.508039
## x2  1.541801

```

7.3.3 분류 규칙

피셔 판별함수에 따른 분류 경계값은 학습표본에 대한 판별함수값의 평균으로 아래와 같이 구할 수 있다.

$$\bar{z} = \frac{1}{N} \sum_i^N \hat{w}^\top \mathbf{x}_i$$

```

z_mean <- t(w_hat) %*% (train_df %>% select(x1, x2) %>% colMeans()) %>% drop()

print(z_mean)

## [1] 0.526438

```

Table 7.2: 학습표본에 대한 피셔 분류 결과

객체번호	\$x_1\$	\$x_2\$	실제범주	\$z\$	추정범주
1	5	7	1	3.2524116	1
2	4	3	2	-1.4067524	2
3	7	8	2	1.7781350	1
4	8	6	2	-2.8135048	2
5	3	6	1	4.7266881	1
6	2	5	1	4.6929260	1
7	6	6	1	0.2025723	2
8	9	6	2	-4.3215434	2
9	5	4	2	-1.3729904	2

위 결과를 통해, 분류규칙은 다음과 같이 주어진다.

- $\hat{\mathbf{w}}^\top \mathbf{x} \geq \bar{z}$ 이면, \mathbf{x} 를 범주 1로 분류
- $\hat{\mathbf{w}}^\top \mathbf{x} < \bar{z}$ 이면, \mathbf{x} 를 범주 2로 분류

```
train_prediction_df <- train_df %>%
  mutate(
    z = w_hat[1]*x1 + w_hat[2]*x2,
    predicted_class = factor(if_else(z >= z_mean, 1, 2), levels = c(1, 2))
  )

knitr::kable(train_prediction_df, booktabs = TRUE,
             align = c('r', 'r', 'r', 'r', 'r', 'r'),
             col.names = c('객체번호', '$x_1$', '$x_2$', '실제범주', '$z$', '추정범주'),
             caption = '학습표본에 대한 피셔 분류 결과')
```

위 결과 객체 3, 7가 오분류된다.

7.3.4 R 패키지를 이용한 분류규칙 도출

패키지 MASS내의 함수 `lda` 수행 시 얻어지는 판별계수 $\hat{\mathbf{w}}$ 는 위 결과와는 사뭇 다른데, `lda` 함수의 경우 아래와 같이 1) 제약식을 포함하여 비례계수를 구하기 때문에 계수의 크기가 달라지며, 2) 목적함수를 최소화하는 대신 최대화하는 값을 찾기 때문에 부호가 달라진다.

$$\begin{aligned} \max \quad & \mathbf{w}^\top (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ \text{s.t.} \quad & \mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w} = 1 \end{aligned}$$

이에 따른 `lda` 함수의 계수 추정 결과는 아래와 같다.

```

fisher_da <- MASS::lda(class ~ x1 + x2, train_df)

w_hat_lda <- fisher_da$scaling
print(w_hat_lda)

##          LD1
## x1  0.6850490
## x2 -0.7003859

z_mean_lda <- t(fisher_da$scaling) %*% (train_df %>% select(x1, x2) %>% colMeans()) %>% drop()
print(z_mean_lda)

##          LD1
## -0.2391423

```

위 결과는 아래와 같은 계산을 통해 앞 장에서 보았던 결과와 동일한 분류 경계식으로 표현될 수 있음을 볼 수 있다.

```

scale_adjust <- t(w_hat) %*% Sigma_hat %*% w_hat %>% drop() %>% sqrt()
sign_adjust <- -1

w_hat <- w_hat_lda * scale_adjust * sign_adjust
print(w_hat)

##          LD1
## x1 -1.508039
## x2  1.541801

z_mean <- z_mean_lda * scale_adjust * sign_adjust
print(z_mean)

##          LD1
## 0.526438

```

아래 스크립트는 위 lda 함수로부터의 경계식 추정을 기반으로 아래 수식값을 계산한다.

$$\hat{\mathbf{w}}^T \mathbf{x} - \bar{z}$$

```

predict(fisher_da, train_df)$x

```

Table 7.3: 학습표본에 대한 피셔 분류 결과 - 'MASS::lda' 분류 경계식 기준

객체번호	$\$x_1\$$	$\$x_2\$$	실제범주	$\$z - \bar{z}$	추정범주
1	5	7	1	-1.2383140	1
2	4	3	2	0.8781805	2
3	7	8	2	-0.5686020	1
4	8	6	2	1.5172187	2
5	3	6	1	-1.9080261	1
6	2	5	1	-1.8926892	1
7	6	6	1	0.1471208	2
8	9	6	2	2.2022677	2
9	5	4	2	0.8628436	2

```
##          LD1
## 1 -1.2383140
## 2  0.8781805
## 3 -0.5686020
## 4  1.5172187
## 5 -1.9080261
## 6 -1.8926892
## 7  0.1471208
## 8  2.2022677
## 9  0.8628436
```

피셔 분류규칙에 따라 해당 값이 0보다 작으면 범주 1, 0보다 크면 범주 2로 분류한다.

```
train_df %>%
  mutate(
    centered_z = predict(fisher_da, .)$x,
    predicted_class = factor(if_else(centered_z <= 0, 1, 2), levels = c(1, 2))
  ) %>%
  knitr::kable(booktabs = TRUE,
               align = c('r', 'r', 'r', 'r', 'r', 'r'),
               col.names = c('객체번호', '$x_1$', '$x_2$', '실제범주', '$z - \bar{z}$', '추정범주'),
               caption = '학습표본에 대한 피셔 분류 결과 - `MASS::lda` 분류 경계식 기준')
```

Table 7.3는 Table 7.2와 동일한 범주 추정 결과를 보인다.

7.4 의사결정론에 의한 선형분류규칙

다음과 같이 객체가 각 범주에 속할 사전확률과 각 범주 내에서의 분류변수의 확률밀도함수에 대한 기호를 정의한다.

- π_k : 임의의 객체가 범주 k 에 속할 사전확률
- $f_k(\mathbf{x})$: 범주 k 에 대한 변수의 확률밀도함수

이 때 통상적으로 \mathbf{x} 는 다변량 정규분포를 따르는 것으로 가정하여 아래와 같이 평균벡터 $\boldsymbol{\mu}_k$ 와 분산-공분산행렬 Σ 로 확률밀도함수를 정의할 수 있다. 이 때 분산-공분산행렬 Σ 는 모든 범주에 대해 동일하다고 가정한다.

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp\left\{-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right\} \quad (7.4)$$

본 장에서는 두 범주 ($k = 1, 2$) 분류 문제만 다루며, 세 범주 이상에 대한 분류 문제는 뒤 장에서 추가적으로 다루기로 한다.

7.4.1 기본 R 스크립트

Table 7.1의 학습표본에 대해 선형판별분석을 적용하는 R 스크립트는 아래에 보이는 것처럼 피서 판별함수를 구하기 위한 동일하며, `prior` 파라미터를 정의하지 않음으로써 π_1 과 π_2 를 학습표본의 범주 1, 2의 비율로 설정한다.

```
lda_fit <- MASS::lda(class ~ x1 + x2, train_df)

print(lda_fit)

## Call:
## lda(class ~ x1 + x2, data = train_df)
##
## Prior probabilities of groups:
##      1      2
## 0.4444444 0.5555556
##
## Group means:
##    x1   x2
## 1 4.0 6.0
## 2 6.6 5.4
##
## Coefficients of linear discriminants:
##          LD1
## x1  0.6850490
## x2 -0.7003859
```

7.4.2 선형판별함수

두 범주 문제에 있어서, 범주를 알지 못하는 변수 \mathbf{x} 에 대한 확률밀도함수는 아래와 같다.

$$f(\mathbf{x}) = \pi_1 f_1(\mathbf{x}) + \pi_2 f_2(\mathbf{x})$$

베이즈 정리(Bayes's theorem)에 따라 변수 \mathbf{x} 값이 주어졌을 때 범주 k 에 속할 사후확률(posterior)은 아래와 같이 구할 수 있다.

$$P(y = k | \mathbf{x}) = \frac{\pi_k f_k(\mathbf{x})}{f(\mathbf{x})} \quad (7.5)$$

각 범주에 대한 사후확률을 계산하여, 확률이 높은 쪽으로 범주를 추정한다.

$$\hat{y} = \begin{cases} 1, & \text{if } P(y = 1 | \mathbf{x}) \geq P(y = 2 | \mathbf{x}) \\ 2, & \text{otherwise} \end{cases} \quad (7.6)$$

이를 다시 정리하면 아래와 같다.

$$\hat{y} = \begin{cases} 1, & \text{if } \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} \geq \frac{\pi_2}{\pi_1} \\ 2, & \text{otherwise} \end{cases}$$

위 분류규칙에 식 (7.4)을 대입하여 정리하면 다음과 같다. 보다 자세한 내용은 교재 (전 치혁, 2012) 참조.

$$\hat{y} = \begin{cases} 1, & \text{if } \boldsymbol{\mu}_1^\top \boldsymbol{\Sigma}^{-1} \mathbf{x} - \frac{1}{2} \boldsymbol{\mu}_1^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 + \ln \pi_1 \geq \boldsymbol{\mu}_2^\top \boldsymbol{\Sigma}^{-1} \mathbf{x} - \frac{1}{2} \boldsymbol{\mu}_2^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_2 + \ln \pi_2 \\ 2, & \text{otherwise} \end{cases}$$

따라서, 각 범주에 대한 판별함수를

$$u_k(\mathbf{x}) = \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1} \mathbf{x} - \frac{1}{2} \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \ln \pi_k \quad (7.7)$$

라 하면, 아래와 같이 분류규칙을 정의할 수 있다.

$$\hat{y} = \begin{cases} 1, & \text{if } u_1(\mathbf{x}) \geq u_2(\mathbf{x}) \\ 2, & \text{otherwise} \end{cases} \quad (7.8)$$

Table 7.1의 학습표본에 대해 판별함수값을 계산하고 범주를 추정하면 아래와 같다.

```

discriminant_func <- function(X, mu, Sigma, pi) {
  Sigma_inv <- solve(Sigma)
  (t(mu) %*% Sigma_inv %*% X %>% drop()) -
    0.5 * (t(mu) %*% Sigma_inv %*% mu %>% drop()) +
    log(pi)
}

lda_discriminant_result_df <- train_df %>%
  mutate(
    u1 = discriminant_func(
      .[c("x1", "x2")], %>% t(),
      mu_hat[1, c("x1", "x2")], %>% unlist(),
      Sigma_hat,
      n_obs$pi[1]
    ),
    u2 = discriminant_func(
      .[c("x1", "x2")], %>% t(),
      mu_hat[2, c("x1", "x2")], %>% unlist(),
      Sigma_hat,
      n_obs$pi[2]
    )
  ) %>%
  mutate(
    predicted_class = factor(if_else(u1 >= u2, 1, 2), levels = c(1, 2))
  )

knitr::kable(
  lda_discriminant_result_df,
  booktabs = TRUE,
  align = rep('r', dim(lda_discriminant_result_df)[2]),
  col.names = c('액체번호', '$x_1$', '$x_2$',
               '실제범주', '$u_1(\mathbf{x})$', '$u_2(\mathbf{x})$',
               '추정범주'),
  caption = '학습표본에 대한 LDA 적용 결과: 판별함수값 및 추정범주')

```

또한 식 (7.5)에 따른 사후확률과 식 (7.6)에 따른 추정범주는 아래와 같이 얻어진다.

```

lda_posterior_result_df <- train_df %>%
  mutate(
    f1 = mvtnorm::dmvnorm(
      .[c("x1", "x2")],
      mu_hat[1, c("x1", "x2")], %>% unlist(),
      Sigma_hat),
    f2 = mvtnorm::dmvnorm(
      .[c("x1", "x2")],

```

Table 7.4: 학습표본에 대한 LDA 적용 결과: 판별함수값 및 추정범주

객체번호	x_1	x_2	실제범주	$u_1(\mathbf{x})$	$u_2(\mathbf{x})$	추정범주
1	5	7	1	9.2051470	6.971538	1
2	4	3	2	-1.9363321	0.489223	2
3	7	8	2	11.0057900	10.246458	1
4	8	6	2	4.5909990	8.423307	2
5	3	6	1	7.4045039	3.696619	1
6	2	5	1	5.0411598	1.367036	1
7	6	6	1	5.7164010	6.532631	2
8	9	6	2	4.0282981	9.368644	2
9	5	4	2	0.4270119	2.818805	2

```

mu_hat[2, c("x1", "x2")] %>% unlist(),
Sigma_hat),
f = n_obs$pi[1] * f1 + n_obs$pi[2] * f2
) %>%
mutate(
  p1 = n_obs$pi[1] * f1 / f,
  p2 = n_obs$pi[2] * f2 / f
) %>%
mutate(
  predicted_class = factor(if_else(p1 >= p2, 1, 2), levels = c(1, 2))
) %>%
select(
  id, x1, x2, class, p1, p2, predicted_class
)

knitr::kable(
  lda_posterior_result_df,
  booktabs = TRUE,
  align = rep('r', dim(lda_posterior_result_df)[2]),
  col.names = c('객체번호', '$x_1$', '$x_2$',
               '실제범주',
               '$P(y = 1 | \mathbf{x})$',
               '$P(y = 2 | \mathbf{x})$',
               '추정범주'),
  caption = '학습표본에 대한 LDA 적용 결과: 사후확률 및 추정범주')

```

때까지 MASS내의 함수 `lda`를 통해 위 Table 7.5 결과를 간편하게 얻을 수 있다.

```
lda_fit <- MASS::lda(class ~ x1 + x2, train_df)
```

Table 7.5: 학습표본에 대한 LDA 적용 결과: 사후확률 및 추정범주

객체번호	x_1	x_2	실제범주	$P(y = 1 \mathbf{x})$	$P(y = 2 \mathbf{x})$	추정범주
1	5	7	1	0.9032273	0.0967727	1
2	4	3	2	0.0812446	0.9187554	2
3	7	8	2	0.6812088	0.3187912	1
4	8	6	2	0.0212004	0.9787996	2
5	3	6	1	0.9760579	0.0239421	1
6	2	5	1	0.9752562	0.0247438	1
7	6	6	1	0.3065644	0.6934356	2
8	9	6	2	0.0047713	0.9952287	2
9	5	4	2	0.0838007	0.9161993	2

```

train_df %>% bind_cols(
  predict(lda_fit, train_df)$posterior %>%
    `colnames<-` (paste0("p", colnames(.))) %>% as_data_frame()
) %>%
  mutate(
  predicted_class = predict(lda_fit, .)$class
)
## # A tibble: 9 x 7
##       id     x1     x2 class      p1      p2 predicted_class
##   <int> <dbl> <dbl> <fct>    <dbl>    <dbl> <fct>
## 1     1     5     7 1        0.903    0.0968 1
## 2     2     4     3 2        0.0812   0.919   2
## 3     3     7     8 2        0.681    0.319   1
## 4     4     8     6 2        0.0212   0.979   2
## 5     5     3     6 1        0.976    0.0239  1
## 6     6     2     5 1        0.975    0.0247  1
## 7     7     6     6 1        0.307    0.693   2
## 8     8     9     6 2        0.00477  0.995   2
## 9     9     5     4 2        0.0838   0.916   2

```

위 결과들은 교재 (전치혁, 2012)의 예제 결과와는 다소 차이가 있는데, 이는 교재에서는 사전확률을 학습표본 내 비율 대신 $\pi_1 = \pi_2 = 0.5$ 로 지정하였기 때문이다. 교재와 동일한 결과는 아래의 스크립트처럼 lda 함수 실행 시 사전확률 파라미터 prior의 값을 지정함으로써 얻을 수 있다.

```

lda_fit_equal_prior <- MASS::lda(class ~ x1 + x2, train_df, prior = c(1/2, 1/2))

train_df %>% bind_cols(
  predict(lda_fit_equal_prior, train_df)$posterior %>%

```

Table 7.6: 학습표본에 대한 LDA 적용 결과: 사후확률 및 추정범주 (사전확률 = 0.5)

객체번호	x_1	x_2	실제범주	$P(y = 1 \mathbf{x})$	$P(y = 2 \mathbf{x})$	추정범주
1	5	7	1	0.9210538	0.0789462	
2	4	3	2	0.0995341	0.9004659	
3	7	8	2	0.7275992	0.2724008	
4	8	6	2	0.0263608	0.9736392	
5	3	6	1	0.9807542	0.0192458	
6	2	5	1	0.9801065	0.0198935	
7	6	6	1	0.3559269	0.6440731	
8	9	6	2	0.0059571	0.9940429	
9	5	4	2	0.1026013	0.8973987	

```

`colnames<-`(paste0("p", colnames(.))) %>% as_data_frame()
) %>%
mutate(
  predicted_class = predict(lda_fit_equal_prior, .)$class
) %>%
knitr::kable(
  booktabs = TRUE,
  align = rep('r', dim(lda_posterior_result_df)[2]),
  col.names = c('객체번호', '$x_1$', '$x_2$',
               '실제범주',
               '$P(y = 1 | \mathbf{x})$',
               '$P(y = 2 | \mathbf{x})$',
               '추정범주'),
  caption = '학습표본에 대한 LDA 적용 결과: 사후확률 및 추정범주 (사전확률 = 0.5)')

```

7.5 오분류비용을 고려한 분류규칙

위 Table 7.5의 객체 3, 7와 같이 선형분류함수가 모든 객체의 범주를 정확하게 추정하지 못하고 오분류가 발생하는 경우가 있다. 이 때 다음과 같이 두 종류의 오분류 비용이 있다고 가정하자.

- $C(1|2)$: 범주 2를 1로 잘못 분류 시 초래 비용
- $C(2|1)$: 범주 1를 2로 잘못 분류 시 초래 비용

이 때 총 기대 오분류 비용은 다음과 같다.

$$C(1|2)\pi_2 \int_{\mathbf{x} \in R_1} f_2(\mathbf{x}) d\mathbf{x} + C(2|1)\pi_1 \int_{\mathbf{x} \in R_2} f_1(\mathbf{x}) d\mathbf{x} \quad (7.9)$$

여기에서 $R_1 \subset \mathbb{R}^p, R_2 = \mathbb{R}^p - R_1$ 는 판별함수에 의해 각각 범주 1, 2로 분류되는 판별변수 영역을 나타낸다. 즉,

$$\hat{y} = \begin{cases} 1, & \text{if } \mathbf{x} \in R_1 \\ 2, & \text{otherwise} \end{cases}$$

식 (7.9)을 최소화하는 영역 R_1, R_2 는 아래와 같다.

$$\begin{aligned} R_1 &= \left\{ \mathbf{x} \in \mathbb{R}^p : \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} \geq \frac{\pi_2}{\pi_1} \left(\frac{C(1|2)}{C(2|1)} \right) \right\} \\ R_2 &= \left\{ \mathbf{x} \in \mathbb{R}^p : \frac{f_1(\mathbf{x})}{f_2(\mathbf{x})} < \frac{\pi_2}{\pi_1} \left(\frac{C(1|2)}{C(2|1)} \right) \right\} \end{aligned}$$

위 중 R_1 에 대한 식을 아래와 같이 단계적으로 전개할 수 있다.

$$\begin{aligned} R_1 &= \left\{ \mathbf{x} \in \mathbb{R}^p : \frac{\pi_1 f_1(\mathbf{x})}{\pi_2 f_2(\mathbf{x})} \geq \frac{C(1|2)}{C(2|1)} \right\} \\ &= \left\{ \mathbf{x} \in \mathbb{R}^p : \frac{\frac{\pi_1 f_1(\mathbf{x})}{\pi_1 f_1(\mathbf{x}) + \pi_2 f_2(\mathbf{x})}}{\frac{\pi_2 f_2(\mathbf{x})}{\pi_1 f_1(\mathbf{x}) + \pi_2 f_2(\mathbf{x})}} \geq \frac{C(1|2)}{C(2|1)} \right\} \\ &= \left\{ \mathbf{x} \in \mathbb{R}^p : \frac{P(y=1|\mathbf{x})}{P(y=2|\mathbf{x})} \geq \frac{C(1|2)}{C(2|1)} \right\} \\ &= \{ \mathbf{x} \in \mathbb{R}^p : C(2|1)P(y=1|\mathbf{x}) \geq C(1|2)P(y=2|\mathbf{x}) \} \end{aligned}$$

따라서 오분류비용을 고려한 분류규칙은 1) 사후확률에 오분류 비용을 곱한 뒤, 2) 그 값이 큰 범주로 분류하여 오분류비용을 최소화한다.

Table 7.5에 오분류비용 $C(1|2) = 1, C(2|1) = 5$ 를 적용한 결과는 아래와 같이 구할 수 있다.

```
lda_fit <- MASS::lda(class ~ x1 + x2, train_df)

misclassification_cost <- c(5, 1)

lda_unequal_cost_result_df <- train_df %>% bind_cols(
  predict(lda_fit, train_df)$posterior %*% diag(misclassification_cost) %>%
    as_data_frame() %>%
    `names<-`(`paste0("s", lda_fit$lev)) )
) %>%
mutate(
  predicted_class = factor(if_else(s1 >= s2, 1, 2), levels = c(1, 2))
```

Table 7.7: 학습표본에 대한 오분류 비용을 고려한 LDA 적용 결과

객체번호	$\$x_1\$$	$\$x_2\$$	실제범주	$\$C(2 \backslash, \backslash, 1) P(y = 1 \mathbf{x})\$$	$\$C(1 \backslash, \backslash, 2) P(y = 2 \mathbf{x})\$$
1	5	7	1		4.5161363
2	4	3	2		0.4062232
3	7	8	2		3.4060438
4	8	6	2		0.1060019
5	3	6	1		4.8802897
6	2	5	1		4.8762808
7	6	6	1		1.5328222
8	9	6	2		0.0238567
9	5	4	2		0.4190033

```
)
knitr::kable(
  lda_unequal_cost_result_df,
  booktabs = TRUE,
  align = rep('r', dim(lda_unequal_cost_result_df)[2]),
  col.names = c('객체번호', '$x_1$', '$x_2$',
    '실제범주',
    '$C(2 \backslash, | \backslash, 1) P(y = 1 | \mathbf{x})$',
    '$C(1 \backslash, | \backslash, 2) P(y = 2 | \mathbf{x})$',
    '추정범주'),
  caption = '학습표본에 대한 오분류 비용을 고려한 LDA 적용 결과')
```

위 Table 7.7에서 보는 바와 같이 오분류 객체는 3로, 이전 장의 Table 7.5에 비해 실제범주가 1인 객체를 더 정확하게 분류함을 확인할 수 있다. 범주 1인 객체를 범주 2로 분류할 때 발생하는 비용이 범주 2인 객체를 범주 1로 분류할 때 발생하는 비용보다 다섯 배나 크기 때문에, 오분류비용을 고려한 분류규칙은 실제 범주가 2인 객체를 범주 2로 정확하게 분류할 확률이 줄어든다 할지라도, 실제 범주가 1인 객체를 범주 1로 정확하게 분류하는 확률을 높이는 방향으로 학습된다.

7.6 이차판별분석

이차판별분석은 판별함수가 변수들에 대한 이차함수로 표현되는 경우인데, 각 범주에 대한 변수벡터 \mathbf{x} 가 서로 다른 분산-공분산행렬을 갖는 다변량 정규분포를 따를 때 의사결정론에 의한 분류규칙으로부터 유도된다.

7.6.1 기본 R 스크립트

Table 7.1의 학습표본에 대해 이차판별분석을 적용하는 R 스크립트는 아래에 보이는 것과 같이 MASS 패키지의 qda 함수를 사용한다.

```
qda_fit <- MASS::qda(class ~ x1 + x2, train_df)

print(qda_fit)
```

```
## Call:
## qda(class ~ x1 + x2, data = train_df)
##
## Prior probabilities of groups:
##      1      2
## 0.4444444 0.5555556
##
## Group means:
##   x1  x2
## 1 4.0 6.0
## 2 6.6 5.4
```

7.6.2 이차 판별함수

각 범주의 확률밀도함수는 아래와 같이 다변량 정규분포로 정의된다.

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{p/2} |\Sigma_k|^{1/2}} \exp\left\{-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)\right\} \quad (7.10)$$

위 식 (7.10)이 선형판별함수에서 사용한 식 (7.4)과 다른 부분은 분산-공분산분포 $\boldsymbol{\Sigma}_k$ 가 범주 k 에 대해 각각 정의된다는 점이다.

이 경우 각 범주에 대한 판별함수는 아래와 같이 정의된다.

$$u_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_k) - \frac{1}{2} \ln |\boldsymbol{\Sigma}_k| + \ln \pi_k \quad (7.11)$$

데이터 행렬 $X = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$ 의 각 객체에 대한 판별함수값을 얻는 함수를 아래와 같이 구현할 수 있다.

```
qda_discriminant_func <- function(X, mu, Sigma, pi) {
  Sigma_inv_sqrt <- chol(solve(Sigma))
  - 0.5 * rowSums((t(X - mu) %*% t(Sigma_inv_sqrt))^2) - 0.5 * log(det(Sigma)) + log(pi)
}
```

7.6.3 이차판별함수에 의한 분류

분류기준은 선형판별분석과 마찬가지로 판별함수값이 큰 범주로 분류한다.

$$\hat{y} = \begin{cases} 1, & \text{if } u_1(\mathbf{x}) \geq u_2(\mathbf{x}) \\ 2, & \text{otherwise} \end{cases}$$

Table 7.1의 학습표본에 대해 이차판별함수값을 계산하고 범주를 추정하면 아래와 같다.

```
qda_discriminant_result_df <- train_df %>% mutate(
  u1 = qda_discriminant_func(
    .[c("x1", "x2")] %>% t(),
    mu_hat[1, c("x1", "x2")] %>% unlist(),
    S_within_group[[1]],
    n_obs$pi[1]
  ),
  u2 = qda_discriminant_func(
    .[c("x1", "x2")] %>% t(),
    mu_hat[2, c("x1", "x2")] %>% unlist(),
    S_within_group[[2]],
    n_obs$pi[2]
  )
) %>%
  mutate(
    predicted_class = factor(if_else(u1 >= u2, 1, 2), levels = c(1, 2))
  )

knitr::kable(
  qda_discriminant_result_df,
  booktabs = TRUE,
  align = rep('r', dim(qda_discriminant_result_df)[2]),
  col.names = c('객체번호', '$x_1$', '$x_2$',
               '실제범주', '$u_1(\mathbf{x})$', '$u_2(\mathbf{x})$',
               '추정범주'),
  caption = '학습표본에 대한 QDA 적용 결과: 판별함수값 및 추정범주')
```

위 Table 7.8에서 보듯이 모든 학습객체가 올바로 분류되고 있다.

또한 선형판별분석의 경우와 마찬가지로 사후확률 비교를 통한 범주 분류를 수행할 수 있다.

```
qda_posterior_result_df <- train_df %>%
  mutate(
    f1 = mvtnorm::dmvnorm(
      .[c("x1", "x2")],
```

Table 7.8: 학습표본에 대한 QDA 적용 결과: 판별함수값 및 추정범주

객체번호	x_1	x_2	실제범주	$u_1(\mathbf{x})$	$u_2(\mathbf{x})$	추정범주
1	5	7	1	-1.729447	-3.950639	1
2	4	3	2	-13.183993	-2.497284	2
3	7	8	2	-3.911266	-3.145402	2
4	8	6	2	-5.274902	-1.868806	2
5	3	6	1	-1.183993	-5.764060	1
6	2	5	1	-1.729447	-6.202685	1
7	6	6	1	-2.002175	-1.934273	2
8	9	6	2	-7.729447	-2.582391	2
9	5	4	2	-8.274902	-1.927726	2

```

mu_hat[1, c("x1", "x2")] %>% unlist(),
S_within_group[[1]]),
f2 = mvtnorm::dmvnorm(
  .[c("x1", "x2")],
  mu_hat[2, c("x1", "x2")] %>% unlist(),
  S_within_group[[2]]),
f = n_obs$pi[1] * f1 + n_obs$pi[2] * f2
) %>%
mutate(
  p1 = n_obs$pi[1] * f1 / f,
  p2 = n_obs$pi[2] * f2 / f
) %>%
mutate(
  predicted_class = factor(if_else(p1 >= p2, 1, 2), levels = c(1, 2))
) %>%
select(
  id, x1, x2, class, p1, p2, predicted_class
)

knitr::kable(
  qda_posterior_result_df,
  booktabs = TRUE,
  align = rep('r', dim(qda_posterior_result_df)[2]),
  col.names = c('객체번호', '$x_1$', '$x_2$',
               '실제범주',
               '$P(y = 1 | \mathbf{x})$',
               '$P(y = 2 | \mathbf{x})$',
               '추정범주'),
  caption = '학습표본에 대한 QDA 적용 결과: 사후확률 및 추정범주')

```

Table 7.9: 학습표본에 대한 QDA 적용 결과: 사후확률 및 추정범주

객체번호	x_1	x_2	실제범주	$P(y = 1 \mathbf{x})$	$P(y = 2 \mathbf{x})$	추정범주
1	5	7	1	0.9021365	0.0978635	
2	4	3	2	0.0000228	0.9999772	
3	7	8	2	0.3173746	0.6826254	
4	8	6	2	0.0321055	0.9678945	
5	3	6	1	0.9898499	0.0101501	
6	2	5	1	0.9887184	0.0112816	
7	6	6	1	0.4830310	0.5169690	
8	9	6	2	0.0057829	0.9942171	
9	5	4	2	0.0017486	0.9982514	

이 또한 MASS 패키지의 `predict.qda` 함수를 통해 아래와 같이 동일한 결과값을 보다 간편하게 얻을 수 있다.

```
train_df %>% bind_cols(
  predict(qda_fit, train_df)$posterior %>%
    `colnames<-`(paste0("p", colnames(.))) %>% as_data_frame()
) %>%
  mutate(
  predicted_class = predict(qda_fit, .)$class
)

## # A tibble: 9 x 7
##       id     x1     x2 class      p1      p2 predicted_class
##   <int> <dbl> <dbl> <fct>    <dbl>    <dbl> <fct>
## 1     1     5     7 1        0.902    0.0979 1
## 2     2     4     3 2        0.0000228 1.000  2
## 3     3     7     8 2        0.317    0.683  2
## 4     4     8     6 2        0.0321   0.968  2
## 5     5     3     6 1        0.990    0.0102 1
## 6     6     2     5 1        0.989    0.0113 1
## 7     7     6     6 1        0.483    0.517  2
## 8     8     9     6 2        0.00578  0.994  2
## 9     9     5     4 2        0.00175  0.998  2
```

7.7 세 범주 이상의 분류

7.7.1 기본 R 스크립트

3개의 범주를 지닌 붓꽃(iris) 데이터에 대해 선형판별분석을 적용하는 R 스크립트는 아래와 같다. 본 예제에서는 각 범주별 50개 데이터 중 첫 30개 관측치만을 학습표본으로 삼아 판별함수를 유도한다.

```
iris_train_df <- datasets::iris %>%
  rename(x1 = Sepal.Length,
         x2 = Sepal.Width,
         x3 = Petal.Length,
         x4 = Petal.Width,
         class = Species) %>%
  group_by(class) %>%
  slice(1:30) %>%
  ungroup() %>%
  mutate(id = row_number())

iris_lda_fit <- MASS::lda(class ~ . - id, iris_train_df)

print(iris_lda_fit)

## Call:
## lda(class ~ . - id, data = iris_train_df)
##
## Prior probabilities of groups:
##   setosa versicolor virginica
## 0.3333333 0.3333333 0.3333333
##
## Group means:
##           x1        x2        x3        x4
## setosa    5.0266667 3.450000 1.473333 0.2466667
## versicolor 6.0700000 2.790000 4.333333 1.3533333
## virginica  6.5833333 2.933333 5.603333 2.0066667
##
## Coefficients of linear discriminants:
##          LD1       LD2
## x1  0.5711419 -1.2397647
## x2  1.8752911  3.0223980
## x3 -1.7361767  0.3159667
## x4 -3.4672646  1.3954748
##
## Proportion of trace:
##      LD1       LD2
```

```
## 0.9929 0.0071
```

7.7.2 일반화된 판별함수

$K (> 2)$ 개의 범주가 있는 경우에 대한 판별분석은 아래와 같이 일반화된다.

- π_k : 범주 k 에 속할 사전확률, $k = 1, 2, \dots, K$
- $C(k' | k) \geq 0$: 실제 범주 k 에 속하는 데 범주 k' 로 분류할 때 소요 비용 ($C(k' | k) = 0$ if $k' = k$)
- $f_k(\mathbf{x})$: 범주 k 에 속하는 \mathbf{x} 의 확률밀도함수
- $R_k \subset \mathbb{R}^p$: 범주 k 로 분류되는 \mathbf{x} 의 영역
- $P(k' | k) = \int_{\mathbf{x} \in R_{k'}} f_k(\mathbf{x}) d\mathbf{x}$: 실제 범주 k 에 속하는 데 범주 k' 로 분류할 확률

이 때, 총 기대 오분류 비용은 아래와 같다.

$$\sum_{k=1}^K \pi_k \sum_{k' \neq k} C(k' | k) \int_{\mathbf{x} \in R_{k'}} f_k(\mathbf{x}) d\mathbf{x}$$

따라서 분류문제는 위 총 기대 오분류 비용을 최소화하는 R_1, \dots, R_K 를 찾는 것이다.

우선, 범주를 고려하지 않은 \mathbf{x} 의 확률밀도함수는 아래와 같이 정의된다.

$$f(\mathbf{x}) = \sum_{k=1}^K \pi_k f_k(\mathbf{x})$$

베이즈 정리에 의하여, 변수 \mathbf{x} 가 주어졌을 때 범주 k 에 속할 사후확률은 아래와 같다.

$$P(y = k | \mathbf{x}) = \frac{\pi_k f_k(\mathbf{x})}{f(\mathbf{x})}$$

오분류비용이 동일한 경우에는 각 객체에 대해 위의 사후확률이 가장 큰 범주로 추정한다.
위 식에서

$$P(y = k | \mathbf{x}) \propto \pi_k f_k(\mathbf{x})$$

이므로, 아래와 같이 범주가 추정된다.

$$\hat{y} = \arg \max_k \pi_k f_k(\mathbf{x})$$

앞 장들에서 살펴본 것과 마찬가지로, 선형판별분석의 경우 각 범주의 확률밀도함수 $f_k(\mathbf{x})$ 가 동일 분산-공분산행렬을 가정하며, 이차판별분석의 경우 서로 다른 분산-공분산행렬을 가정한다.

아래 스크립트는 MASS 패키지의 lda 함수를 통해 각 범주에 속할 사후확률과 범주 추정값을 얻는 과정을 보여준다.

```
iris_lda_fit <- MASS::lda(class ~ x1 + x2 + x3 + x4, iris_train_df)

iris_lda_result <- iris_train_df %>%
  bind_cols(
    predict(iris_lda_fit, .)$posterior %>%
      as_data_frame()
  ) %>%
  mutate(
    predicted_class = predict(iris_lda_fit, .)$class
  )

print(iris_lda_result)

## # A tibble: 90 x 10
##       x1     x2     x3     x4 class     id setosa versicolor virginica
##   <dbl> <dbl> <dbl> <dbl> <fct> <int> <dbl>      <dbl>      <dbl>
## 1  5.1   3.5   1.4   0.2 seto~     1   1   5.57e-22  6.34e-42
## 2  4.9   3.0   1.4   0.2 seto~     2   1   3.32e-17  5.67e-36
## 3  4.7   3.2   1.3   0.2 seto~     3   1   2.75e-19  2.14e-38
## 4  4.6   3.1   1.5   0.2 seto~     4   1   8.12e-17  5.62e-35
## 5  5.0   3.6   1.4   0.2 seto~     5   1   1.14e-22  1.25e-42
## 6  5.4   3.9   1.7   0.4 seto~     6   1   3.20e-21  4.38e-40
## 7  4.6   3.4   1.4   0.3 seto~     7   1   8.74e-19  4.07e-37
## 8  5.0   3.4   1.5   0.2 seto~     8   1   3.27e-20  1.62e-39
## 9  4.4   2.9   1.4   0.2 seto~     9   1.000  2.22e-15  3.42e-33
## 10 4.9   3.1   1.5   0.1 seto~    10   1   9.36e-19  4.85e-38
## # ... with 80 more rows, and 1 more variable: predicted_class <fct>

knitr::kable(
  iris_lda_result %>%
    select(id, class, predicted_class,
           setosa, versicolor, virginica) %>%
    filter(class != predicted_class),
  booktabs = TRUE,
  align = rep('r', 6),
  col.names = c('객체번호', '실제범주', '추정범주',
               'setosa', 'versicolor', 'virginica'),
  caption = '붓꽃 학습표본에 대한 LDA 적용 결과 - 오분류 객체 사후 확률')
```

아래 스크립트는 MASS 패키지의 qda 함수를 통해 각 범주에 속할 사후확률과 범주 추정값을 얻는 과정을 보여준다.

Table 7.10: 붓꽃 학습표본에 대한 LDA 적용 결과 - 오분류 객체 사후 확률

객체번호	실제범주	추정범주	setosa	versicolor	virginica
51	versicolor	virginica	0	0.3088912	0.6911088

Table 7.11: 붓꽃 학습표본에 대한 QDA 적용 결과 - 오분류 객체 사후 확률

객체번호	실제범주	추정범주	setosa	versicolor	virginica
51	versicolor	virginica	0	0.4274712	0.5725288

```

iris_qda_fit <- MASS::qda(class ~ x1 + x2 + x3 + x4, iris_train_df)

iris_qda_result <- iris_train_df %>%
  bind_cols(
    predict(iris_qda_fit, .)$posterior %>%
      as_data_frame()
  ) %>%
  mutate(
    predicted_class = predict(iris_qda_fit, .)$class
  )

knitr::kable(
  iris_qda_result %>%
    select(id, class, predicted_class,
           setosa, versicolor, virginica) %>%
    filter(class != predicted_class),
  booktabs = TRUE,
  align = rep('r', 6),
  col.names = c('객체번호', '실제범주', '추정범주',
               'setosa', 'versicolor', 'virginica'),
  caption = '붓꽃 학습표본에 대한 QDA 적용 결과 - 오분류 객체 사후 확률')

```

위 결과에서 선형판별분석과 이차판별분석은 동일한 객체를 오분류한다. 해당 객체의 실제 범주에 대한 사후확률은 이차판별분석 결과에서 보다 높게 나타난다.

Chapter 8

트리기반 기법

8.1 CART 개요

CART(Classification and Regression Trees)는 Breiman et al. (1984)에 의하여 개발된 것인데, 각 (독립) 변수를 이분화(binary split)하는 과정을 반복하여 트리 형태를 형성함으로써 분류(종속변수가 범주형일 때) 또는 회귀분석(종속변수가 연속형일 때)을 수행하는 것이다. 이 때 독립변수들은 범주형 또는 연속형 모두에 적용될 수 있다. 본 장에서는 분류를 위한 목적만을 설명하도록 한다.

8.2 필요 R package 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
rpart	4.1-15
rpart.plot	3.0.7

8.3 CART 트리 생성

8.3.1 기본 R 스크립트

```
train_df <- tibble(  
  x1 = c(1,2,2,2,2,3,4,4,4,5),  
  x2 = c(4,6,5,4,3,6,6,5,4,3),
```

Table 8.1: 학습표본 데이터

x1	x2	class
1	4	1
2	6	1
2	5	1
2	4	2
2	3	2
3	6	1
4	6	1
4	5	2
4	4	2
5	3	2

```
    class = as.factor(c(1,1,1,2,2,1,1,2,2,2))
)
```

Table 8.1와 같이 두 독립변수 x_1 , x_2 와 이분형 종속변수 $class$ 의 관측값으로 이루어진 10개의 학습표본을 $train_df$ 라는 data frame에 저장한다.

```
library(rpart)
library(rpart.plot)
cart.est <- rpart(
  class ~ x1 + x2
  , data = train_df
  , method = "class"
  , parms = list(split = "gini")
  , control = rpart.control(minsplit = 2
                            , minbucket = 1
                            , cp = 0
                            , xval = 0
                            , maxcompete = 0)
)
rpart.plot(cart.est)
```

`rpart`라는 package를 기반으로, 두 변수 x_1 과 x_2 를 이용하여 이분형 종속변수 $class$ 를 분류하는 CART 트리를 생성할 수 있으며, `rpart.plot` package를 이용하여 Figure 8.1과 같이 시각화할 수 있다.

8.3.2 기호 정의

본 장에서 사용될 수학적 기호는 아래와 같다.

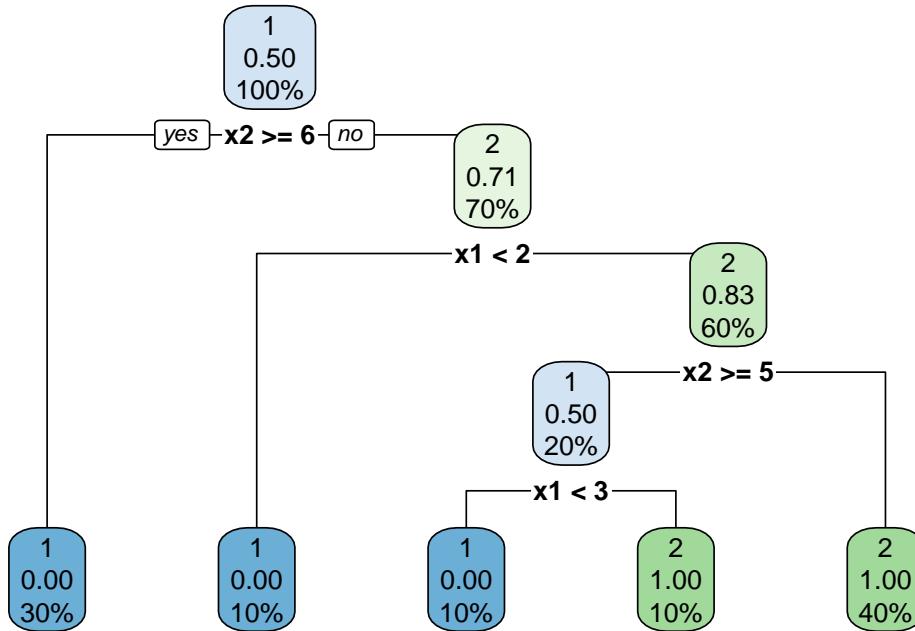


Figure 8.1: CART 트리

- T : 트리
- $A(T)$: 트리 T 의 최종노드의 집합
- J : 범주수
- N : 학습표본의 총 객체수
- N_j : 범주 j 에 속한 객체 수
- $N(t)$: 노드 t 에서의 객체수
- $N_j(t)$: 노드 t 에서 범주 j 에 속한 객체수
- $p(j, t)$: 임의의 객체가 범주 j 와 노드 t 에 속할 확률
- $p(t)$: 임의의 객체가 노드 t 에 속할 확률

$$p(t) = \sum_{j=1}^J p(j, t)$$

- $p(j|t)$: 임의의 객체가 노드 t 에 속할 때 범주 j 에 속할 조건부 확률

$$p(j|t) = \frac{p(j, t)}{p(t)}, \quad \sum_{j=1}^J p(j|t) = 1$$

이 때, 각 확률은 학습표본에서 아래와 같이 추정할 수 있다.

$$p(j, t) \approx \frac{N_j(t)}{N} \quad (8.1)$$

$$p(t) \approx \frac{N(t)}{N} \quad (8.2)$$

$$p(j|t) \approx \frac{N_j(t)}{N(t)} \quad (8.3)$$

8.3.3 노드 및 트리의 불순도

8.3.3.1 노드의 불순도

CART는 지니 지수(Gini index)를 불순도 함수로 사용한다. 총 J 개의 범주별 객체비율을 p_1, \dots, p_J 라 할 때 ($\sum_{j=1}^J p_j = 1$), 지니 지수는 식 (8.4)와 같다.

$$G(p_1, \dots, p_J) = \sum_{j=1}^J p_j(1 - p_j) = 1 - \sum_{j=1}^J p_j^2 \quad (8.4)$$

노드 t 에서의 범주별 객체비율은 $p(1|t), \dots, p(J|t)$ 으므로, 노드 t 의 불순도는 식 (8.5) 와 같이 산출된다.

$$\begin{aligned} i(t) &= 1 - \sum_{j=1}^J p(j|t)^2 \\ &\approx 1 - \sum_{j=1}^J \left[\frac{N_j(t)}{N(t)} \right]^2 \end{aligned} \quad (8.5)$$

8.3.3.2 트리 불순도

트리 T 의 불순도는 식 (8.6)와 같이 최종노드들의 불순도의 가중평균으로 정의된다.

$$I(T) = \sum_{t \in A(T)} i(t)p(t) \quad (8.6)$$

여기서

$$I(t) = i(t)p(t)$$

라 하면, 다음이 성립한다.

$$I(T) = \sum_{t \in A(T)} I(t)$$

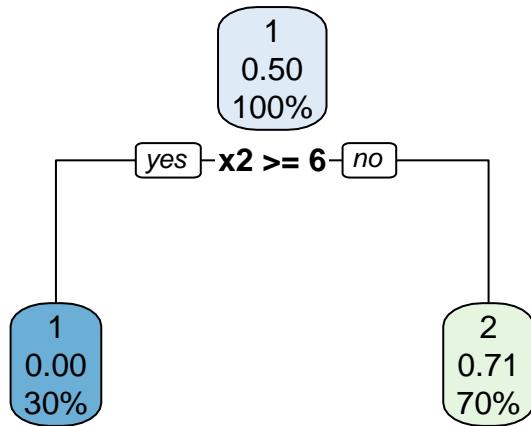


Figure 8.2: 뿌리노드 분지

8.3.4 분지기준

뿌리 노드에서의 분지만을 살펴보기 위해 control parameter *maxdepth*의 값을 1으로 설정한다. 이 경우, CART 알고리즘은 뿌리노드에서의 양 갈래 분지만을 선택한 뒤 종료된다. 아래 스크립트를 이용하여 뿌리노드에서 최적분지된 트리를 얻는다.

```

cart.firstsplit <- rpart(class ~ x1 + x2
  , data = train_df
  , method = "class"
  , parms = list(split = "gini")
  , control = rpart.control(minsplit = 2
    , minbucket = 1
    , maxdepth = 1
    , cp = 0
    , xval = 0
    , maxcompete = 0
  )
)
rpart.plot(cart.firstsplit)
  
```

또한 분지 결과 트리는 Table 8.2와 같이 *frame*이라는 이름의 data frame에 설명된다. 각 행 앞의 번호는 노드 인덱스 *t*를 나타내며, 각 열에 대한 설명은 아래와 같다.

- var: 노드 *t*를 분지하는 데 이용된 변수. 값이 <leaf>인 경우에는 노드 *t*가 최종 노드임을 나타낸다.
- n: 노드 내 객체 수 *N(t)*
- wt: 가중치 적용 후 객체 수 (추후 appendix에서 설명)
- dev: 오분류 객체 수

Table 8.2: 뿌리노드 분자 상세 (frame)

var	n	wt	dev	yval	complexity	ncompete	nsurrogate
1 x2	10	10	5	1	0.6	0	0
2 \<leaf\>	3	3	0	1	0.0	0	0
3 \<leaf\>	7	7	2	2	0.0	0	0

Table 8.3: 노드 내 객체 및 범주 정보 (yval2)

열1	열2	열3	열4	열5	nodeprob
1	1	5	5	0.50	0.50
2	1	3	0	1.00	0.00
3	2	2	5	0.29	0.71

- yval: 노드 t 를 대표하는 범주
- complexity: 노드 t 에서 추가로 분지할 때 감소하는 relative error값; 본 분류트리 예제에서 error는 오분류율이며, 뿌리 노드의 relative error값을 1으로 한다.

또한 *frame*에는 트리 내 각 노드에 속한 객체와 범주에 대한 정보를 나타내는 *yval2*라는 행렬이 Table 8.3와 같이 존재한다. 실제 *yval2*의 열의 개수는 전체 학습 대상 범주 수에 따라 달라지며, 본 예는 이분 분류 트리(범주개수 = 2)에 해당하는 열 구성을 보여준다. 각 행 앞의 번호는 노드 인덱스 t 를 나타내며, 각 열에 대한 설명은 아래와 같다.

- 열1: 노드 t 에서의 최적 추정 범주 j^*
- 열2: 노드 t 내 범주 $class=1$ 객체 수 $N_1(t)$
- 열3: 노드 t 내 범주 $class=2$ 객체 수 $N_2(t)$
- 열4: 노드 t 내 범주 $class=1$ 관측 확률 $p(1|t) \approx \frac{N_1(t)}{N(t)}$
- 열5: 노드 t 내 범주 $class=2$ 관측 확률 $p(2|t) \approx \frac{N_2(t)}{N(t)}$
- nodeprob: 노드 t 확률 $p(t) \approx \frac{N(t)}{N}$

```
## Warning: Setting row names on a tibble is deprecated.
```

위 CART 모델 데이터를 이용하여 트리의 불순도를 계산해보자.

우선 노드 상세 정보 행렬 *yval2*의 x 번째 노드의 불순도($i(t)$)를 계산하는 함수 *rpartNodeImpurity*를 아래와 같이 구현한다.

```
rpartNodeImpurity <- function(x, yval2) {
  node_vec <- yval2[x, ]
  n.columns <- length(node_vec)
  class.prob <- node_vec[((n.columns/2)+1):(n.columns-1)]
  return(1 - sum(class.prob^2))
}
```

CART tree 객체의 각 leaf node에 함수 *rpartNodeImpurity*를 적용하여 노드 불순도 $i(t)$ 를 계산한 뒤, 노드 확률 $p(t)$ 을 이용한 가중합을 통해 트리 불순도 $I(T)$ 를 계산하는 함수 *rpartImpurity*를 아래와 같이 구현한다.

```
rpartImpurity <- function(rpart.obj) {
  leaf.nodes <- which(rpart.obj$frame$var=="<leaf>")
  node.impurity <- sapply(leaf.nodes,
    rpartNodeImpurity,
    yval2 = rpart.obj$frame$yval2)
  node.prob <- rpart.obj$frame$yval2[leaf.nodes, 'nodeprob']
  return(sum(node.prob * node.impurity))
}
```

위 함수를 이용하여 계산한 트리 Figure 8.1의 불순도는 0.29이다.

```
rpartImpurity(cart.firstsplit)
```

```
## [1] 0.2857143
```

분지를 추가할수록 불순도는 감소한다. 분지를 추가하기 위해서는 *maxdepth*라는 control parameter 값을 증가시키면 된다.

- *maxdepth*: 뿐리노드부터 임의의 최종노드에 도달하는 최대 가능 분지 수 (default=30)

maxdepth 파라미터의 값을 1부터 4까지 증가시키며 불순도의 변화를 살펴보자.

```
library(ggplot2)

tree.impurity <- sapply(c(1:4), function(depth) {
  rpart(class ~ x1 + x2
    , data = train_df
    , method = "class"
    , parms = list(split = "gini")
    , control = rpart.control(minsplit = 2
      , minbucket = 1
      , maxdepth = depth
      , cp = 0
      , xval = 0
      , maxcompete = 0)) %>%
  rpartImpurity()
})

tibble(maxdepth=c(1:4), impurity=tree.impurity) %>%
  ggplot(aes(x=maxdepth, y=impurity)) +
  geom_line()
```

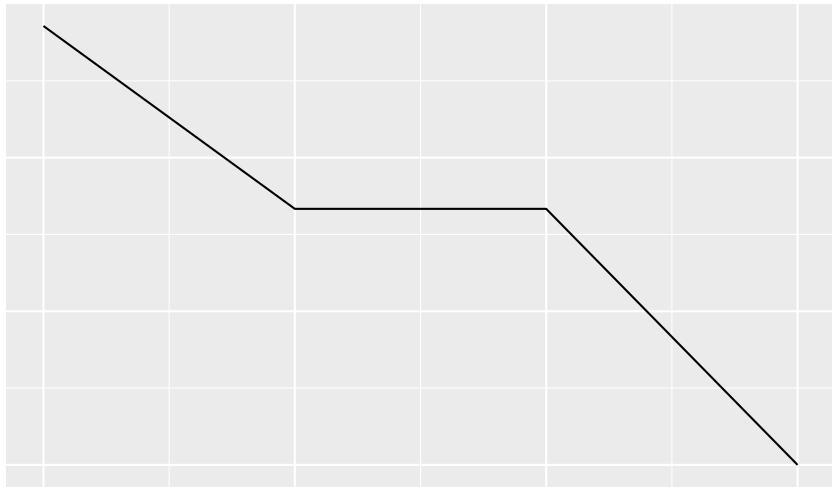


Figure 8.3: 파라미터 maxdepth값에 따른 트리불순도 변화

위 예에서, 트리의 분지가 증가함에 따라 불순도는 0.29, 0.17, 0.17, 0로 감소한다. *maxdepth*값이 3일 때 불순도가 감소하지 않는 이유는, 세 번째 분지 결과가 전체적인 오분류를 감소시키지 않아 *rpart* 함수가 해당 분지를 취소하기 때문이다. 여기에 작용하는 파라미터는 *cp*라는 control parameter이다.

- *cp*: 노드가 분지되기 위한 최소 relative error 감소치 (default = 0.01). 값이 0 일 경우 최대트리를 생성한다.

위 예제에서는 *cp*값을 0으로 설정하여, 해당 분지가 트리 불순도를 감소시킨다 하더라도 전체 트리의 오분류를 감소시키는 데 기여하지 않는다면 시도하지 않도록 하였다.

8.4 가지치기 및 최종 트리 선정

8.4.1 가지치기

앞 장의 최대 트리 그림 8.1은 학습 데이터를 오분류 없이 완벽하게 분류하기 위해 복잡한 분류 구조를 형성하였다. 이러한 복잡한 분류 구조는 학습 데이터가 아닌 새로운 데이터에 대한 분류 정확도를 떨어뜨릴 수 있다. 이는 bias-variance tradeoff라 부르는 현상으로, 비단 분류트리 뿐 아니라 모든 데이터마이닝 방법에 일반적으로 적용된다.

분류 트리는 가지치기라는 방식을 통해, 분류 구조를 단순화함으로써 분류 트리가 새로운 데이터에도 정확한 분류를 제공하기를 추구한다. 가지치기란 트리 내 특정 내부노드를

Table 8.4: 최대 트리 분지 상세 (frame)

	var	n	wt	dev	yval	complexity	ncompete	nsurrogate
1	x2	10	10	5	1	0.6	0	0
2	\<leaf\>	3	3	0	1	0.0	0	0
3	x1	7	7	2	2	0.2	0	0
6	\<leaf\>	1	1	0	1	0.0	0	0
7	x2	6	6	1	2	0.1	0	0
14	x1	2	2	1	1	0.1	0	0
28	\<leaf\>	1	1	0	1	0.0	0	0
29	\<leaf\>	1	1	0	2	0.0	0	0
15	\<leaf\>	4	4	0	2	0.0	0	0

기준으로 그 하위에 발생한 분지를 모두 제거하고, 해당 내부노드를 최종노드로 치환하는 방식이다.

Table 8.4에서 생성 가능한 가지치기는 최종 노드(var값이 <leaf>)가 아닌 모든 노드(1, 3, 7, 14)에서 가능하며, 함수 *snip.rpart*를 이용하여 가지치기 된 트리를 생성할 수 있다. 각 내부 노드에서 가지치기된 트리들은 아래와 같이 얻어진다.

```
internal.node.index <- rownames(cart.est$frame)[which(cart.est$frame$var != '<leaf>')] %>%
  as.numeric()
snipped <- lapply(internal.node.index, function(x){snip.rpart(cart.est, x)})
n.trees <- length(snipped)
par(mfrow=c(2,2))
invisible(lapply(c(1:n.trees), function(x) {
  rpart.plot(snipped[[x]])})
  ))
```

위 각 가지치기 후보 노드의 오분류 비용은 함수 *nodeCost*를 아래와 같이 구현하여 계산할 수 있다.

```
nodeCost <- function(node, tree) {
  node_vec <- tree$frame$yval2[as.character(node) == row.names(tree$frame), ]
  n.columns <- length(node_vec)
  class.prob.max <- max(node_vec[((n.columns/2)+1):(n.columns-1)])
  node.prob <- node_vec[n.columns]
  node.misclassification.cost <- (1-class.prob.max)*node.prob
  return(node.misclassification.cost)
}

tibble(
  pruning_node = internal.node.index,
  node_cost = sapply(internal.node.index, nodeCost, tree=cart.est)
```

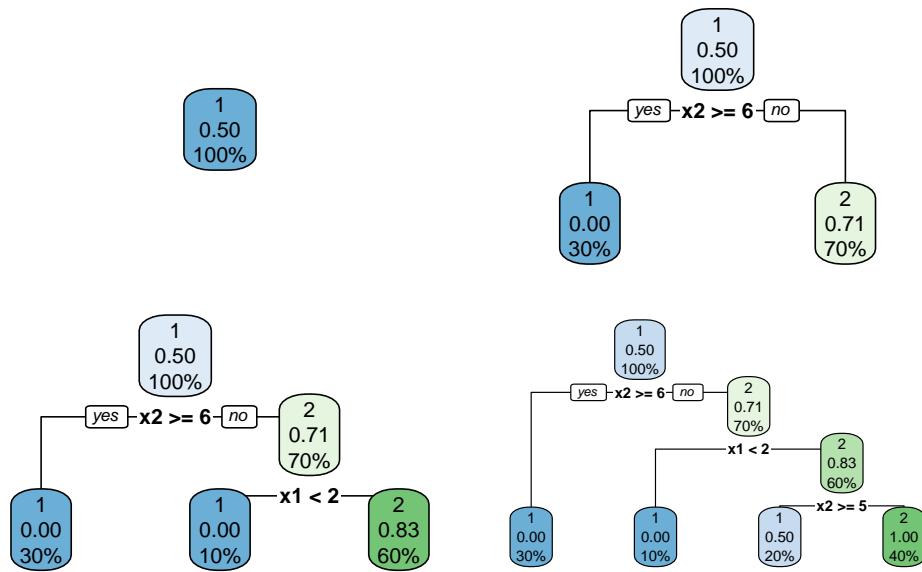


Figure 8.4: 각 내부노드 기준으로 가지치기된 트리

```
) %>%
  knitr::kable()
```

pruning_node	node_cost
1	0.5
3	0.2
7	0.1
14	0.1

각 가지치기 노드에 해당하는 하부 트리의 오분류비용 및 복잡도를 구하기 위해 *subtreeEval*라는 함수를 아래와 같이 구현한다.

```
subtreeEval <- function(node, tree) {
  snipped <- snip.rpart(tree, node)$frame
  leaf.nodes <- setdiff(rownames(tree$frame[tree$frame$var=="<leaf>",]),
                        rownames(snipped)) %>%
    as.numeric()

  tibble(
    pruning_node = node,
    node.cost = nodeCost(node, tree),
    subtree.cost = sapply(leaf.nodes, nodeCost, tree=tree) %>% sum(),
    subtree.size = length(leaf.nodes)
  ) %>%
```

Table 8.5: 내부노드 가지치기 평가 (df.cost)

pruning_node	node.cost	subtree.cost	subtree.size	alpha
1	0.5	0	5	0.12
3	0.2	0	4	0.07
7	0.1	0	3	0.05
14	0.1	0	2	0.10

```
  mutate(alpha = (node.cost - subtree.cost) / (subtree.size - 1))
}
```

각 노드에 대하여 알파값을 다음과 같이 계산할 수 있다.

```
df.cost <- lapply(internal.node.index, subtreeEval, tree=cart.est) %>%
  bind_rows()
```

위 Table 8.5에서 최소 알파값에 해당하는 노드 7에서 가지치기를 한다.

```
pruned.tree.1 <- snip.rpart(cart.est,
                           df.cost$pruning_node[which.min(df.cost$alpha)])
rpart.plot(pruned.tree.1)
```

가지치기로 형성된 트리에서 다시 각 가지치기 노드의 오분류비용, 복잡도 및 알파값을 구한다.

```
df.cost <- rownames(pruned.tree.1$frame)[pruned.tree.1$frame$var!="<leaf>"] %>%
  as.numeric() %>%
  lapply(subtreeEval, tree=pruned.tree.1) %>%
  bind_rows()

knitr::kable(df.cost)
```

pruning_node	node.cost	subtree.cost	subtree.size	alpha
1	0.5	0.1	3	0.2
3	0.2	0.1	2	0.1

위 결과에서 다시 최소 알파값에 해당하는 노드 3에서 가지치기를 하면 아래와 같은 트리가 형성된다.

```
pruned.tree.2 <- snip.rpart(pruned.tree.1,
                           df.cost$pruning_node[which.min(df.cost$alpha)])
rpart.plot(pruned.tree.2)
```

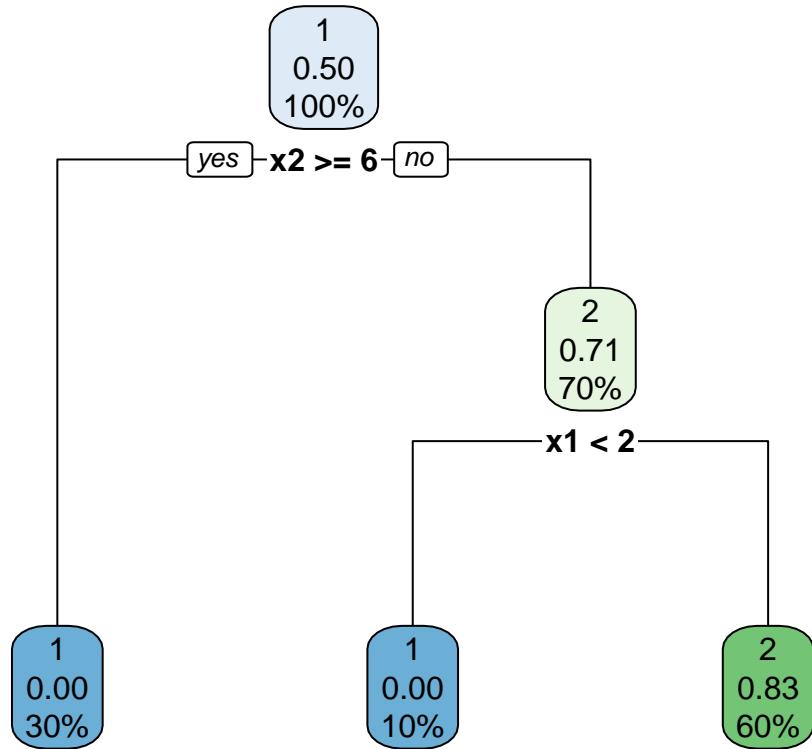


Figure 8.5: 1단계 가지치기 결과

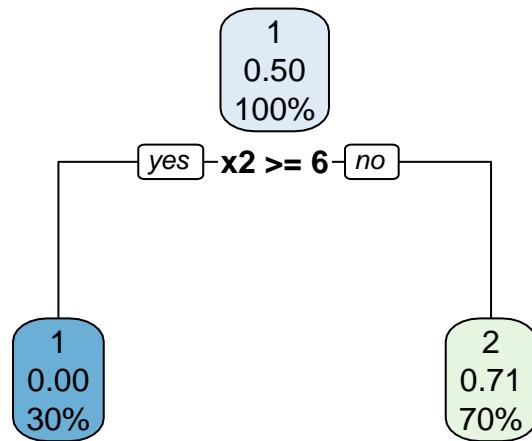


Figure 8.6: 2단계 가지치기 결과

Table 8.6: 테스트 데이터

x1	x2	class
1	5	1
0	5	1
3	4	2
4	3	2
2	7	1
1	4	2

8.4.2 최적 트리의 선정

위 가지치기 과정에서 얻는 가지친 트리들이 최종 트리의 후보가 되며, 이 중 테스트 표본에 대한 오분류율이 가장 작은 트리를 최적 트리로 선정하게 된다.

트리를 학습할 때 사용된 학습데이터 Table 8.1 외에, Table 8.6과 같은 6개의 테스트 데이터가 있다고 하자.

```
test_df <- tibble(
  x1 = c(1,0,3,4,2,1),
  x2 = c(5,5,4,3,7,4),
  class = factor(c(1,1,2,2,1,2), levels=c(1,2))
)
```

테스트 데이터에 위에서 학습된 세 개의 트리, 즉 최대 트리 *cart.est*와 두 개의 가지치기 트리 *pruned.tree.1* & *pruned.tree.2*를 적용하여 각 트리가 각각의 테스트 데이터를 어떻게 분류하는지 살펴보자.

```
test_pred <- test_df %>%
  bind_cols(
    pred_maxtree = predict(cart.est, test_df, type="class"),
    pred_prune1 = predict(pruned.tree.1, test_df, type="class"),
    pred_prune2 = predict(pruned.tree.2, test_df, type="class")
)
```

결과 Table 8.7에서 최대트리가 오분류한 테스트 표본은 1개, 첫번째 가지치기 트리가 오분류한 테스트 표본은 1개, 그리고 두 번째 가지치기 트리가 오분류한 테스트 표본은 2개이다.

위 결과를 토대로, 최적의 트리를 선정하는 과정은 아래와 같다.

- 각각의 트리에 의해 오분류된 테스트 표본의 개수를 전체 테스트 표본의 개수로 나누어 오분류율 R^{ts} 를 구한다.

Table 8.7: 테스트 데이터에 대한 예측 결과

x1	x2	class	pred_maxtree	pred_prune1	pred_prune2
1	5	1	1	1	2
0	5	1	1	1	2
3	4	2	2	2	2
4	3	2	2	2	2
2	7	1	1	1	1
1	4	2	1	1	2

Table 8.8: 분류 성능

트리	오분류율(\$R^{ts}\$)	표준편차(\$SE\$)	척도(\$R^{ts} + SE\$)
cart.est	0.17	0.15	0.32
pruned.tree.1	0.17	0.15	0.32
pruned.tree.2	0.33	0.19	0.53

2. 테스트 표본 수를 n_{test} 라 할 때, 오분류의 표준편차를 아래와 같이 계산한다.

$$SE = \sqrt{\frac{R^{ts}(1 - R^{ts})}{n_{test}}}$$

3. 1에서 구한 오분류율에 2에서 구한 표준편차를 더하여 $R^{ts} + SE$ 를 각 트리의 평가척도로 계산한다. 후보 트리들 중 해당 평가척도가 가장 작은 트리를 최종 트리로 선정한다.

```
test.summary <- test_pred %>%
  summarize(n.test = n(),
            cart.est = sum(pred_maxtree != class) / n.test,
            pruned.tree.1 = sum(pred_prune1 != class) / n.test,
            pruned.tree.2 = sum(pred_prune2 != class) / n.test) %>%
  gather("tree", "R.ts", -n.test) %>%
  mutate(SE = sqrt((R.ts * (1 - R.ts)) / n.test),
        score = R.ts + SE) %>%
  select(-n.test)
```

위 결과, 최적 트리는 최대 트리 혹은 첫 번째 가지치기 트리가 된다.

위 절차를 임의의 데이터에 대해 수행하는 함수를 구현해보자.

```
rpart_learn <- function(formula, train_df, test_df) {
  # 최대 트리 생성
  max_tree <- rpart(formula
```

```

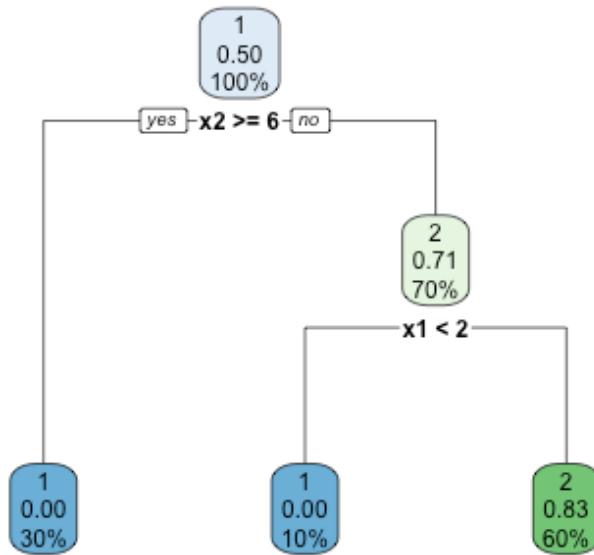
        , data = train_df
        , method = "class"
        , parms = list(split = "gini")
        , control = rpart.control(minsplit = 2
                                  , minbucket = 1
                                  , cp = 0
                                  , xval = 0
                                  , maxcompete = 0
                                )
      )

# 가지치기
curr_tree <- list()
k <- 1
curr_tree[[k]] <- max_tree
while(dim(curr_tree[[k]]$frame)[1] > 1) {
  internal.node.index <- rownames(curr_tree[[k]]$frame)[which(curr_tree[[k]]$frame$var != '<leaf'
    as.numeric())
  df.cost <- lapply(internal.node.index, subtreeEval, tree=curr_tree[[k]]) %>% bind_rows()
  curr_tree[[k + 1]] <- snip.rpart(curr_tree[[k]],
    df.cost$pruning_node[which.min(df.cost$alpha)])
  k <- k + 1
}

# 최적 가지치기 트리 선정
n.test <- dim(test_df)[1]
R.ts <- lapply(curr_tree, function(x) {
  sum(predict(x, test_df, type="class") != test_df$class) / n.test
}) %>% unlist()
score <- R.ts + sqrt((R.ts*(1 - R.ts))/n.test)
return(curr_tree[[max(which(score == min(score)))]])
}

optimal_tree <- rpart_learn(class ~ x1 + x2, train_df, test_df)
rpart.plot(optimal_tree)

```



8.5 R패키지 내 분류 트리 방법

앞 장에서는 *rpart*의 결과를 이용하여 교재 8.2 - 8.3장의 예제를 재현해보았다. 실제로 *rpart* 내부의 기본 트리 방법은 교재의 예제와는 다소 다른 부분이 있다. 이 장에서는 실제 *rpart* 패키지의 분류 트리 방법에 대해 알아본다.

8.5.1 트리 확장

트리 내 임의의 노드 t 에 대한 불순도는 아래와 같이 정의된다.

$$i(t) = \sum_{j=1}^J f(p(j|t))$$

여기에서 $p(j|t)$ 는 노드 t 내 전체 샘플 $N(t)$ 중 범주 j 의 샘플 $N_j(t)$ 의 비율로 추정된다.

$$p(j|t) \approx \frac{N_j(t)}{N(t)}$$

또한 함수 f 는 concave 함수로, $f(0) = f(1) = 0$ 의 조건을 만족시켜야 한다. *rpart*에서 설정할 수 있는 함수 f 의 종류에 대해서는 아래에서 좀 더 자세히 살펴보기로 한다.

트리 내 임의의 노드 t 가 분지규칙 s 에 따라 두 개의 노드 t_L 과 t_R 로 분지된다고 할 때, 불순도의 감소량은 아래와 같이 계산된다.

$$\Delta I(s, t) = I(t) - I(t_L) - I(t_R) \quad (8.7)$$

$$= p(t)i(t) - p(t_L)i(t_L) - p(t_R)i(t_R) \quad (8.8)$$

*rpart*는 위 $\Delta I(s, t)$ 값이 최대가 되는 분지 기준 s^* 를 찾아 노드 t 를 분지하여 트리를 확장하고, 확장된 트리의 최종 노드에서 다시 최적 분지를 찾는 과정을 반복한다.

8.5.1.1 분지 함수

함수 *rpart* 사용 시 *parms* 파라미터에 *split* 값으로 분지 방법을 설정할 수 있다.

1. Gini index (*parms=list(split='gini')*) 교재의 예제에 사용된 방법으로, 우선 아래와 같은 함수 f 를 사용한다.

$$f(p) = p(1 - p)$$

2. information index (*parms=list(split='information')*) 교재에 엔트로피 지수 (Entropy index)로 설명된 지수로, 아래와 같은 함수를 사용한다.

$$f(p) = -p \log(p)$$

3. user-defined function 사용자가 임의로 함수를 정의하여 사용할 수 있다. 본 장에서는 자세한 설명은 생략한다.

8.5.2 가지치기

임의의 노드 t 에 대한 위험도(오분류 비용의 기대치)는 아래와 같이 계산된다.

$$r(t) = \sum_{j \neq \tau(t)} p(j|t)C(\tau(t)|j)$$

여기에서 함수 $C(i|j)$ 는 범주 j 에 속하는 객체를 범주 i 로 분류할 때의 오분류 비용이며, $\tau(t)$ 는 노드 t 내의 오분류 비용을 최소화하도록 노드 t 에 지정된 범주값이다.

*rpart*의 오분류 비용 $C(i|j)$ 의 기본값은

$$C(i|j) = \begin{cases} 1, & \text{if } i \neq j \\ 0, & \text{if } i = j \end{cases}$$

으로 설정되어 있으며, *parms* 파라미터에 *loss* 값으로 오분류 비용 $C(i|j)$ 를 재설정할 수 있다. 본 장에서는 기본값을 사용하도록 하자.

$A(T)$ 를 트리 T 의 최종 노드의 집합이라 정의하고, 트리의 최종 노드의 개수를 $|T|$ 라 할 때, 트리 T 의 위험도 $R(T)$ 는 아래와 같이 정의된다.

$$R(T) = \sum_{t \in A(T)} p(t)r(t)$$

복잡도 계수(complexity parameter) $\alpha \in [0, \infty)$ 를 이용하여, 트리의 비용-복잡도 척도를 다음과 같이 정의한다.

$$R_\alpha(T) = R(T) + \alpha|T|$$

이 때, 임의의 계수 α 에 대해 비용 $R_\alpha(T)$ 가 최소가 되게하는 가지치기 트리를 T_α 라 하면, 아래와 같은 관계들이 성립한다.

- T_0 : 최대 트리
- T_∞ : 뿌리 노드 트리 (분지 없음)
- $\alpha > \beta$ 일 때, T_α 는 T_β 와 동일하거나 혹은 T_β 에서 가지치기된 트리이다.

8.5.3 파라미터값 결정

함수 `rpart`을 사용할 때 여러가지 사용자 정의 파라미터값을 설정할 수 있으며, 그 파라미터 값에 따라 생성되는 트리의 결과가 달라진다. 대표적인 파라미터 값으로는 아래와 같은 것들이 있다.

- `minsplit`: 분지를 시도하기 위해 필요한 노드 내 최소 관측액체 수 (default=20)
- `cp`: 노드가 분지되기 위한 최소 relative error 감소치 (default = 0.01). 값이 0 일 경우 최대트리를 생성한다.
- `maxdepth`: 뿌리노드부터 임의의 최종노드에 도달하는 최대 가능 분지 수 (default=30)

Chapter 9

서포트 벡터 머신

9.1 개요

서포트 벡터 머신(support vector machine; 이하 SVM)은 기본적으로 두 범주를 갖는 객체들을 분류하는 방법이다. 물론 세 범주 이상의 경우로 확장이 가능하다.

9.2 필요 R package 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
e1071	1.7-1
Matrix	1.2-17
quadprog	1.5-7

9.3 선형 SVM - 분리 가능 경우

9.3.1 기본 R 스크립트

```
train_df <- tibble(
  x1 = c(5, 4, 7, 8, 3, 2, 6, 9, 5),
  x2 = c(7, 3, 8, 6, 6, 5, 6, 6, 4),
  class = c(1, -1, 1, 1, -1, -1, 1, 1, -1)
)
```

Table 9.1: 선형분리가능 학습표본 데이터

x1	x2	class
5	7	1
4	3	-1
7	8	1
8	6	1
3	6	-1
2	5	-1
6	6	1
9	6	1
5	4	-1

```
knitr::kable(train_df, booktabs = TRUE,
             align = c('r', 'r', 'r'),
             caption = '선형분리가능 학습표본 데이터')
```

Table 9.1와 같이 두 독립변수 x_1, x_2 와 이분형 종속변수 $class$ 의 관측값으로 이루어진 9 개의 학습표본을 $train_df$ 라는 data frame에 저장한다.

```
library(e1071)
svm_model <- svm(as.factor(class) ~ x1 + x2, data = train_df, kernel = "linear", scale = FALSE)
plot(svm_model, data = train_df, formula = x2 ~ x1, grid = 200)
```

그림 9.1에서 각 객체의 기호는 서포트 벡터 여부("X"이면 서포트 벡터), 각 객체의 색상은 범주값(검정 = -1, 빨강 = 1)을 나타내며, 분리 하이퍼플레인은 아래와 같다.

$$0.6666667x_1 + 0.6666667x_2 = 7$$

9.3.2 기호 정의

본 장에서 사용될 수학적 기호는 아래와 같다.

- $\mathbf{x} \in \mathbb{R}^p$: p차원 변수벡터
- $y \in \{-1, 1\}$: 범주
- N : 객체 수
- (\mathbf{x}_i, y_i) : i 번째 객체의 변수벡터와 범주값

9.3.3 최적 하이퍼플레인

선형 SVM은 주어진 객체들의 두 범주를 완벽하게 분리하는 하이퍼플레인 중 각 범주의 서포트 벡터들로부터의 거리가 최대가 되는 하이퍼플레인을 찾는 문제로 귀착된다.

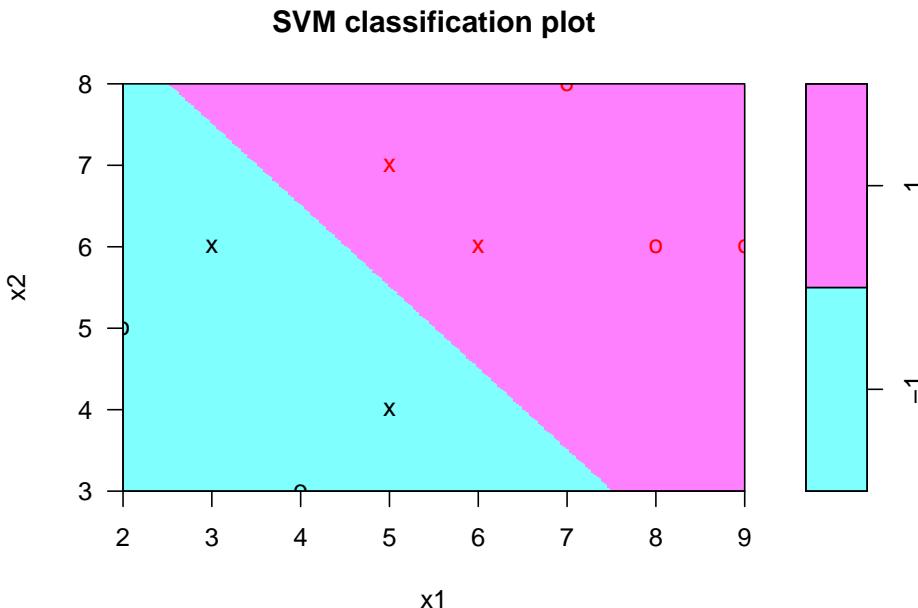


Figure 9.1: 선형 SVM 분리 하이퍼플레인

우선 아래와 같이 하이퍼플레인을 정의한다.

$$\mathbf{w}^\top \mathbf{x} + b = 0 \quad (9.1)$$

여기서 $\mathbf{w} \in \mathbb{R}^p$ 와 $b \in \mathbb{R}$ 이 하이퍼플레인의 계수이다.

범주값이 1인 객체들 중 하이퍼플레인에서 가장 가까운 객체에 대해 다음과 같은 조건이 만족한다고 가정하자.

$$H_1 : \mathbf{w}^\top \mathbf{x} + b = 1$$

또한 범주값이 -1인 객체들 중 하이퍼플레인에서 가장 가까운 객체에 대해 다음과 같은 조건이 만족한다고 가정하자.

$$H_2 : \mathbf{w}^\top \mathbf{x} + b = -1$$

이 때 두 하이퍼플레인 H_1 과 H_2 간의 거리(margin)는 $2/\|\mathbf{w}\|$ 이다. 선형 SVM은 아래와 같이 H_1 과 H_2 간의 거리를 최대로 하는 최적화 문제가 된다.

$$\begin{aligned} \max & \frac{2}{\mathbf{w}^\top \mathbf{w}} \\ \text{s.t.} & \mathbf{w}^\top \mathbf{x}_i + b \geq 1 \text{ for } y_i = 1 \\ & \mathbf{w}^\top \mathbf{x}_i + b \leq -1 \text{ for } y_i = -1 \end{aligned}$$

이를 간략히 정리하면

$$\begin{aligned} \min & \frac{\mathbf{w}^\top \mathbf{w}}{2} \\ \text{s.t.} & y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 \end{aligned}$$

과 같이 정리할 수 있으며, 각 객체 i 에 대한 제약조건에 라그랑지 계수(Lagrange multiplier) $\alpha_i \geq 0$ 를 도입하여 라그랑지 함수를 유도하면 식 (9.2)와 같은 최적화 문제가 된다. 이를 원문제(primal problem)라 하자.

$$\begin{aligned} \min & L_P = \frac{1}{2} \mathbf{w}^\top \mathbf{w} + \sum_{i=1}^N \alpha_i [y_i (\mathbf{w}^\top \mathbf{x}_i + b) - 1] \\ \text{s.t.} & \alpha_i \geq 0, \quad i = 1, \dots, N \end{aligned} \tag{9.2}$$

원문제 식 (9.2)에 대한 을프쌍대문제(Wolfe dual problem)는 아래 식 (9.3)과 같이 도출된다. 보다 자세한 내용은 교재(전치혁, 2012) 참고.

$$\begin{aligned} \max & L_D = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \\ \text{s.t.} & \sum_{i=1}^N \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, \dots, N \end{aligned} \tag{9.3}$$

식 (9.3)은 이차계획(quadratic programming) 문제로, 각종 소프트웨어와 알고리즘을 이용하여 구할 수 있다. 본 장에서는 quadprog 패키지를 이용하여 해를 구하기로 한다. 이는 실제로 e1071의 `svm` 함수 호출 시 사용하는 방법은 아니며, 실제 `svm` 함수가 호출하는 알고리즘은 다음 장에서 다시 설명하기로 한다.

quadprog의 solve.QP 함수는 아래와 같은 형태로 formulation된 문제(Goldfarb and Idnani, 1983)에 대한 최적해를 구한다.

$$\begin{aligned} \min \quad & -\mathbf{d}^\top \boldsymbol{\alpha} + \frac{1}{2} \boldsymbol{\alpha}^\top \mathbf{D} \boldsymbol{\alpha} \\ \text{s.t. } & \mathbf{A}^\top \boldsymbol{\alpha} \geq \mathbf{b}_0 \end{aligned} \quad (9.4)$$

식 (9.4)과 식 (9.3)이 동일한 문제를 나타내도록 아래와 같이 목적함수에 필요한 벡터 및 행렬을 정의한다.

$$\begin{aligned} \mathbf{d} &= \mathbf{1}_{N \times 1} \\ \mathbf{D} &= \mathbf{y} \mathbf{y}^\top \mathbf{X} \mathbf{X}^\top \end{aligned}$$

where

$$\begin{aligned} \mathbf{y} &= [y_1 \quad y_2 \quad \cdots \quad y_N]^\top \\ \mathbf{X} &= [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \cdots \quad \mathbf{x}_N]^\top \end{aligned}$$

```
N <- dim(train_df)[1]
X <- train_df[c('x1', 'x2')] %>% as.matrix()
y <- train_df[['class']] %>% as.numeric()

d <- rep(1, N)
D <- (y %*% t(y)) * (X %*% t(X))
```

여기에서 행렬 \mathbf{D} 의 determinant 값은 0으로, Goldfarb and Idnani (1983) 가 가정하는 symmetric positive definite matrix 조건에 위배되어 solve.QP 함수 실행 시 오류가 발생한다. 이를 방지하기 위해 아래 예에서는 Matrix 패키지의 nearPD함수를 이용하여 행렬 \mathbf{D} 와 근사한 symmetric positive definite matrix를 아래와 같이 찾는다.

```
D_pd <- Matrix:::nearPD(D, doSym = T)$mat %>% as.matrix()
```

식 (9.4)의 제약식은 모두 inequality 형태로, 식 (9.3)의 equality constraint $\sum_{i=1}^N \alpha_i y_i = 0$ 를 표현하기 위해서 두 개의 제약식 $\sum_{i=1}^N \alpha_i y_i \geq 0$ 와 $\sum_{i=1}^N -\alpha_i y_i \geq 0$ 를 생성한다.

$$\mathbf{A}^\top = \begin{bmatrix} y_1 & y_2 & \cdots & y_N \\ -y_1 & -y_2 & \cdots & -y_N \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}, \mathbf{b}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Table 9.2: 이차계획문제의 최적해

variable	solution
alpha_1	0.2234
alpha_2	0.0000
alpha_3	0.0000
alpha_4	0.0000
alpha_5	0.2228
alpha_6	0.0000
alpha_7	0.2210
alpha_8	0.0000
alpha_9	0.2216

```
A <- cbind(
  y,
  -y,
  diag(N)
)
b_zero <- rep(0, 2 + N)
```

이제 위에서 구한 행렬과 벡터들을 `solve.QP` 함수에 입력하여 최적해를 구한다.

```
res <- quadprog::solve.QP(D_pd, d, A, b_zero)
alpha_sol <- res$solution
obj_val <- -res$value
```

표 9.2의 결과는 교재(전치혁, 2012)에 나타난 최적해와는 다소 차이가 있으나, 결과적으로 목적함수값은 0.4444로 동일하다.

위의 과정으로 최적해 α_i^* 를 구한 뒤, 아래와 같이 분리 하이퍼플레인의 계수를 결정할 수 있다.

$$\mathbf{w} = \sum_{i=1}^N \alpha_i^* y_i \mathbf{x}_i$$

$$b = \sum_{i:\alpha_i^*>0} \frac{1 - y_i \mathbf{w}^\top \mathbf{x}_i}{y_i} \left/ \sum_{i:\alpha_i^*>0} 1 \right.$$

```
w <- colSums(alpha_sol * y * X)
print(w)
```

```

##           x1          x2
## 0.6666658 0.6666657

sv_ind <- which(round(alpha_sol, digits = 4) > 0)
b <- mean((1 - y[sv_ind] * (X[sv_ind, ] %*% w)) / y[sv_ind])
print(b)

## [1] -6.99999

```

위 결과와 같이, 분리 하이퍼플레인은 교재와 동일하게 얻어진다.

9.4 선형 SVM - 분리 불가능 경우

본 장에서는 학습표본 내의 두 범주가 어떠한 선형 하이퍼플레인으로도 완전하게 분리되지 않아 식 (9.2)이 해를 갖지 못하는 경우에 대한 문제를 다룬다.

9.4.1 기본 R 스크립트

앞 장에서 사용한 학습표본에 아래와 같이 하나의 객체를 추가하여 전체 학습표본이 선형 하이퍼플레인으로 분리될 수 없도록 하자.

```

inseparable_train_df <- bind_rows(train_df,
                                      tibble(x1 = 7, x2 = 6, class = -1))

knitr::kable(inseparable_train_df, booktabs = TRUE,
              align = c('r', 'r', 'r'),
              caption = '선형분리불가능 학습표본 데이터')

library(e1071)
svm_model <- svm(as.factor(class) ~ x1 + x2, data = inseparable_train_df,
                  kernel = "linear", cost = 1, scale = FALSE)
plot(svm_model, data = inseparable_train_df, formula = x2 ~ x1, grid = 200)

```

Figure 9.2에서 보이듯, 하나의 검정 객체(범주 = -1)가 범주 1로 분류되는 영역에 존재하여 오분류가 발생한다. 이처럼 선형 하이퍼플레인으로 두 범주 학습표본의 분리가 불가능한 경우, 오분류 학습표본에 대한 페널티를 적용하여 최적 분리 하이퍼플레인을 도출하게 된다. 위 예에서의 최적 하이퍼플레인은 아래와 같다.

$$0.6x_1 + 0.8x_2 = 7.6$$

Table 9.3: 선형분리불가능 학습표본 데이터

x1	x2	class
5	7	1
4	3	-1
7	8	1
8	6	1
3	6	-1
2	5	-1
6	6	1
9	6	1
5	4	-1
7	6	-1

9.4.2 최적 하이퍼플레인

여유변수(slack variable) ξ_i 를 각 학습객체 $i = 1, \dots, N$ 에 대해 아래와 같이 정의한다.

$$\xi_i = \max \{0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\}$$

이는 객체가 자신의 범주의 서포트 벡터를 지나는 하이퍼플레인(범주 1인 경우 H_1 , 범주 -1인 경우 H_2)으로부터 다른 범주 방향으로 떨어진 거리를 나타낸다. 이 여유변수 ξ_i 에 단위당 페널티 단가 C 를 부여하여 아래와 같은 최적화 문제를 정의한다.

$$\begin{aligned} \min \quad & \frac{\mathbf{w}^\top \mathbf{w}}{2} + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \dots, N \\ & \xi \geq 0, \quad i = 1, \dots, N \end{aligned}$$

이에 대한 올프쌍대문제를 앞 9.3.3장과 같은 과정으로 도출하면 아래 식 (9.5)와 같다.

$$\begin{aligned} \max \quad & L_D = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \\ \text{s.t.} \quad & \sum_{i=1}^N \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, N \end{aligned} \tag{9.5}$$

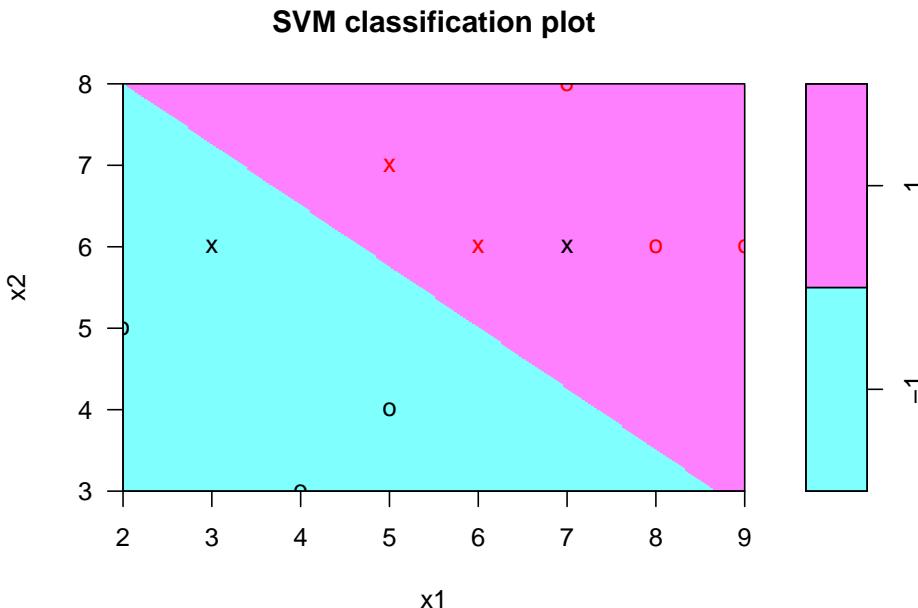


Figure 9.2: 선형 SVM 분리 불가능 경우의 하이퍼플레인

이는 분리 가능 경우의 식 (9.3)에 변수 α_i 에 대한 상한값 C 의 제약이 추가된 문제로, 이는 e1071 패키지의 `svm` 함수가 기본 방법으로 사용하는 LIBSVM 라이브러리(Chang and Lin, 2011)의 C -support vector classification(C -SVC)이 사용하는 문제식이며, LIBSVM 라이브러리는 특정 알고리즘(Fan et al., 2005)을 이용하여 해를 제공한다.

아래 `svm` 함수의 입력 변수에서 `type = "C-classification"`은 식 (9.3)을 최적화하겠다는 것을 나타내며, `cost = 1`은 페널티 단가 C 의 값을 1로 설정하겠다는 것을 나타낸다.

```
svm_model <- svm(as.factor(class) ~ x1 + x2, data = inseparable_train_df,
                  kernel = "linear", scale = FALSE,
                  type = "C-classification", cost = 1)
```

위 결과 모델 객체 `svm_model`의 원소 중 `index`는 학습표본 중 서포트 벡터에 해당하는 인덱스 i 를 나타내며, `coefs`는 각 서포트 벡터의 $\alpha_i y_i$ 값을 나타낸다. 따라서, `coefs`를 각 서포트 벡터의 범주값 y_i 로 나누면 식 (9.5)의 최적해를 아래와 같이 볼 수 있다.

```
N <- dim(inseparable_train_df)[1]
X <- inseparable_train_df[c('x1', 'x2')] %>% as.matrix()
y <- inseparable_train_df[['class']] %>% as.numeric()

sv_ind <- svm_model$index
alpha_sol <- vector("numeric", N)
alpha_sol[sv_ind] <- drop(svm_model$coefs[, 1]) / y[sv_ind]
```

Table 9.4: 이차계획문제의 최적해: 선형 분리 불가능 경우

variable	solution
alpha_1	0.8
alpha_2	0.0
alpha_3	0.0
alpha_4	0.0
alpha_5	0.8
alpha_6	0.0
alpha_7	1.0
alpha_8	0.0
alpha_9	0.0
alpha_10	1.0

하이퍼플레인의 계수 \mathbf{w} 는 분리 가능한 경우와 동일하게 구할 수 있다.

$$\mathbf{w} = \sum_{i=1}^N \alpha_i^* y_i \mathbf{x}_i$$

```
w <- colSums(alpha_sol * y * X)
print(w)
```

```
##   x1   x2
## 0.6  0.8
```

상수 b 는 아래와 같이 $0 < \alpha_i^* < C$ 인 객체들을 이용해 산출할 수 있다.

$$b = \sum_{i:0<\alpha_i^*<C} \frac{1 - y_i \mathbf{w}^\top \mathbf{x}_i}{y_i} \left/ \sum_{i:0<\alpha_i^*<C} 1 \right.$$

```
ind <- sv_ind[alpha_sol[sv_ind] < svm_model$cost]
b <- mean((1 - y[ind] * (X[ind, ] %*% w)) / y[ind])
print(b)
```

```
## [1] -7.6
```

위와 같은 하이퍼플레인의 계수 \mathbf{w} 와 상수 b 값은 `svm` 객체에 원소들을 이용하여 보다 쉽게 확인할 수 있다.

Table 9.5: 페널티 단가 C 에 따른 하이퍼플레인 계수 및 결과

페널티 단가 \$C\$	$\$(w_1, w_2)$	\$b\$	서포트 벡터 인덱스	오분류 객체
1	0.6, 0.8	-7.6	1, 7, 5, 10	10
5	0.4, 1.2	-9.4	1, 4, 7, 5, 10	10
100	0.4, 1.2	-9.4	1, 4, 7, 5, 10	10

```
w <- t(svm_model$coefs) %*% svm_model$SV
print(w)
```

```
##      x1   x2
## [1,] 0.6 0.8
```

```
b <- -svm_model$rho
print(b)
```

```
## [1] -7.6
```

선형 하이퍼플레인으로 분리 불가능한 경우, 페널티 단가 C 의 값에 따라 도출되는 분리 하이퍼플레인이 달라진다. C 의 값이 1, 5, 100일 때의 하이퍼플레인을 비교해보자.

```
svm_models <- lapply(c(1, 5, 100), function(C)
  svm(as.factor(class) ~ x1 + x2, data = inseparable_train_df,
    kernel = "linear", scale = FALSE,
    type = "C-classification", cost = C))

getHyperplane <- function(model) {
  list(C = model$cost,
    w = paste(round(t(model$coefs) %*% model$SV, digits = 2), collapse = ", "),
    b = -round(model$rho, digits = 2),
    sv = paste(model$index, collapse = ", "),
    misclassified = paste(which(model$fitted != as.factor(inseparable_train_df$class)), collapse = ", "))
}

svm_summary <- lapply(svm_models, getHyperplane) %>% bind_rows()
```

Table 9.5에서 보이는 바와 같이, 페널티 단가 C 의 값이 1과 5일 때 분리 하이퍼플레인이 변하는 것을 볼 수 있다. C 값이 5와 100일 때의 분리 하이퍼플레인은 거의 동일하다.

```
plot(svm_models[[2]], data = inseparable_train_df, formula = x2 ~ x1, grid = 200)
```

Figure 9.3의 하이퍼플레인($C = 5$ 인 경우)은 Figure 9.2의 하이퍼플레인($C = 1$ 인 경우)보다 오분류 객체에 가깝게 위치함을 확인할 수 있다.

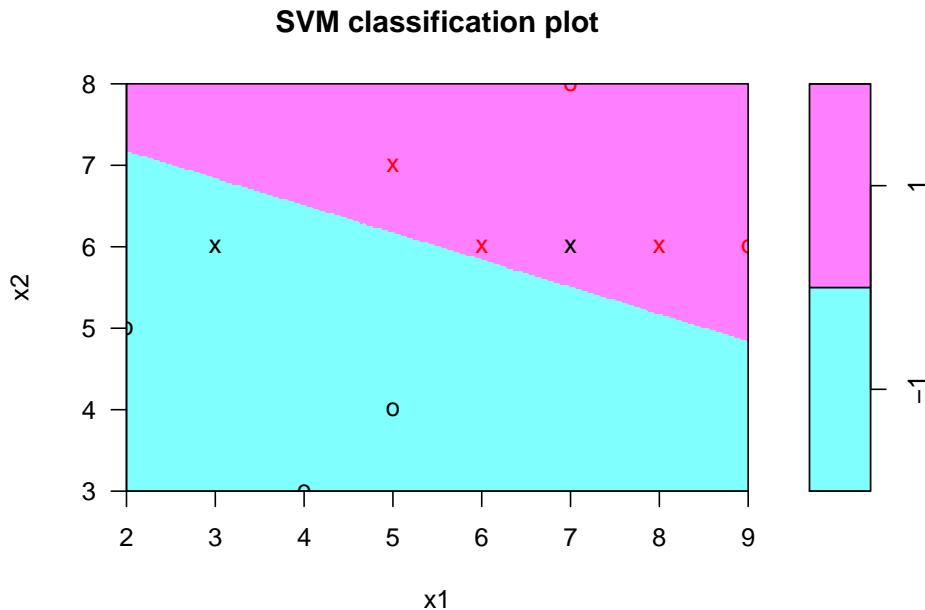


Figure 9.3: 선형 SVM 분리 불가능 경우의 하이퍼플레인 ($C = 5$)

9.5 비선형 SVM

본 장에서는 선형으로 분리 성능이 좋지 않은 경우에 대해 원 입력변수에 대해 비선형인 하이퍼플레인을 찾는 문제를 다룬다. 이는 원 입력변수에 대해 비선형인 기저함수 공간으로 객체를 이동시킨 후 해당 공간에서 선형 분리 하이퍼플레인을 찾는 과정이다.

9.5.1 기본 R 스크립트

```
nonlinear_train_df <- tibble(
  x1 = c(5, 4, 7, 8, 3, 2, 6, 9, 5),
  x2 = c(7, 3, 8, 6, 6, 5, 6, 6, 4),
  class = c(1, -1, -1, -1, 1, 1, 1, -1, -1)
)

knitr::kable(nonlinear_train_df, booktabs = TRUE,
             align = c('r', 'r', 'r'),
             caption = '비선형 SVM 학습표본 데이터')
```

Table 9.6: 비선형 SVM 학습표본 데이터

x1	x2	class
5	7	1
4	3	-1
7	8	-1
8	6	-1
3	6	1
2	5	1
6	6	1
9	6	-1
5	4	-1

```
library(e1071)
svm_model <- svm(as.factor(class) ~ x1 + x2, data = nonlinear_train_df,
                  kernel = "polynomial", coef0 = 1, gamma = 1, degree = 2,
                  cost = 5, scale = FALSE)
plot(svm_model, data = nonlinear_train_df, formula = x2 ~ x1, grid = 200)
```

9.5.2 최적 하이퍼플레인

식 (9.1)을 일반화한 다음과 같은 하이퍼플레인을 고려하자.

$$f(\mathbf{x}) = \Phi(\mathbf{x})^\top \mathbf{w} + b \quad (9.6)$$

여기서 벡터함수 $\Phi : \mathbb{R}^p \rightarrow \mathbb{R}^m$ 은 \mathbf{x} 에 대한 새로운 특징(feature)을 추출하는 변환함수라 할 수 있는데, 통상 추출되는 특징의 차원 m 이 원 변수 \mathbf{x} 의 차원 p 보다 높다. 이를 \mathbf{x} 의 기저함수(basis function)라 부르며, 하이퍼플레인 계수 또한 m 차원의 벡터가 된다 ($\mathbf{w} \in \mathbb{R}^m$). 이 때, 비선형 SVM 문제는 선형 SVM 문제 식 (9.5)에서 변수를 기저변수로 치환한 형태로 아래 식 (9.7)과 같이 나타낼 수 있다.

$$\begin{aligned} \max L_D &= \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j) \\ \text{s.t.} \\ \sum_{i=1}^N \alpha_i y_i &= 0 \\ 0 \leq \alpha_i &\leq C, \quad i = 1, \dots, N \end{aligned} \quad (9.7)$$

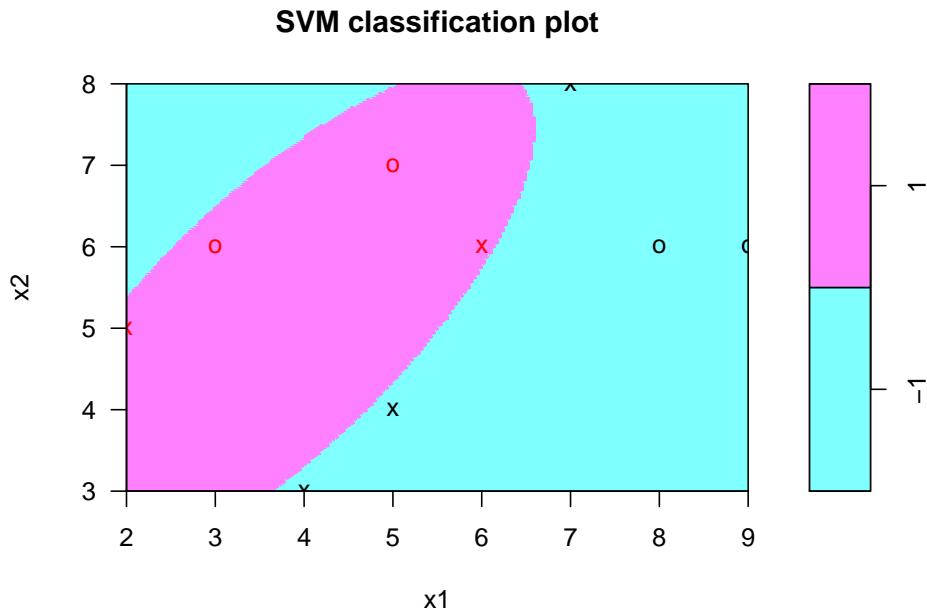


Figure 9.4: 비선형 SVM 하이퍼플레인

식 (9.7)의 목적함수에서 기저함수의 내적 $\Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j)$ 을 아래와 같이 커널함수(kernel function)로 나타낼 수 있으며, 이는 두 객체 $\mathbf{x}_i, \mathbf{x}_j$ 간의 일종의 유사성 척도(similarity measure)로 해석될 수 있다.

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}_j)$$

널리 사용되는 커널함수로는 아래와 같은 함수들이 있다.

$$\begin{aligned} \text{Gaussian RBF: } K(\mathbf{x}_i, \mathbf{x}_j) &= \exp\left(\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right) \\ r\text{-th order polynomial: } K(\mathbf{x}_i, \mathbf{x}_j) &= (\mathbf{x}_i^\top \mathbf{x}_j + 1)^r \\ \text{Sigmoid: } K(\mathbf{x}_i, \mathbf{x}_j) &= \tanh(\kappa \mathbf{x}_i^\top \mathbf{x}_j - \delta) \end{aligned}$$

커널함수를 이용하여 분리 하이퍼플레인을 찾기 위한 식을 아래와 같이 나타낸다.

$$\begin{aligned}
 \max \quad & L_D = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j k_{ij} \\
 \text{s.t.} \quad & \sum_{i=1}^N \alpha_i y_i = 0 \\
 & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, N
 \end{aligned} \tag{9.8}$$

이 때 k_{ij} 는 $K(\mathbf{x}_i, \mathbf{x}_j)$ 를 나타낸다. 식 (9.8)의 최적해 α^* 는 선형 SVM과 마찬가지로 이차계획(quadratic programming) 소프트웨어/알고리즘을 이용하여 구할 수 있다.

Table 9.6의 학습데이터에 대해 e1071 패키지의 `svm` 함수를 이용하여 이차 다항 커널에 기반한 분리 하이퍼플레인을 구해보자. `svm` 함수에 파라미터값 `kernel = "polynomial"`를 설정함으로써 다항 커널을 사용할 수 있다. `svm` 함수의 다항 커널은 위에서 설명된 것보다 일반화된 형태로 아래와 같이 정의된다.

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^\top \mathbf{x}_j + \beta_0)^r$$

위 커널함수의 파라미터 γ, β_0, r 은 `svm` 함수에 파라미터 `gamma`, `coef0`, `degree`로 각각 정의된다. 따라서 이차 커널

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + 1)^2$$

에 기반한 SVM을 학습하기 위해서 아래와 같이 `svm` 함수를 호출한다.

```
svm_model <- svm(as.factor(class) ~ x1 + x2, data = nonlinear_train_df,
                  kernel = "polynomial", coef0 = 1, gamma = 1, degree = 2,
                  scale = FALSE)
```

위 함수 호출 결과 서포트 벡터 객체는 1, 7, 2, 3, 9이다.

비선형 SVM의 분리 하이퍼플레인 또한 페널티 단가 C 의 값에 따라 달라진다. 선형 SVM의 경우와 같이 $C = 1, 5, 100$ 에 대해 각각 비선형 SVM을 구해보자.

```
svm_models <- lapply(
  c(1, 5, 100),
  function(C)
    svm(as.factor(class) ~ x1 + x2, data = nonlinear_train_df,
        kernel = "polynomial", coef0 = 1, gamma = 1, degree = 2,
        cost = C, scale = FALSE)
)
```

Table 9.7: 페널티 단가 C에 따른 비선형 SVM 결과

페널티 단가 \$C\$	서포트 벡터 객체	오분류 객체
1	1, 7, 2, 3, 9	7
5	6, 7, 2, 3, 9	
100	6, 7, 2, 3, 9	

```
getSummary <- function(model) {
  list(C = model$cost,
       sv = paste(model$index, collapse = ", "),
       misclassified = paste(which(model$fitted != as.factor(nonlinear_train_df$class))
  )}

svm_summary <- lapply(svm_models, getSummary) %>% bind_rows()
```

9.6 R 패키지 내 SVM

9.6.1 커널함수

앞 장에서는 선형 커널과 다항 커널함수의 예를 살펴보았다. 본 장에서는 가우시안 커널 및 시그모이드 커널을 사용하는 법을 살펴보자.

가우시안 커널의 경우

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

과 같이 γ 파라미터를 이용하여 함수를 정의하며, `svm` 함수에 `gamma` 파라미터값을 통해 설정할 수 있다.

```
svm_model <- svm(as.factor(class) ~ x1 + x2, data = nonlinear_train_df,
                  kernel = "radial", gamma = 1,
                  cost = 5, scale = FALSE)
plot(svm_model, data = nonlinear_train_df, formula = x2 ~ x1, grid = 200)
```

시그모이드 커널의 경우

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^\top \mathbf{x}_j + \beta_0)$$

와 같이 두 파라미터 γ, β_0 의 값에 대응하는 `svm` 함수의 파라미터 `gamma`, `coef0` 값을 통해 설정할 수 있다.

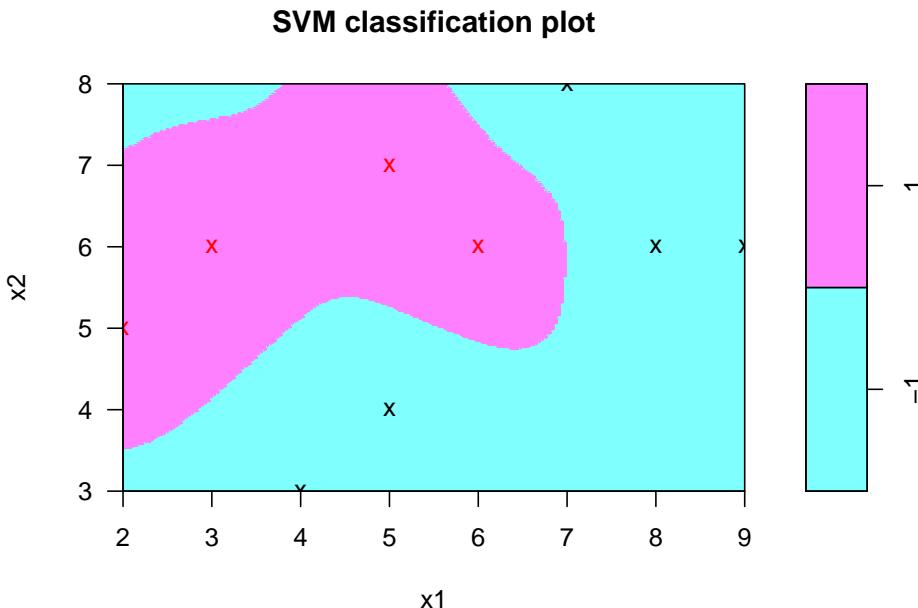


Figure 9.5: 가우시안 커널을 이용한 비선형 SVM 하이퍼플레인

```
svm_model <- svm(as.factor(class) ~ x1 + x2, data = nonlinear_train_df,
                  kernel = "sigmoid", gamma = 0.01, coef0 = -1,
                  cost = 5, scale = FALSE)
plot(svm_model, data = nonlinear_train_df, formula = x2 ~ x1, grid = 200)
```

커널 함수의 종류 `kernel`, 커널 함수의 파라미터 `gamma`, `coef0`, `degree`, 페널티 단가 `cost`등의 `svm` 함수 파라미터는 학습 표본과는 별도의 테스트 데이터에 대해 오분류율을 최소화하는 값을 선택하는 것이 일반적이다.

9.6.2 ν -SVC

ν -support vector classification(ν -SVC) (Schölkopf et al., 2000, Chang and Lin (2001))은 C -SVC의 이차계획식 (9.8)과 다른 형태로, 페널티 단가 C 대신 ν 라는 파라미터를 이용한 아래 최적화 문제의 해를 구한다.

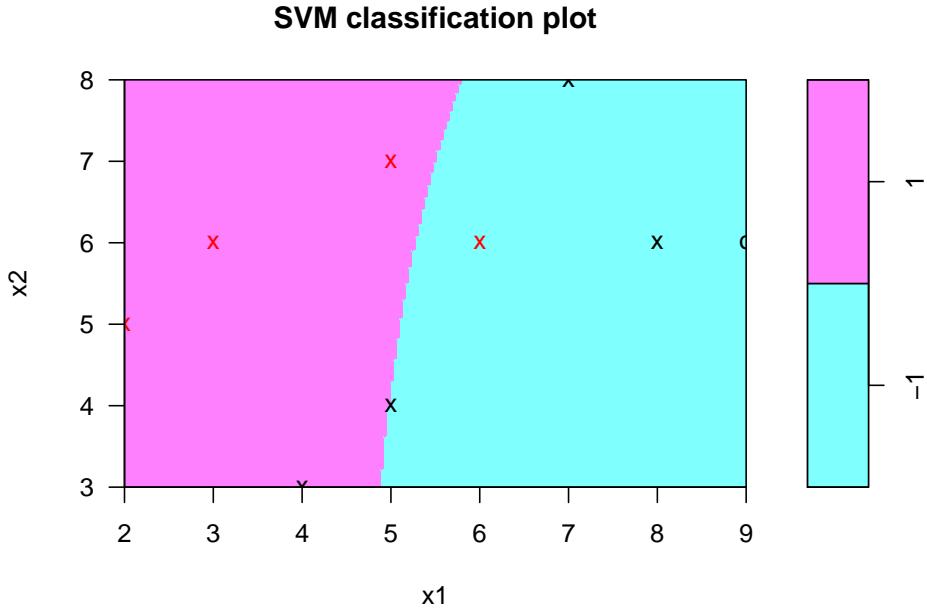


Figure 9.6: 시그모이드 커널을 이용한 비선형 SVM 하이퍼플레인

$$\min L_D = \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j k_{ij}$$

s.t.

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad (9.9)$$

$$0 \leq \alpha_i \leq \frac{1}{N}, \quad i = 1, \dots, N$$

$$\sum_{i=1}^N \alpha_i = \nu$$

이 때, 각 α_i 의 최대값은 $1/N$ 으로, ν 를 포함한 제약식을 무시할 때 모든 객체에 대한 α_i 값의 합의 이론적 최대치는 1이 되며, $\nu \in (0, 1]$ 은 전체 객체 중 서포트 벡터 객체의 개수를 제한하는 개념으로 생각할 수 있다. 식 (9.9)이 실제로 최적해를 가지기 위한 ν 값의 범위는

$$0 < \nu \leq \frac{2}{N} \min \left(\sum_i I(y_i = 1), \sum_i I(y_i = -1) \right)$$

으로 (Chang and Lin, 2001), 이를 들어 범주 1에 속하는 학습표본 객체 수가 전체의 10% 라면, ν 값은 최대 0.2 까지 설정할 수 있다. 또한

`svm` 함수가 호출하는 LIBSVM 라이브러리는 위 식 (9.9)을 N 이 큰(학습 표본 수가 매우 많은) 경우에도 안정된 결과를 얻을 수 있도록 아래와 같이 변환한 문제를 다룬다.

$$\begin{aligned} \min L_D = & \sum_{i=1}^N \sum_{j=1}^N \bar{\alpha}_i \bar{\alpha}_j y_i y_j k_{ij} \\ \text{s.t.} \quad & \sum_{i=1}^N \bar{\alpha}_i y_i = 0 \\ & 0 \leq \bar{\alpha}_i \leq 1, \quad i = 1, \dots, N \\ & \sum_{i=1}^N \bar{\alpha}_i = \nu N \end{aligned} \tag{9.10}$$

이 때 $\bar{\alpha}_i = \alpha_i N$ 이다.

ν -SVC은 아래와 같이 `svm` 함수를 호출할 때 `type = "nu-classification"`과 파라미터 `nu` 값을 설정함으로써 학습할 수 있다.

```
svm_model <- svm(as.factor(class) ~ x1 + x2, data = nonlinear_train_df,
                  type = "nu-classification",
                  kernel = "radial", gamma = 1,
                  nu = 0.5, scale = FALSE)
plot(svm_model, data = nonlinear_train_df, formula = x2 ~ x1, grid = 200)
```

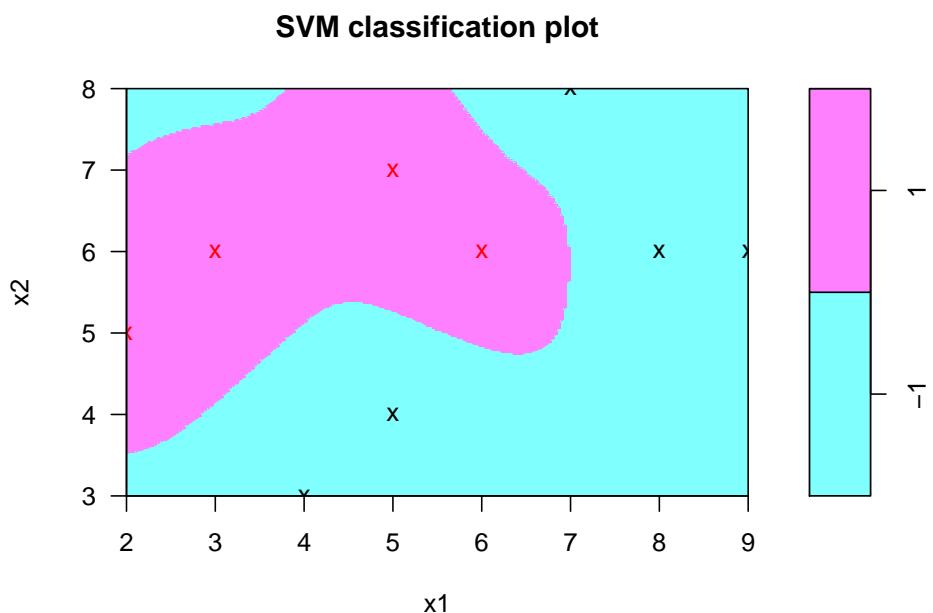


Figure 9.7: 가우시안 커널을 이용한 ν -SVC 하이퍼플레인

Chapter 10

분류규칙의 성능 평가

도출된 분류규칙에 대한 평가는 범주를 아는 학습표본이 있으므로 비교적 용이하게 이루어진다. 분류정확도 또는 분류오류율이 기본이 되나, 특히 범주가 2개인 경우에는 다양한 성능평가척도가 개발되어 사용되고 있다.

10.1 필요 R 패키지 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
caret	6.0-84

10.2 분류오류율

범주를 아는 데이터 $\{(\mathbf{x}_i, y_i)\}_{i=1,\dots,N}$ 를 학습표본이라 한다.

- \mathbf{x}_i : p 개의 독립변수로 이루어진 i 번째 객체의 변수벡터 ($\mathbf{x}_i = [x_{i1} \ x_{i2} \ \cdots \ x_{ip}]^\top$)
- J : 총 범주 수
- y_i : i 번째 객체의 범주 변수; $y_i \in \{1, 2, \dots, J\}$

분류규칙 $d(\mathbf{x})$ 의 성능은 주로 분류오류율(misclassification rate)을 사용하는데, 분류규칙이 추정한 범주와 실제범주가 일치하지 않는 비율을 나타낸다.

$$R(d) = \frac{1}{N} \sum_{i=1}^N I(d(\mathbf{x}_i) \neq y_i) \quad (10.1)$$

여기서 함수 지시함수 $I(x)$ 는 x 가 참(true) 일 때 1, 거짓(false) 일 때 0의 값을 갖는다.

식 (10.1)은 학습표본에 대한 오분류율로, 이를 최소화하려할 경우 분류규칙이 해당 학습 데이터에만 과적용(overfitting) 되는 문제가 발생할 수 있다. 즉, 새로운 데이터에 적용할 때도 오분류율이 최소화될 것이라는 보장이 없다.

이 때문에, 통상 관측수가 상당수 있는 데이터에 대해서는 전체 데이터를 두 부분으로 나누어, 분류규칙을 만드는 데 한 부분을 사용하고, 분류오류율을 산출하는 데 다른 한 부분을 사용하는 방안이 일반적이다. 아래와 같이 범주가 알려져있지만 분류규칙 $d(\mathbf{x})$ 를 학습하는 데 사용하지 않은 L 개의 테스트 표본 $\{(\mathbf{x}_i, y_i)\}_{i=N+1, \dots, N+L}$ 이 있다고 하자. 이 때 테스트 표본에 대한 분류오류율을 아래와 같이 계산한다.

$$R^{ts}(d) = \frac{1}{L} \sum_{i=N+1}^{N+L} I(d(\mathbf{x}_i) \neq y_i) \quad (10.2)$$

테스트 표본으로 분리하기에 충분하지 않은 데이터의 경우에는 cross validation 기법을 사용한다.

10.3 정확도, 민감도 및 특이도

의학 분야에서 어떤 질병에 대한 진단방법을 평가할 때 오류율 이외에 정확도, 민감도 및 특이도를 분석하는 경우가 종종 있다. 실제범주가 질병이 있는 경우(1 또는 +로 표기)와 질병이 없는 경우(0 또는 -로 표기)의 두 가지로 분류된다고 하고, 진단 방법이 양성(1 또는 +) 또는 음성(0 또는 -)으로 판정할 때, 아래와 같이 네 가지 경우가 발생한다. 이와 같은 표를 정오분류표(confusion matrix)라 한다.

```
cm <- matrix(c('a', 'b', 'c', 'd'), nrow = 2, byrow = TRUE)
attr(cm, "dimnames") <- list(Prediction = c("1", "0"), Reference = c("1", "0"))
class(cm) <- "table"
print(cm)

##             Reference
## Prediction 1 0
##           1 a b
##           0 c d
```

위 표의 문자들은 다음과 같이 정의된다.

- a : number of true positive prediction
- b : number of false positive prediction
- c : number of false negative prediction
- d : number of true negative prediction

여기서 “positive” 또는 “negative”는 “양성” 또는 “음성”으로 추정됨을 나타내고, “true” 또는 “false”는 추정의 사실 또는 거짓을 나타낸다. 이 때 분류오류율은 다음과 같이 산출된다.

$$\text{misclassification rate} = \frac{b + c}{a + b + c + d} \quad (10.3)$$

정확도(accuracy)는 오류율의 반대 개념으로, 실제 범주를 제대로 추정한 전체 비율을 나타내며 아래와 같이 산출된다.

$$\text{accuracy} = \frac{a + d}{a + b + c + d} = 1 - \text{misclassification rate} \quad (10.4)$$

한편, 민감도(sensitivity)는 실제 질병이 있는 경우를 양성으로 판정하는 비율을 나타내는 것으로, 다음과 같이 산출된다.

$$\text{sensitivity} = \frac{a}{a + c} \quad (10.5)$$

그리고 특이도(specificity)란 실제 질병이 없는 경우를 음성으로 판정하는 비율을 나타내는 것으로 다음과 같다.

$$\text{specificity} = \frac{d}{b + d} \quad (10.6)$$

정확도를 민감도 및 특이도로 표현하면 다음과 같다.

$$\text{accuracy} = \frac{a + c}{a + b + c + d} \text{sensitivity} + \frac{b + d}{a + b + c + d} \text{specificity}$$

민감도 및 특이도를 별도로 산출하여 분석하는 이유 중 하나는, 동일한 정확도를 갖는다 하더라도 민감도와 특이도는 다를 수 있기 때문이다. 경우에 따라서는 높은 민감도를 원하거나 높은 특이도를 원할 수 있다.

10.3.1 R 패키지 내 정오분류표

100개의 객체에 대한 실제범주와 추정범주가 아래와 같이 주어진다고 하자.

$$\begin{aligned} y_i &= \begin{cases} 1 & i = 1, \dots, 20 \\ 0 & i = 21, \dots, 100 \end{cases}, \\ \hat{y}_i &= \begin{cases} 1 & i = 1, \dots, 15, 91, \dots, 100 \\ 0 & i = 16, \dots, 90 \end{cases} \end{aligned}$$

```
y <- factor(c(rep(1, 20), rep(0, 80)), levels = c(1, 0))
y_hat <- factor(c(rep(1, 15), rep(0, 75), rep(1, 10)), levels = c(1, 0))
```

해당 추정결과에 대한 정오분류표 및 각종 평가지표를 얻기 위해 caret 패키지의 confusionMatrix 함수를 이용한다.

```
cm <- caret::confusionMatrix(data = y_hat, reference = y)
```

우선 정오분류표는 결과 객체의 table component에 저장된다.

```
cm$table
```

```
##           Reference
## Prediction 1 0
##           1 15 10
##           0 5 70
```

정확도를 비롯한 각종 전반적인 지표는 overall이라는 component에 벡터 형태로 저장된다.

```
cm$overall
```

```
##      Accuracy      Kappa AccuracyLower AccuracyUpper AccuracyNull
##      0.8500000    0.5714286    0.7646925    0.9135456    0.8000000
##  AccuracyPValue  McnemarPValue
##      0.1285055    0.3016996
```

또한, 민감도, 특이도를 비롯한 몇 가지 분류성능 지표들은 byClass라는 component에 역시 벡터 형태로 저장된다.

```
cm$byClass
```

```
##      Sensitivity      Specificity      Pos Pred Value
##      0.7500000    0.8750000    0.6000000
##      Neg Pred Value      Precision      Recall
##      0.9333333    0.6000000    0.7500000
##      F1      Prevalence      Detection Rate
##      0.6666667    0.2000000    0.1500000
##  Detection Prevalence      Balanced Accuracy
##      0.2500000    0.8125000
```

10.4 ROC 곡선

일반적으로 민감도와 특이도를 동시에 증가시키는 것은 불가능하다. 다시 말하면, 민감도를 높이면 특이도가 감소하고, 또한 반대가 성립하게 된다.

예를 들어 다음과 같이 10개의 객체로 이루어진 학습표본이 있다고 하자.

```
train_df <- tribble(
  ~x, ~y,
  24, 0,
  35, 0,
  37, 1,
  42, 0,
  49, 1,
  54, 1,
  56, 0,
  68, 1,
  72, 1,
  73, 1
) %>%
  mutate(y = factor(y, levels = c(1, 0)))
```

분류기준이 만약 $x < 40$ 이면 범주 0, $x \geq 40$ 이면 범주 1로 추정할 때, 정오분류표는 다음과 같다.

```
cm40 <- caret::confusionMatrix(
  factor(as.integer(train_df$x >= 40), levels = c(1, 0)),
  train_df$y
)

cm40$table

##           Reference
## Prediction 1 0
##             1 5 2
##             0 1 2
```

이 때 구해지는 민감도 및 특이도는 아래와 같다.

```
cm40$byClass[c("Sensitivity", "Specificity")]

## Sensitivity Specificity
## 0.8333333 0.5000000
```

한편, 분류기준이 만약 $x < 50$ 이면 범주 0, $x \geq 50$ 이면 범주 1로 추정할 때, 정오분류표는 다음과 같다.

```
cm50 <- caret::confusionMatrix(
  factor(as.integer(train_df$x >= 50), levels = c(1, 0)),
  train_df$y
)

cm50$table

##             Reference
## Prediction 1 0
##           1 4 1
##           0 2 3
```

또한, 이 때 구해지는 민감도 및 특이도는 아래와 같다.

```
cm50$byClass[c("Sensitivity", "Specificity")]

## Sensitivity Specificity
## 0.6666667 0.7500000
```

위 x 값 40을 기준으로 분류를 하는 경우와 비교하여 민감도는 감소하고 특이도는 증가함을 관찰할 수 있다.

분류를 위한 x 기준값(threshold)을 증가시켜가면서 민감도와 특이도가 어떻게 변하는지 살펴보도록 하자.

```
univariate_binary_rule <- function(x, y, th) {
  cm <- caret::confusionMatrix(
    factor(as.integer(x >= th), levels = c(1, 0)),
    y
  )

  tibble(threshold = th,
         sensitivity = cm$byClass["Sensitivity"],
         specificity = cm$byClass["Specificity"])
}

th <- c(sort(train_df$x), Inf)

roc_df <- map_dfr(th, univariate_binary_rule, x = train_df$x, y = train_df$y)

knitr::kable(
```

Table 10.1: 분류기준별 민감도 및 특이도

분류기준값(\$x\$)	민감도(sensitivity)	특이도(specificity)
24	1.0000000	0.00
35	1.0000000	0.25
37	1.0000000	0.50
42	0.8333333	0.50
49	0.8333333	0.75
54	0.6666667	0.75
56	0.5000000	0.75
68	0.5000000	1.00
72	0.3333333	1.00
73	0.1666667	1.00
Inf	0.0000000	1.00

```
roc_df, booktabs = TRUE,
align = c('r', 'r', 'r', 'r'),
col.names = c('분류기준값($x$)', '민감도(sensitivity)', '특이도(specificity)'),
caption = '분류기준별 민감도 및 특이도'
)
```

민감도와 특이도를 동시에 그래프로 나타낸 것 중 ROC(receiver operating characteristic) 곡선이 널리 사용되는데, 이는 분류기의 경계치를 조정하여 가면서 $(1 - \text{특이도})$ (또는 false positive rate)을 x 축에, 민감도를 y 축에 도식화한 것이다.

위 Table 10.1를 바탕으로 ROC 곡선을 작성해보자.

```
roc_df %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity)) +
  geom_path()
```

10.5 이익도표

이익도표는 마케팅을 위하여 수익을 창출하는 목표고객(target)을 추출할 목적으로 사용되는데, 단순히 분류를 위한 여러 모형을 비교하기 위한 목적으로도 종종 사용되고 있다. 목표 마케팅의 목적에서는, 특정 범주의 고객을 목표고객으로 할 때, 이러한 목표고객의 비율이 상대적으로 높은 서브그룹을 찾고자 하는 것이다. 이를 위해, 우선 전체 데이터를 특정 범주의 사후확률의 순서로 정렬한 후, K 개(주로 $K = 10$ 을 사용)의 집단으로 구분하고, 각 집단별로 다음과 같은 통계량을 산출한다.

k 번째 집단 내에서 범주 j 에 속한 객체의 수를 n_{kj} 라 할 때, 다음과 같은 범주 j 에 대한 k

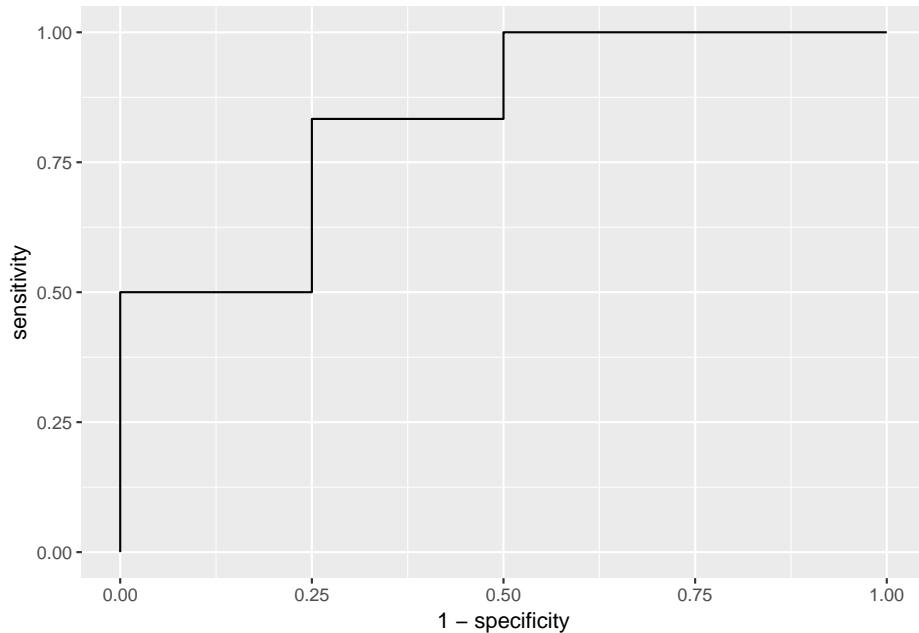


Figure 10.1: ROC 곡선

번째 집단의 통계량들을 산출할 수 있다. (본 장에서 K 개의 집단은 동일한 크기라 가정하자. 즉, 모든 집단 k 에 대해 $\sum_{j=1}^J n_{kj} = \frac{N}{K}$ 가 성립한다고 하자.)

$$\begin{aligned}\% \text{ captured response} &= \frac{n_{kj}}{\sum_{k=1}^K n_{kj}} \times 100 \\ \text{cumulative \% captured response} &= \frac{\sum_{l=1}^k n_{lj}}{\sum_{k=1}^K n_{kj}} \times 100 \\ \% \text{ response} &= \frac{n_{kj}}{\sum_{j=1}^J n_{kj}} \times 100 \\ \text{lift} &= \frac{n_{kj}}{\frac{1}{K} \sum_{k=1}^K n_{kj}}\end{aligned}$$

1,000개의 객체로 이루어진 어떤 데이터의 실제 범주별 빈도가 다음과 같다고 하자.

```
y_freq <- tribble(
  ~y, ~n,
  1, 437,
  2, 348,
  3, 215
) %>%
```

```

  mutate(y = factor(y, levels = c(1, 2, 3)))

y_freq

## # A tibble: 3 x 2
##   y      n
##   <fct> <dbl>
## 1 1      437
## 2 2      348
## 3 3      215

```

한편, 어떤 분류모형을 사용하여 각 객체의 범주 1(특정 범주)에 대한 사후확률을 산출한 후, 전체 객체를 사후확률의 내림차순으로 정렬한 뒤 100개 객체씩 한 집단으로 구분하였다. 각 집단에 속하는 범주 1의 빈도는 다음과 같았다.

```

freq_within_group <- tribble(
  ~k, ~n,
  1, 92,
  2, 78,
  3, 64,
  4, 57,
  5, 43,
  6, 35,
  7, 29,
  8, 22,
  9, 7,
  10, 10
)

freq_within_group

```

```

## # A tibble: 10 x 2
##       k     n
##   <dbl> <dbl>
## 1     1     92
## 2     2     78
## 3     3     64
## 4     4     57
## 5     5     43
## 6     6     35
## 7     7     29
## 8     8     22
## 9     9      7
## 10    10     10

```

Table 10.2: 이익도표를 위한 통계량

집단	범주 1의 빈도	% captured response	cum. % captured response	% response	lift
1	92	21.05	21.05	92	2.11
2	78	17.85	38.90	78	1.78
3	64	14.65	53.55	64	1.46
4	57	13.04	66.59	57	1.30
5	43	9.84	76.43	43	0.98
6	35	8.01	84.44	35	0.80
7	29	6.64	91.08	29	0.66
8	22	5.03	96.11	22	0.50
9	7	1.60	97.71	7	0.16
10	10	2.29	100.00	10	0.23

이를 바탕으로 각 집단 별 범주 1에 대한 통계량을 산출해보자.

```
stat_df <- freq_within_group %>%
  mutate(cum_n = cumsum(n)) %>%
  mutate(
    captured_response_pct = n / sum(n) * 100,
    cum_captured_response_pct = cum_n / sum(n) * 100,
    response_pct = n / 100 * 100,
    lift = n / mean(n)
  ) %>%
  select(-cum_n)

knitr::kable(
  stat_df,
  booktabs = TRUE,
  align = rep('r', 6),
  col.names = c('집단', '범주 1의 빈도', '% captured response',
               'cum. % captured response', '% response', 'lift'),
  caption = '이익도표를 위한 통계량',
  digits = 2
)
```

Table 10.2를 바탕으로 네 가지 이익도표를 작성해보자.

```
stat_df %>%
  gather(key = "stat", value = "value",
         captured_response_pct:lift) %>%
  ggplot(aes(x = k, y = value)) +
  geom_point() +
  geom_line()
```

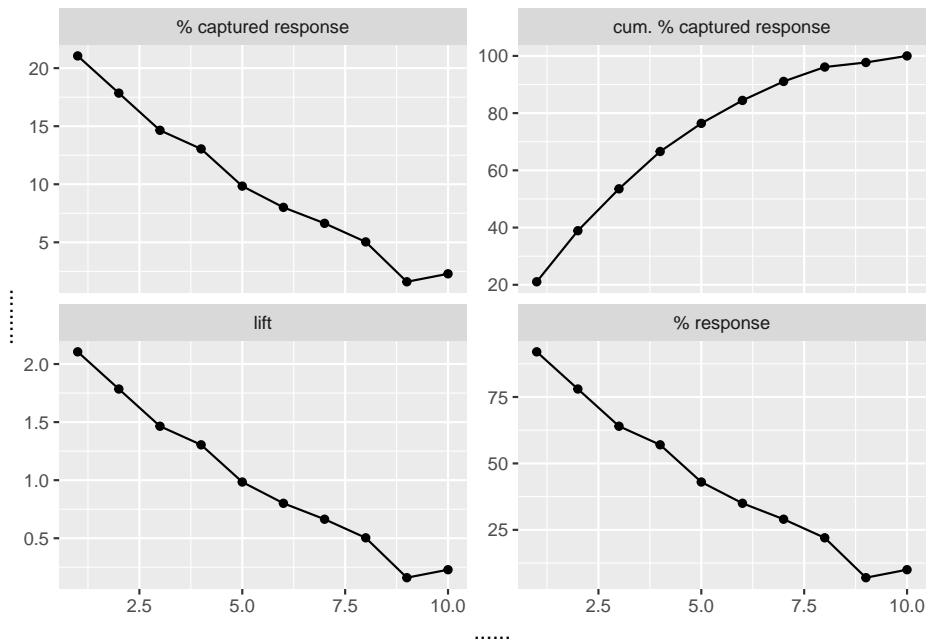


Figure 10.2: 이익도표

```

facet_wrap(vars(stat), nrow = 2, ncol = 2, scales = "free_y",
           labeller = as_labeller(
             c("captured_response_pct" = "% captured response",
               "cum_captured_response_pct" = "cum. % captured response",
               "response_pct" = "% response",
               "lift" = "lift"))
           )) +
  xlab("group") +
  ylab("statistics")

```


Part III

3부 - 군집분석

Chapter 11

군집분석 개요

하나의 객체(object)가 여러 속성(attribute)을 갖는다 하고, 이러한 객체가 다수 있다고 하자. 군집분석이란 유사한 속성들을 갖는 객체들을 묶어 전체의 객체들을 몇 개의 그룹 또는 군집(cluster)으로 나누는 것을 말한다.

11.1 필요 R 패키지 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
stats	3.6.0
corr	0.3.2
cluster	2.0.8

11.2 군집분석 기법

전체 객체의 개수를 n 이라 하고, i 번째 객체를 O_i 라 할 때, 전체 객체의 집합 S 는 다음과 같다.

$$S = \{O_1, \dots, O_n\}$$

군집분석이란 집합 S 를 서로 배타적인 K 개의 부분집합 C_1, \dots, C_K 로 나누는 것이다. 따라서 다음이 성립한다.

$$\begin{aligned}C_i \cap C_j &= \emptyset, 1 \leq i \neq j \leq K \\ \cup_{i=1}^K C_i &= S\end{aligned}$$

이 때, C_j 를 j 번째 군집(또는 군집 j)이라 한다. 각 객체는 한 군집에만 속하여야 하며, 한 군집에는 적어도 하나의 객체를 포함하여야 한다. 군집들을 다음과 같이 모아놓은 것을 군집결과(clustering result) 또는 군집해(clustering solution)라 한다.

$$C = \{C_1, \dots, C_K\}$$

군집방법(clustering method)은 무수히 많다. 다음 장들에서 아래에 분류된 방법들을 보다 자세히 다룬다.

- 계층적 방법(hierarchical method)
 - 집과법(agglomerative method)
 - 분리법(divisive method)
- 비계층적 방법(non-hierarchical method)

11.3 객체 간의 유사성 척도

11.3.1 거리 관련 척도

각 객체가 p 개의 속성 또는 변수(variable)를 갖는다 하고, j 번째 변수의 객체 i 에 대한 관측치를 x_{ji} 라 하면, 객체 i 의 p 차원 공간에서의 좌표는 아래와 같은 열벡터로 표현된다.

$$\mathbf{x}_i = [x_{1i} \ x_{2i} \ \dots \ x_{pi}]^\top$$

이 때, 객체 i 와 객체 j 의 거리를 나타내는 척도들은 아래와 같은 것들이 있다.

- 유clidean 거리(Euclidean distance)

$$\begin{aligned} d(\mathbf{x}_i, \mathbf{x}_j) &= \sqrt{\sum_{a=1}^p (x_{ai} - x_{aj})^2} \\ &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^\top (\mathbf{x}_i - \mathbf{x}_j)} \end{aligned}$$

- 맨하탄 거리(Manhattan distance)

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{a=1}^p |x_{ai} - x_{aj}|$$

- 민코프스키 거리 (Minkowski distance)

$$d(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{a=1}^p |x_{ai} - x_{aj}|^m \right)^{\frac{1}{m}}$$

- 표준 유clidean 거리 (standardized Euclidean distance)

$$\begin{aligned} d(\mathbf{x}_i, \mathbf{x}_j) &= \sqrt{\sum_{a=1}^p \left(\frac{x_{ai} - x_{aj}}{s_a} \right)^2} \\ &= \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{S}_d^{-1} (\mathbf{x}_i - \mathbf{x}_j)} \end{aligned}$$

여기서

$$\begin{aligned} \mathbf{S}_d &= \begin{bmatrix} s_1^2 & 0 & \dots & 0 \\ 0 & s_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & s_p^2 \end{bmatrix} \\ s_a &= \sqrt{\frac{\sum_{i=1}^n (x_{ai} - \bar{x}_a)^2}{n-1}} \\ \bar{x}_a &= \frac{1}{n} \sum_{i=1}^n x_{ai} \end{aligned}$$

- 마할라노비스 거리 (Mahalanobis distance)

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{S}^{-1} (\mathbf{x}_i - \mathbf{x}_j)}$$

여기서

$$\begin{aligned} \mathbf{S} &= \begin{bmatrix} s_1^2 & s_{12} & \dots & s_{1p} \\ s_{21} & s_2^2 & \dots & s_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ s_{p1} & s_{p2} & \dots & s_p^2 \end{bmatrix} \\ s_{ab} &= \frac{\sum_{i=1}^n (x_{ai} - \bar{x}_a)(x_{bi} - \bar{x}_b)}{n-1} \\ \bar{x}_a &= \frac{1}{n} \sum_{i=1}^n x_{ai} \end{aligned}$$

위와 같은 거리 척도들을 이용하여 객체들의 모든 쌍에 대한 거리를 다음과 같이 ($n \times n$) 행렬 \mathbf{D} 로 나타낼 수 있다.

$$\mathbf{D} = \begin{bmatrix} 0 & d(\mathbf{x}_1, \mathbf{x}_2) & \dots & d(\mathbf{x}_1, \mathbf{x}_n) \\ d(\mathbf{x}_2, \mathbf{x}_1) & 0 & \dots & d(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ d(\mathbf{x}_n, \mathbf{x}_1) & d(\mathbf{x}_n, \mathbf{x}_2) & \dots & 0 \end{bmatrix}$$

아래 표는 가정에서 PC를 사용하는 10명에 대한 나이(x_1), PC 경험연수(x_2), 주당 사용시간(x_3)을 나타낸 것이다.

```
df <- tribble(
  ~id, ~x1, ~x2, ~x3,
  1, 20, 6, 14,
  2, 28, 8, 13,
  3, 42, 14, 6,
  4, 35, 12, 7,
  5, 30, 15, 7,
  6, 30, 7, 15,
  7, 45, 13, 6,
  8, 46, 4, 2,
  9, 51, 3, 3,
  10, 41, 3, 2
)

df %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r', 'r'),
    col.names = c('객체번호', '나이($x_1$)', 'PC 경험연수($x_2$)', '주당 사용시간($x_3$)'),
    caption = 'PC 사용 데이터'
  )
```

R 함수 `dist`를 이용하여 다양한 거리를 계산할 수 있다.

우선 객체 2로부터 객체 4, 5까지의 유클리드 거리는 아래와 같이 계산된다.

```
dist(df[, c("x1", "x2", "x3")], upper = TRUE) %>%
  broom::tidy() %>%
  filter(
    item1 == 2,
    item2 %in% c(4, 5)
  ) %>%
  knitr::kable(
    booktabs = TRUE,
```

Table 11.1: PC 사용 데이터

객체번호	나이(\$x_1\$)	PC 경험연수(\$x_2\$)	주당 사용시간(\$x_3\$)
1	20	6	14
2	28	8	13
3	42	14	6
4	35	12	7
5	30	15	7
6	30	7	15
7	45	13	6
8	46	4	2
9	51	3	3
10	41	3	2

Table 11.2: 유클리드 거리

객체번호(from)	객체번호(to)	거리
2	4	10.049876
2	5	9.433981

```


```

위 표에서 나타나는 바와 같이, 객체 2를 기준으로 할 때, 객체 4가 객체 5보다 멀리 떨어져 있다고 할 수 있다.

표준화된 거리를 계산하기 위해서는 데이터를 함수 `scale`을 이용하여 데이터를 표준화한 뒤 `dist`함수를 적용한다.

```
dist(scale(df[, c("x1", "x2", "x3")]), upper = TRUE) %>%
  broom::tidy() %>%
  filter(
    item1 == 2,
    item2 %in% c(4, 5)
  ) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c('객체번호(from)', '객체번호(to)', '거리'),
    caption = '표준 유클리드 거리'
)
```

Table 11.3: 표준 유클리드 거리		
객체번호 (from)	객체번호 (to)	거리
2	4	1.663576
2	5	1.954486

표준화된 거리로는 객체 5가 객체 4보다 객체 2에서 멀리 떨어짐을 알 수 있다.

유클리드 거리 외에 민코프스키 거리, 마할라노비스 거리 등은 `dist` 함수의 파라미터 `method` 및 `p`값을 설정하여 계산할 수 있다.

11.3.2 상관계수 관련 척도

또 다른 유사성 척도로 다음과 같은 객체 간의 상관계수를 사용할 수 있다.

$$sim(\mathbf{x}_i, \mathbf{x}_j) = r_{ij} = \frac{\sum_{a=1}^p (x_{ai} - m_i)(x_{aj} - m_j)}{\sqrt{\sum_{a=1}^p (x_{ai} - m_i)^2} \sqrt{\sum_{a=1}^p (x_{aj} - m_j)^2}} \quad (11.1)$$

여기서 m_i 는 객체 i 의 평균값으로 다음과 같다.

$$m_i = \frac{1}{p} \sum_{a=1}^p x_{ai}$$

식 (11.1)은 -1에서 1 사이의 값을 가지며, 값이 클수록 두 객체의 유사성이 크다고 할 수 있다. 여기서도 데이터를 변수별로 표준화한 후 상관계수를 산출함을 추천한다.

아래는 Table 11.1의 객체 1과 객체 6, 8간의 상관계수를 계산한 것이다.

```
t(scale(df[, c("x1", "x2", "x3")])) %>%
  corrr::correlate() %>%
  corrr::stretch(na.rm = TRUE) %>%
  mutate(
    x = as.integer(gsub("V", "", x)),
    y = as.integer(gsub("V", "", y)))
  ) %>%
  filter(
    x == 1,
    y %in% c(6, 8)
  ) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c('객체번호(from)', '객체번호(to)', '상관계수'),
```

Table 11.4: 객체 간 상관계수

객체번호(from)	객체번호(to)	상관계수
1	6	0.9718362
1	8	-0.8348917

```

    caption = '객체 간 상관계수'
)

## 
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'

한편, 상관계수로부터 거리 개념의 비유사성 척도를 원하면 다음의 척도를 사용할 수 있다.


$$d(\mathbf{x}_i, \mathbf{x}_j) = 1 - r_{ij}$$


t(scale(df[, c("x1", "x2", "x3")])) %>%
  corrr::correlate() %>%
  corrr::stretch(na.rm = TRUE) %>%
  mutate(
    x = as.integer(gsub("V", "", x)),
    y = as.integer(gsub("V", "", y)),
    d = 1 - r
  ) %>%
  select(-r) %>%
  filter(
    x == 1,
    y %in% c(6, 8)
  ) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c('객체번호(from)', '객체번호(to)', '거리'),
    caption = '상관계수 기반 비유사성 척도'
  )

## 
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'

```

Table 11.5: 상관계수 기반 비유사성 척도		
객체번호(from)	객체번호(to)	거리
1	6	0.0281638
1	8	1.8348917

11.4 범주형 객체의 유사성 척도

객체의 변수(속성)들 중 일부 또는 전체가 범주형인 경우에는 유사성 척도를 다소 다르게 정의할 필요가 있다. 범주형 변수는 다시 이분형(binary), 서열형(ordinal), 명목형(nominal)으로 구분된다. 이분형은 서열형 또는 명목형에 속할 수도 있으나, 통상적으로 별도로 구분하고 있다.

11.4.1 이분형 변수의 경우

이분형 변수란 변수가 취하는 값이 두 개인 것을 의미하며, 통상 0과 1을 부여한다. 이 경우 사용되는 유사성 척도는 다양하나, 단순매칭(simple matching)과 자카드(Jaccard) 척도가 주로 사용된다.

- 단순매칭

객체 \mathbf{x}_i 와 \mathbf{x}_j 에 대하여 k 번째 변수가 이분형일 때, 해당 변수값에 대한 유사성을 아래와 같이 계산한다.

$$\text{sim}(\mathbf{x}_{ki}, \mathbf{x}_{kj}) = \begin{cases} 1 & \text{if } \mathbf{x}_{ki} = \mathbf{x}_{kj} \\ 0 & \text{if } \mathbf{x}_{ki} \neq \mathbf{x}_{kj} \end{cases}$$

객체의 p 개의 모든 변수가 이분형일 때, 두 객체의 유사성은 아래와 같이 변수별 유사성의 평균으로 계산한다.

$$\text{sim}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{p} \sum_{k=1}^p \text{sim}(\mathbf{x}_{ki}, \mathbf{x}_{kj})$$

- 자카드(Jaccard) 척도

자카드 척도에서는 변수값을 특정 속성이 나타나는(presence) 경우에 1, 나타나지 않는(absence) 경우 0으로 표현할 때, 두 객체에서 모두 나타나는 경우에만 유사한 것으로 평가한다. 결국, 이 척도에서는 두 객체에서 특정 속성이 0인 경우에는 전반적 유사성 척도 산출에 포함되지 않고 무시된다.

Table 11.6: 건강 문진

객체번호	운동여부(\$x_1\$)	음주여부(\$x_2\$)	흡연여부(\$x_3\$)	가족력여부(\$x_4\$)	고혈압여부(\$x_5\$)
1	1	1	1	1	0
2	1	0	0	1	0
3	0	1	1	0	1

$$sim(x_{ki}, x_{kj}) = \begin{cases} 1 & \text{if } x_{ki} = x_{kj} = 1 \\ \text{ignored} & \text{if } x_{ki} = x_{kj} = 0 \\ 0 & \text{if } x_{ki} \neq x_{kj} \end{cases}$$

따라서, 객체의 p 개의 모든 변수가 이분형일 때, 두 객체의 유사성은 아래와 같이 계산한다.

$$sim(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sum_{k:x_{ki}+x_{kj}>0} sim(x_{ki}, x_{kj})}{\sum_{k:x_{ki}+x_{kj}>0} 1}$$

다음은 3명에 대한 건강 관련 문진에 대한 답을 나타낸 자료이다.

```
df <- tribble(
  ~id, ~x1, ~x2, ~x3, ~x4, ~x5,
  1, 1, 1, 1, 0, 1,
  2, 1, 0, 1, 0, 0,
  3, 0, 1, 0, 1, 0
)

df %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r', 'r', 'r', 'r'),
    col.names = c('객체번호', '운동여부($x_1$)', '음주여부($x_2$)', '흡연여부($x_3$)', '가족력여부($x_4$)', '고혈압여부($x_5$)'),
    caption = '건강 문진'
  )
```

객체 1과 2의 단순매칭에 의한 유사성은 다음과 같다.

```
similarity_simplematching <- function(vec_1, vec_2) {
  sum(1 - abs(vec_1 - vec_2)) / length(vec_1)
}

df_pairs <- df %>%
  select(id) %>%
  expand(id_1 = id, id_2 = id) %>%
```

Table 11.7: 단순매칭 유사성 척도

객체번호(from)	객체번호(to)	유사도
1	2	0.6

```

filter(id_1 != id_2)

df_pairs$similarity <- df_pairs %>%
  inner_join(df, by=c("id_1" = "id")) %>%
  inner_join(df, by=c("id_2" = "id")) %>%
  rowwise() %>%
  do(similarity = similarity_simplematching(
    .[c("x1.x", "x2.x", "x3.x", "x4.x", "x5.x")] %>% unlist(),
    .[c("x1.y", "x2.y", "x3.y", "x4.y", "x5.y")] %>% unlist())) %>%
  unlist()

df_pairs %>%
  filter(
    id_1 == 1,
    id_2 == 2
  ) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c('객체번호(from)', '객체번호(to)', '유사도'),
    caption = '단순매칭 유사성 척도'
  )

```

한편 자카드 유사성은 아래와 같이 함수 `dist`를 이용하여 구할 수 있다. 함수 `dist`는 거리 척도 함수로, 자카드 기반 거리의 경우 $d(\mathbf{x}_i, \mathbf{x}_j) = 1 - sim(\mathbf{x}_i, \mathbf{x}_j)$ 를 계산한다. 따라서, 거리값에 기반하여 자카드 유사성을 구하고 싶은 경우, $sim(\mathbf{x}_i, \mathbf{x}_j) = 1 - d(\mathbf{x}_i, \mathbf{x}_j)$ 를 계산하면 된다.

```

dist(df[, -1], method = "binary", upper = TRUE) %>%
  broom::tidy() %>%
  mutate(similarity = 1 - distance) %>%
  select(-distance) %>%
  filter(
    item1 == 1,
    item2 == 2
  ) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c('객체번호(from)', '객체번호(to)', '유사도')
  )

```

Table 11.8: 자카드 유사성 척도

객체번호(from)	객체번호(to)	유사도
1	2	0.5

```

col.names = c('객체번호(from)', '객체번호(to)', '유사도'),
caption = '자카드 유사성 척도'
)

```

11.4.2 서열형 변수의 경우

객체의 k 번째 변수가 서열형이고 $1, 2, \dots, M_k$ 중 한 값을 갖는다고 할 때, 거리척도로는 우선 아래와 같은 직접적 방법이 있다.

$$d(x_{ki}, x_{kj}) = \frac{|x_{ki} - x_{kj}|}{M_k - 1}$$

위에서 분모는 해당 변수가 취할 수 있는 범위(range)를 나타내며, 따라서 위의 값은 0에서 1 사이 값을 갖는다. 이 방법을 사용할 경우, 객체의 모든 변수가 서열형이면 두 객체의 거리는 다음과 같다.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^p d(x_{ki}, x_{kj}) = \sum_{k=1}^p \frac{|x_{ki} - x_{kj}|}{M_k - 1}$$

또 다른 방법은 우선 각 변수를 0에서 1 사이의 값으로 변환한 후, 연속형 변수의 경우와 같이 거리척도를 산출하는 것이다. 이 경우 객체 i 의 k 번째 변수는 다음과 같이 변환한다.

$$x'_{ki} = \frac{x_{ki} - 1}{M_k - 1}$$

11.4.3 명목형 변수의 경우

두 객체에 대한 k 번째 변수가 명목형인 경우, 이분형 변수의 경우와 같이 두 변수가 일치하면 1, 그렇지 않으면 0으로 유사성을 평가한다. 즉,

$$sim(x_{ki}, x_{kj}) = \begin{cases} 1 & \text{if } x_{ki} = x_{kj} \\ 0 & \text{if } x_{ki} \neq x_{kj} \end{cases}$$

p 개의 모든 변수가 명목형인 경우, 두 객체 간 유사성은 다음과 같다.

$$sim(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{p} \sum_{k:x_{ki}=x_{kj}} 1$$

11.4.4 혼합형의 경우

두 객체의 유사성 또는 비유사성을 산출하는 데 각 변수의 형태가 연속형, 이분형, 서열형, 명목형 등으로 다른 경우에는, 각 변수의 형태에 따라 위에서 언급한 바와 같이 각기 다른 방법으로 유사성 또는 비유사성을 평가한 후, 최종적으로 합 또는 평균으로 도출하게 된다. 따라서 편의상 각 변수에 대하여 0에서 1 사이의 값을 갖는 척도를 사용하고 있다. 위에서 언급한 이분형, 서열형, 명목형인 경우에는 이미 0에서 1 사이의 유사성 척도가 제시되었다.

연속형의 경우, 0에서 1 사이의 값을 갖는 거리(비유사성)의 척도로는 아래와 같이 각 변수의 범위를 활용하는 방법을 사용한다.

$$d(x_{ki}, x_{kj}) = \frac{|x_{ki} - x_{kj}|}{R_k}$$

여기서 R_k 는 k 번째 변수의 범위 (=최대값 - 최소값)를 의미한다. 유사성 척도를 원할 경우에는 다음과 같이 산출할 수 있다.

$$sim(x_{ki}, x_{kj}) = 1 - d(x_{ki}, x_{kj})$$

결국, 여러 형태의 변수가 혼합되어 있는 경우, 각 변수에 대한 유사성 척도가 산출되어 있을 때, 두 객체의 유사성은 다음과 같이 계산한다.

$$sim(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{p} \sum_{k=1}^p sim(x_{ki}, x_{kj})$$

또는 각 변수의 거리가 산출도니 경우, 두 객체의 거리는 다음과 같다.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{p} \sum_{k=1}^p d(x_{ki}, x_{kj})$$

위에 설명한 혼합형 거리 척도는 Gower (1971)에 기반하여, R에서는 `cluster` 패키지의 `daisy` 함수를 이용하여 구할 수 있다. `daisy` 함수는 연속형 및 서열형 변수의 경우 입력 데이터에 기반하여 `range`를 계산하므로, 입력 데이터의 최소값, 최대값이 아닌 이론적 최소값, 최대값에 의하여 `range`를 계산하고 싶은 경우에는 명시적으로 각 변수의 최소값과 최대값을 나타내는 데이터를 입력 데이터에 추가하여야 한다.

Table 11.9: 혼합형 Gower 거리

객체번호(from)	객체번호(to)	거리
2	1	0.5513333
3	1	0.5768889
3	2	0.2744444

```

df <- tribble(
  ~id, ~x1, ~x2, ~x3, ~x4, ~x5,
  1, "남", 46, "공무원", 35000, 2,
  2, "여", 28, "은행원", 51000, 3,
  3, "여", 32, "주부", 46000, 4
) %>%
  mutate(
    x1 = factor(x1, levels = c("남", "여")),
    x3 = factor(x3),
    x5 = factor(x5, levels = c(1:5), ordered = TRUE)
  )

n_obs <- nrow(df)

range_df <- tibble(
  x2 = c(25, 70),
  x4 = c(0, 150000),
  x5 = factor(c(1, 5), levels = c(1:5), ordered = TRUE)
)

df %>%
  bind_rows(range_df) %>%
  select(-id) %>%
  cluster::daisy() %>%
  as.dist() %>%
  broom::tidy() %>%
  filter(
    item1 <= n_obs,
    item2 <= n_obs
  ) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c('객체번호(from)', '객체번호(to)', '거리'),
    caption = '혼합형 Gower 거리'
  )

```


Chapter 12

계층적 군집방법

계층적 군집방법에는 집괴법과 분리법이 있으나 주로 집괴법이 사용된다. 본 장에서는 집괴법으로는 연결법을 소개하고, 분리법으로는 다이아나(DIANA)를 소개한다.

12.1 필요 R 패키지 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
stats	3.6.0
cluster	2.0.8

12.2 군집 간 거리척도 및 연결법

계층적 군집방법에서는 유사한 객체들을 군집으로 묶고, 다시 유사한 군집을 새로운 군집으로 묶는 등 단계적 절차를 사용한다. 이를 위해서는 군집 간의 유사성 척도 혹은 비유사성 척도가 필요하다.

- C_i : i 번째 군집(군집 i)
- $|C_i|$: 군집 i 의 객체수
- $\mathbf{c}_i = (\bar{x}_1^{(i)}, \bar{x}_2^{(i)}, \dots, \bar{x}_p^{(i)})$: 군집 i 의 중심좌표(centroid) $(\bar{x}_a^{(i)}) = \frac{1}{|C_i|} \sum_{j \in C_i} x_{aj}$
- $d(u, v) = d(\mathbf{x}_u, \mathbf{x}_v)$: 객체 u 와 객체 v 의 거리(또는 비유사성 척도)
- $D(C_i, C_j)$: 군집 i 와 군집 j 의 거리(또는 비유사성 척도)

군집과 군집 간의 거리척도를 평가하는 방법에 따라 다양한 연결법(linkage method)이 존재한다. 아래에 대표적인 연결법과 군집 간 거리척도를 소개한다.

Table 12.1: 연결법 종류

연결법	군집거리 $\$D(C_i, C_j)$
단일연결법(single linkage method)	$\$min_{\{u \in C_i, v \in C_j\}} d(u, v)$
완전연결법(complete linkage method)	$\$max_{\{u \in C_i, v \in C_j\}} d(u, v)$
평균연결법(average linkage method)	$\$frac{1}{ C_i C_j } \sum_{u \in C_i, v \in C_j} d(u, v)$
중심연결법(centroid linkage method)	$\$d(\mathbf{c}_i, \mathbf{c}_j)$

Table 12.2: PC 사용자 데이터

객체번호	PC 경력(년, $\$x_1$)	사용시간(시간, $\$x_2$)
1	6	14
2	8	13
3	14	6
4	11	8
5	15	7
6	7	15
7	13	6
8	5	4
9	3	3
10	3	2

12.3 연결법의 군집 알고리즘

12.3.1 기본 R 스크립트

```

train_df <- tibble(
  id = c(1:10),
  x1 = c(6, 8, 14, 11, 15, 7, 13, 5, 3, 3),
  x2 = c(14, 13, 6, 8, 7, 15, 6, 4, 3, 2)
)

knitr::kable(train_df, booktabs = TRUE,
  align = c('r', 'r', 'r'),
  col.names = c('객체번호', 'PC 경력(년, $x_1$)', '사용시간(시간, $x_2$)'),
  caption = 'PC 사용자 데이터')

theme_set(theme_gray(base_family='NanumGothic'))
ggplot(train_df, aes(x = x1, y = x2)) +
  geom_text(aes(label = id)) +

```

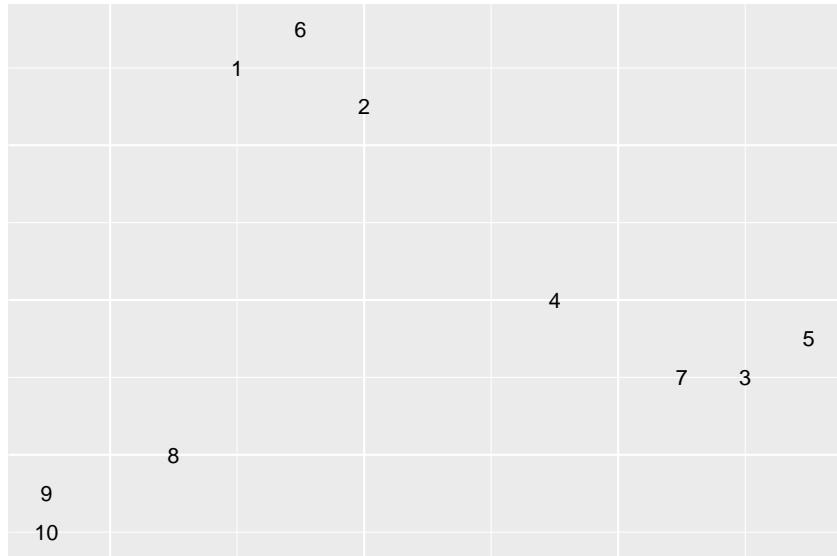


Figure 12.1: PC 사용자 데이터

```
xlab("PC 경력") +
ylab("사용시간")
```

Table 12.2는 10명의 사람(객체)에 대한 PC 사용경력과 주당 PC 사용시간을 나타낸 것이다. 각 객체가 두 변수로 이루어져 있으며, Figure 12.1에서 보는 바와 같이 세 개의 군집($\{1, 2, 6\}$, $\{3, 4, 5, 7\}$, $\{8, 9, 10\}$)으로 이루어져 있다고 볼 수 있다.

본 장에서 평균연결법에 의한 군집화 과정을 살펴보기로 하자. 우선 R 패키지를 이용해서 간단하게 군집해를 구하는 과정은 아래와 같다.

1. `stats` 패키지의 함수 `dist`를 이용하여 객체간 거리를 계산한다.
2. 1에서 얻은 거리 행렬을 `stats` 패키지의 `hclust` 함수에 입력하여 데이터 군집을 분석한다. 이 때, 파라미터 `method`의 값을 “average”로 설정하면 평균연결법을 이용한다.

```
dist(train_df[, -1]) %>%
hclust(method = "average") %>%
plot(
  main = NULL,
  ylab = "distance",
  xlab = "observation"
)
```

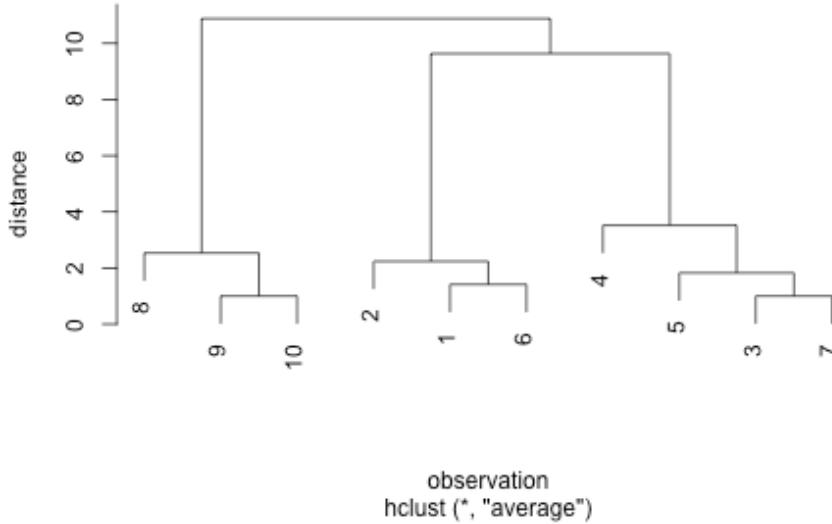


Figure 12.2: PC 사용자 데이터에 대한 평균연결법 텐드로그램

12.3.2 연결법 군집 알고리즘

각 연결법들은 군집간 유사성 척도 평가 방법이 다를 뿐, 군집화를 위한 알고리즘은 동일하게 아래와 같이 진행된다.

0. 단계0: 초기화
 1. 연결법을 선정한다.
 2. 각 객체를 하나의 군집으로 간주한다.
 3. $k \leftarrow n$
1. 단계1: 군집
 1. 현재의 군집결과에 있는 모든 군집 간의 쌍에 대하여 $D(C_i, C_j)$ 를 산출하여, 이 중 최소가 되는 군집 i 와 j 를 묶어 하나의 군집으로 만든 후 군집결과를 수정한다.
 2. $k \leftarrow k - 1$
2. 단계2: $k = 1$ 이면 Stop, 그렇지 않으면 단계 1을 반복한다.

단계1은 객체 수 n 만큼 반복된다.

```
iteration <- vector("list", length = nrow(train_df))
```

임의의 군집해에 대하여, 단계1을 수행하는 함수를 아래와 같이 구현해보자. 아래 함수 `merge_cluster`는 아래와 같은 두 개의 입력변수를 사용한다.

- `df`: 객체 데이터 프레임. 열 이름이 `id`인 열은 객체번호를 나타내어, 객체간 거리 계산에 포함하지 않는다.
- `cluster_label`: 두 개의 열로 이루어진 데이터 프레임. 열 `id`는 객체번호를 나타내며, 열 `cluster`는 군집 이름을 나타낸다. 하나의 객체는 하나의 군집에만 속할 수 있으나, 하나의 군집은 여러 개의 객체를 포함할 수 있다.

함수 수행 결과, 아래와 같은 세 개의 원소를 지닌 리스트를 리턴한다.

- `cluster_dist`: 군집 간 거리를 나타낸 데이터 프레임. 평균연결법에 기반한 거리.
- `closest_clusters`: 입력된 군집해 내에서 가장 가까운 두 군집을 나타낸 데이터 프레임. 두 열 `item1`과 `item2`는 각각 군집 이름을 나타내며, `distance`는 해당 두 군집간의 거리를 나타낸다.
- `new_cluster_label`: `closest_clusters`에 포함된 두 군집을 하나로 묶어 새로운 군집을 만든 후 얻어진 군집해.

```
merge_cluster <- function(df, cluster_label) {
  # 군집간 거리 계산한다. - 유클리드 거리 및 평균연결법 기반
  cluster_dist <- dist(subset(df, select = -id), upper = TRUE) %>%
    broom::tidy() %>%
    inner_join(
      cluster_label %>% rename(
        item1 = id, cluster1 = cluster
      ),
      by = "item1") %>%
    inner_join(
      cluster_label %>% rename(
        item2 = id, cluster2 = cluster
      ),
      by = "item2") %>%
    filter(cluster1 != cluster2) %>%
    group_by(cluster1, cluster2) %>%
    summarize(distance = mean(distance)) %>%
    ungroup()

  # 서로 가장 가깝게 위치하는 두 군집을 찾는다.
  closest_clusters <- cluster_dist %>%
    arrange(distance) %>%
    slice(1)
```

```
# 군집해를 업데이트한다.
cluster_label[
  cluster_label$cluster %in% (
    closest_clusters[, c("cluster1", "cluster2")] %>% unlist()
  ),
  "cluster"
] <- paste(
  closest_clusters[, c("cluster1", "cluster2")] %>% unlist(),
  collapse = ","
)

list(cluster_dist = cluster_dist,
      closest_clusters = closest_clusters,
      new_cluster_label = cluster_label)
}
```

우선 단계 0에서 얻어지는 군집해에 대한 데이터를 아래와 같이 생성한다.

```
init_cluster <- tibble(
  id = train_df$id,
  cluster = as.character(1:nrow(train_df))
)

print(unique(init_cluster$cluster))

## [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"

k <- length(unique(init_cluster$cluster))

print(k)

## [1] 10
```

위와 같이, 초기 군집해에서 군집 수는 전체 객체수와 같은 10개이다.

위 초기해로부터 단계1을 아래와 같이 수행해보자.

```
iteration[[1]] <- merge_cluster(train_df, init_cluster)
```

찾아진 가장 가까운 두 군집은 아래와 같다.

```
iteration[[1]]$closest_cluster

## # A tibble: 1 x 3
##   cluster1 cluster2 distance
##   <chr>     <chr>      <dbl>
## 1 10         9           1
```

위 두 군집을 하나로 묶은 새로운 군집해는 아래와 같다.

```
iteration[[1]]$new_cluster_label

## # A tibble: 10 x 2
##       id cluster
##   <int> <chr>
## 1     1     1
## 2     2     2
## 3     3     3
## 4     4     4
## 5     5     5
## 6     6     6
## 7     7     7
## 8     8     8
## 9     9    10,9
## 10    10   10,9
```

위 새로운 군집해의 군집 수는 9이다. 이는 아직 1보다 크므로, 새로 얻어진 군집해로부터 단계 1을 반복한다.

```
iteration[[2]] <- merge_cluster(
  train_df,
  iteration[[1]]$new_cluster_label
)
```

이번에 찾아진 가장 가까운 두 군집은 아래와 같다.

```
iteration[[2]]$closest_cluster

## # A tibble: 1 x 3
##   cluster1 cluster2 distance
##   <chr>     <chr>      <dbl>
## 1 3         7           1
```

위 두 군집을 하나로 묶은 새로운 군집해는 아래와 같다.

```
iteration[[2]]$new_cluster_label

## # A tibble: 10 x 2
##       id cluster
##   <int> <chr>
## 1     1 1
## 2     2 2
## 3     3 3,7
## 4     4 4
## 5     5 5
## 6     6 6
## 7     7 3,7
## 8     8 8
## 9     9 10,9
## 10    10 10,9
```

위 군집해에 기반하여 단계 1을 다시 반복해보자.

```
iteration[[3]] <- merge_cluster(
  train_df,
  iteration[[2]]$new_cluster_label
)

print(iteration[[3]]$closest_cluster)

## # A tibble: 1 x 3
##   cluster1 cluster2 distance
##   <chr>     <chr>      <dbl>
## 1 1         6           1.41

print(iteration[[3]]$new_cluster_label)

## # A tibble: 10 x 2
##       id cluster
##   <int> <chr>
## 1     1 1,6
## 2     2 2
## 3     3 3,7
## 4     4 4
## 5     5 5
## 6     6 1,6
## 7     7 3,7
## 8     8 8
## 9     9 10,9
## 10    10 10,9
```

위와 같은 과정을 전체 객체가 하나의 군집으로 묶일 때까지 아래와 같이 반복하며 군집 결과를 출력해보자.

```
#단계0
init_cluster <- tibble(
  id = train_df$id,
  cluster = as.character(1:nrow(train_df))
)

i <- 0L
current_clusters <- unique(init_cluster$cluster)
k <- length(current_clusters)

print_clusters <- function(i, k, clusters) {
  cat("Iteration: ", i, ", k = ", k, ", clusters = ", paste0("{", clusters, "}"), "\n")
}

print_clusters(i, k, current_clusters)

## Iteration: 0 , k = 10 , clusters = {1} {2} {3} {4} {5} {6} {7} {8} {9} {10}

#단계1
iteration <- vector("list", length = nrow(train_df) - 1)
while(k > 1) {
  i <- i + 1
  if(i == 1) {
    iteration[[i]] <- merge_cluster(
      train_df,
      init_cluster
    )
  } else {
    iteration[[i]] <- merge_cluster(
      train_df,
      iteration[[i-1]]$new_cluster_label
    )
  }

  current_clusters <- unique(iteration[[i]]$new_cluster_label$cluster)
  k <- length(current_clusters)

  print_clusters(i, k, current_clusters)
}

## Iteration: 1 , k = 9 , clusters = {1} {2} {3} {4} {5} {6} {7} {8} {10,9}
## Iteration: 2 , k = 8 , clusters = {1} {2} {3,7} {4} {5} {6} {8} {10,9}
```

```
## Iteration: 3 , k = 7 , clusters = {1,6} {2} {3,7} {4} {5} {8} {10,9}
## Iteration: 4 , k = 6 , clusters = {1,6} {2} {3,7,5} {4} {8} {10,9}
## Iteration: 5 , k = 5 , clusters = {1,6,2} {3,7,5} {4} {8} {10,9}
## Iteration: 6 , k = 4 , clusters = {1,6,2} {3,7,5} {4} {10,9,8}
## Iteration: 7 , k = 3 , clusters = {1,6,2} {3,7,5,4} {10,9,8}
## Iteration: 8 , k = 2 , clusters = {1,6,2,3,7,5,4} {10,9,8}
## Iteration: 9 , k = 1 , clusters = {1,6,2,3,7,5,4,10,9,8}
```

12.3.3 R 패키지 내 연결법

R에서는 `stats` 패키지의 `hclust` 함수를 이용하여 군집해를 구할 수 있다.

우선, 객체간 거리 행렬을 함수 `dist`를 이용하여 구한다. 아래는 유클리드 거리를 구하는 예이며, 상황에 따라 다른 거리 척도를 이용할 수도 있다.

```
distance_matrix <- dist(train_df[, -1])
```

객체간 거리를 구한 후, 함수 `hclust`를 이용하여 군집분석을 수행한다. 기본설정은 완전연결법이며, 파라미터 `method`의 값을 설정함으로써 단일연결법, 평균연결법, 중심연결법을 수행할 수 있다.

```
cluster_solution <- hclust(distance_matrix, method = "average")
```

결과 객체 `cluster_solution`은 아래와 같은 컴포넌트(components)를 지닌 리스트(list) 객체이다.

```
names(cluster_solution)
```

```
## [1] "merge"          "height"        "order"         "labels"        "method"
## [6] "call"           "dist.method"
```

이 중, `merge`는 2개의 열과 $n - 1$ 개의 행으로 이루어진 행렬로, 연결법 알고리즘의 단계 1 iteration에서 둘이지는 두 군집을 기록한 것이다.

```
cluster_solution$merge
```

```
##      [,1] [,2]
## [1,]   -3   -7
## [2,]   -9  -10
## [3,]   -1   -6
## [4,]   -5    1
## [5,]   -2    3
## [6,]   -8    2
```

```
## [7,] -4 4
## [8,] 5 7
## [9,] 6 8
```

위에서 각 행은 iteration을 나타내며, 두 열은 묶어지는 두 군집을 나타낸다. 값이 0보다 작은 경우에는 번호가 원 객체 번호를 나타내며, 값이 0보다 큰 경우에는 해당 번호의 iteration에서 묶어진 군집을 나타낸다. 예를 들어, 위 결과의 6번째 행 (-8, 2)은 객체 8과 두 번째 iteration에서 얻어진 군집 (객체 9와 10이 묶여진 군집)이 묶여 하나의 군집 (객체 8, 9, 10)을 이루게 됨을 나타낸다.

`height`는 각 iteration에서 묶이는 두 군집간의 거리를 나타내며, 위 Figure 12.2의 텐드로그램에서 세로선의 높이를 나타낸다. Iteration이 증가함에 따라 묶이는 두 군집간의 거리도 증가한다. 일반적으로 이 거리값이 크게 증가하는 iteration에서 두 군집을 묶지 않고 최종 군집해를 도출한다.

```
cluster_solution$height
```

```
## [1] 1.000000 1.000000 1.414214 1.825141 2.236068 2.532248 3.519028
## [8] 9.635217 10.881878
```

위 결과의 경우 iteration 8에서 거리값이 크게 증가한다. 이는 위 Figure 12.2의 텐드로그램에서 3개의 군집에서 2개의 군집으로 묶이는 과정에서 세로선의 높이가 현격히 증가하는 지점이다. 따라서, iteration 7에서 얻어진 3개의 군집이 적절한 군집해라 판단할 수 있겠다.

12.4 워드 방법

워드방법(Ward's method) 역시 각 객체를 하나의 군집으로 간주함을 시작으로 군집들을 묶어 단계적으로 그 수를 하나가 될 때까지 줄여나가는 것인데, 군집의 제곱합을 활용한다.

12.4.1 기본 R 스크립트

아래 Table 12.3는 8명의 운전자에 대한 운전경력과 교통위반 횟수를 나타낸 것이다.

```
train_df <- tibble(
  id = c(1:8),
  x1 = c(4, 20, 3, 19, 17, 8, 19, 18),
  x2 = c(15, 13, 13, 4, 17, 11, 12, 6)
)

knitr::kable(train_df, booktabs = TRUE,
             align = c('r', 'r', 'r'),
             col.names = c('객체번호', '운전경력($x_1$)', '위반횟수($x_2$)'),
             caption = '운전경력에 따른 교통위반 횟수')
```

Table 12.3: 운전경력에 따른 교통위반 횟수

객체번호	운전경력 (\$x_1\$)	위반횟수 (\$x_2\$)
1	4	15
2	20	13
3	3	13
4	19	4
5	17	17
6	8	11
7	19	12
8	18	6

앞 절의 연결법에서 사용했던 `hclust` 함수를 이용하여 워드 방법에 의한 군집해도 구할 수 있으며, 이 때 파라미터 `method`의 값으로 “ward.D2”를 사용한다.

```
dist(train_df[, -1]) %>%
  hclust(method = "ward.D2") %>%
  plot(
    main = NULL,
    xlab = "observation"
  )
```

12.4.2 워드 군집 알고리즘

군집결과가 $\mathbf{C} = \{C_1, C_2, \dots, C_k\}$ 일 때, 군집 C_i 내의 제곱합(within sum of squares)은 다음과 같이 산출된다.

$$SS(C_i) = \sum_{u \in C_i} (\mathbf{x}_u - \mathbf{c}_i)^\top (\mathbf{x}_u - \mathbf{c}_i)$$

이 때, 전체 군집 내 제곱합을 SSW 라 할 때, 이는 다음과 같다.

$$SSW = \sum_{i=1}^k SS(C_i)$$

다음으로, 현 군집의 각 쌍을 묶는다고 할 때의 새로운 SSW 를 산출한 후, 이 값이 가장 크게 되는 군집 쌍을 묶는다.

1. 단계0

- 각 객체를 하나의 군집으로 간주한다.

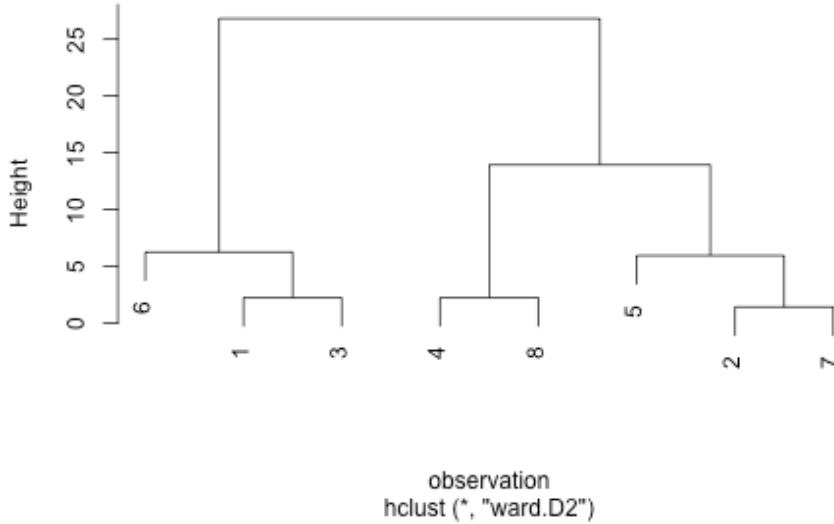


Figure 12.3: 운전자 데이터에 대한 워드 방법 텐드로그램

2. $k \leftarrow n$
2. 단계1
 1. 현재의 군집 결과에 있는 모든 군집간의 쌍에 대하여 뭉을 경우 전체제곱합 (SSW)을 산출하고, 이 중 최소가 되는 군집 i 와 군집 j 를 묶어 하나의 군집으로 만든 후, 군집 결과를 수정한다.
 2. $k \leftarrow k - 1$
3. 단계2: $k = 1$ 이면 Stop, 그렇지 않으면 단계1을 반복한다.

워드 군집 알고리즘을 R script로 구현해보자. 우선, 객체 데이터 SSW를 계산하는 사용자 정의 함수 `calculate_ssw`를 아래와 같이 두 입력변수 `df` 및 `cluster_label`를 이용하여 구현하자.

- `df`: 객체 데이터 프레임. 열 이름이 `id`인 열은 객체번호를 나타내어, 객체간 거리 계산에 포함하지 않는다.
- `cluster_label`: 두 개의 열로 이루어진 데이터 프레임. 열 `id`는 객체번호를 나타내며, 열 `cluster`는 군집 이름을 나타낸다. 하나의 객체는 하나의 군집에만 속할 수 있으나, 하나의 군집은 여러 개의 객체를 포함할 수 있다.

```
# SSW 계산
calculate_ssw <- function(df, cluster_label) {
  df %>%
    inner_join(cluster_label, by = "id") %>%
    group_by(cluster) %>%
    select(-id) %>%
    summarize_all(function(x) sum((x - mean(x))^2)) %>%
    ungroup() %>%
    mutate(ss = rowSums(subset(., select = -cluster))) %>%
    `[[`("ss") %>%
    sum()
}
```

워드 군집 알고리즘은 현재 군집해 내의 모든 군집쌍에 대하여 두 군집을 하나의 군집으로 묶을 경우의 *SSW*를 계산해야 한다. 따라서, 우선 고려할 모든 군집해를 생성하는 사용자 정의 함수 `generate_clusters`를 아래와 같이 구현한다.

아래 사용자 정의 함수 `generate_clusters`는 임의의 군집해 `cluster_label`을 입력변수로 사용하며, 해당 입력변수에 대한 설명은 위 함수 `calculate_ssw`에서와 같다. 함수 수행 결과, 가능한 각각의 군집쌍 결합의 결과물인 군집해 데이터 프레임을 리스트 (list) 형태로 출력한다.

```
# 임의의 군집해로부터 가능한 다음단계 군집해 생성
generate_clusters <- function(cluster_label) {
  unique_clusters <- unique(cluster_label$cluster)

  potential_pairs <- crossing(cluster1 = unique_clusters,
                                cluster2 = unique_clusters) %>%
    filter(cluster1 < cluster2) %>%
    mutate(cluster = paste(cluster1, cluster2, sep = ","))

  candidate_solutions <- potential_pairs %>%
    rowwise() %>%
    do(candidate_solution = merge_cluster(cluster_label, .)) %>%
    `[[`("candidate_solution")

  candidate_solutions
}
```

위에서 보이는 바와 같이, 함수 `generate_clusters`는 또 다른 사용자 정의함수 `merge_cluster`를 호출한다. 이 함수는 두 입력변수 `cluster_label` 및 `cluster_merge`를 사용하는데, `cluster_label`에 대한 설명은 위 다른 사용자 정의 함수에서와 동일하며, `cluster_merge`에 대한 설명은 아래와 같다.

- `cluster_merge`: 3차원 character 벡터. 첫 두 element는 현재 `cluster_label`

에 존재하는 군집 중 하나의 군집으로 묶일 두 군집의 이름을 나타내며, 세 번째 element는 그 결과 나타나는 군집 이름을 나타낸다.

함수 수행 결과, 입력된 cluster_label에서 군집이름이 cluster_merge[1] 혹은 cluster_merge[2]에 해당하는 객체들은, 출력된 군집해에서는 군집이름 cluster_merge[3]을 지닌다.

```
# 임의의 군집 결합 규칙 cluster_merge에 따른 군집해
merge_cluster <- function(cluster_label, cluster_merge) {
  idx <- cluster_label$cluster %in% cluster_merge[1:2]
  cluster_label[idx, "cluster"] <- cluster_merge[3]
  cluster_label
}
```

마지막으로, 현재 군집해로부터 가장 최적의 다음단계 군집해를 얻는 사용자 함수 best_merge_cluster를 아래와 같이 구현해보자.

1. generate_clusters를 실행하여 다음 단계에 가능한 모든 군집해를 구한다.
2. 1의 각 군집해에 함수 calculate_ssw를 적용하여 SSW 값을 구한다.
3. SSW 값이 최소인 군집해를 최적 군집해로 선정한다.

```
# 최적 군집 결합
best_merge_cluster <- function(df, cluster_label) {
  candidate_solutions <- generate_clusters(cluster_label)
  ssw <- sapply(candidate_solutions, function(x) calculate_ssw(df, x))
  list(
    new_cluster_label = candidate_solutions[[which.min(ssw)]],
    new_ssw = min(ssw)
  )
}
```

위 사용자 함수들을 이용하여 Table 12.3에 대한 워드 군집 분석을 수행해보자.

```
#단계0
init_cluster <- tibble(
  id = train_df$id,
  cluster = as.character(1:nrow(train_df))
)
i <- 0L
current_clusters <- unique(init_cluster$cluster)
k <- length(current_clusters)
ssw <- calculate_ssw(train_df, init_cluster)

print_clusters <- function(i, k, clusters, ssw) {
```

```

    cat("Iteration: ", i, ", k = ", k, ", clusters = ", paste0("{", clusters, "}"), ", SSW = ", ssw)
}

print_clusters(i, k, current_clusters, ssw)

## Iteration: 0 , k = 8 , clusters = {1} {2} {3} {4} {5} {6} {7} {8} , SSW = 0

#단계 1
iteration <- vector("list", length = nrow(train_df) - 1)
while(k > 1) {
  i <- i + 1
  if(i == 1) {
    iteration[[i]] <- best_merge_cluster(
      train_df,
      init_cluster
    )
  } else {
    iteration[[i]] <- best_merge_cluster(
      train_df,
      iteration[[i-1]]$new_cluster_label
    )
  }
}

current_clusters <- unique(iteration[[i]]$new_cluster_label$cluster)
k <- length(current_clusters)
ssw <- iteration[[i]]$new_ssw

print_clusters(i, k, current_clusters, ssw)
}

## Iteration: 1 , k = 7 , clusters = {1} {2,7} {3} {4} {5} {6} {8} , SSW = 1
## Iteration: 2 , k = 6 , clusters = {1,3} {2,7} {4} {5} {6} {8} , SSW = 3.5
## Iteration: 3 , k = 5 , clusters = {1,3} {2,7} {4,8} {5} {6} , SSW = 6
## Iteration: 4 , k = 4 , clusters = {1,3} {2,7,5} {4,8} {6} , SSW = 23.66667
## Iteration: 5 , k = 3 , clusters = {1,3,6} {2,7,5} {4,8} , SSW = 43.16667
## Iteration: 6 , k = 2 , clusters = {1,3,6} {2,7,5,4,8} , SSW = 140.4
## Iteration: 7 , k = 1 , clusters = {1,3,6,2,7,5,4,8} , SSW = 499.875

```

12.4.3 R 패키지 내 워드 방법

R 패키지로 구현된 워드 군집은 위에서 구현한 *SSW*와는 다소 다른 metric을 이용하여 군집해를 구한다. 따라서, 우선 워드 방법이 제안된 논문들을 살펴볼 필요가 있다.

우선 월 논문 Ward Jr (1963) 는 ESS (error sum of squares)를 아래와 같이 정의하였으며, 이는 위에서 사용한 SSW 와 일치한다.

$$\begin{aligned} ESS(\{C_1, \dots, C_k\}) &= \sum_{i=1}^k ESS(C_i) \\ &= \sum_{i=1}^k \sum_{u \in C_i} \mathbf{x}_u^\top \mathbf{x}_u - |C_i| \mathbf{c}_i^\top \mathbf{c}_i \\ &= SSW \end{aligned}$$

위 식에서 임의의 두 군집 C_i, C_j 를 하나의 군집으로 묶을 때 SSW 의 변화는 아래와 같다. C_i 와 C_j 외의 군집은 SSW 의 변화에 영향을 미치지 않으므로, SSW 변화량은 아래와 같이 군집 C_i 와 C_j 에 속하는 객체만을 이용하여 구할 수 있으며, 결과적으로 C_i 와 C_j 의 군집 크기 $|C_i|$ 와 $|C_j|$ 및 군집 중심벡터 \mathbf{c}_i 와 \mathbf{c}_j 를 이용하여 구할 수 있다.

$$\begin{aligned} \Delta SSW &= ESS(C_i \cup C_j) - ESS(C_i) - ESS(C_j) \\ &= \sum_{u \in C_i \cup C_j} \mathbf{x}_u^\top \mathbf{x}_u - (|C_i| + |C_j|) \left[\frac{|C_i|\mathbf{c}_i + |C_j|\mathbf{c}_j}{|C_i| + |C_j|} \right]^\top \left[\frac{|C_i|\mathbf{c}_i + |C_j|\mathbf{c}_j}{|C_i| + |C_j|} \right] \\ &\quad - \left(\sum_{u \in C_i} \mathbf{x}_u^\top \mathbf{x}_u - |C_i| \mathbf{c}_i^\top \mathbf{c}_i \right) - \left(\sum_{u \in C_j} \mathbf{x}_u^\top \mathbf{x}_u - |C_j| \mathbf{c}_j^\top \mathbf{c}_j \right) \\ &= - \frac{1}{|C_i| + |C_j|} (|C_i|\mathbf{c}_i + |C_j|\mathbf{c}_j)^\top (|C_i|\mathbf{c}_i + |C_j|\mathbf{c}_j) + |C_i|\mathbf{c}_i^\top \mathbf{c}_i + |C_j|\mathbf{c}_j^\top \mathbf{c}_j \\ &= \frac{|C_i||C_j|}{|C_i| + |C_j|} (\mathbf{c}_i - \mathbf{c}_j)^\top (\mathbf{c}_i - \mathbf{c}_j) \end{aligned} \tag{12.1}$$

따라서 워드 방법은 각 iteration에서 식 (12.1)를 최소화하는 두 군집 C_i, C_j 를 선택하여 두 군집을 하나로 묶는 방법이다.

한편, $SS(C_i)$ 는 아래와 같이 군집 C_i 내 객체들 간의 제곱 유clidean 거리로 나타낼 수 있다.

$$\begin{aligned}
D^2(C_i) &= \sum_{u,v \in C_i} (\mathbf{x}_u - \mathbf{x}_v)^\top (\mathbf{x}_u - \mathbf{x}_v) \\
&= \sum_{u,v \in C_i} ((\mathbf{x}_u - \mathbf{c}_i) - (\mathbf{x}_v - \mathbf{c}_i))^\top ((\mathbf{x}_u - \mathbf{c}_i) - (\mathbf{x}_v - \mathbf{c}_i)) \\
&= 2 \sum_{u \in C_i} (\mathbf{x}_u - \mathbf{c}_i)^\top (\mathbf{x}_u - \mathbf{c}_i) - 2 \sum_{u,v \in C_i} (\mathbf{x}_u - \mathbf{c}_i)^\top (\mathbf{x}_v - \mathbf{c}_i) \\
&= 2 \sum_{u \in C_i} (\mathbf{x}_u - \mathbf{c}_i)^\top (\mathbf{x}_u - \mathbf{c}_i) \\
&= 2SS(C_i)
\end{aligned} \tag{12.2}$$

위 식 (12.2)을 달리 표현하면, 객체간의 제곱 유clidean 거리를 표현한 행렬에서 군집 i 에 속한 객체들에 해당하는 부분행렬(submatrix)를 뽑아 행렬의 원소값을 모두 더하면, 그 값이 $2SS(C_i)$ 와 같다. 이를 통해 각 군집의 중심벡터를 계산하지 않고도 각 iteration에서 SSW를 최소화하는 군집 결합을 찾을 수 있다.

R 패키지 `stats` 내의 `hclust` 함수는 워드 방법으로 `method` 파라미터의 값을 “ward.D” 혹은 “ward.D2”로 설정할 수 있다. 이 두 방법의 차이는 입력 거리행렬을 제곱 유clidean 거리로 사용하는지 일반 유clidean 거리로 사용하는지의 차이로, 아래에서 R 스크립트 예제와 함께 설명하기로 한다.

우선 `method`값을 “ward.D2”로 설정하는 경우, `dist` 함수의 결과를 입력 거리행렬로 그대로 사용하면 된다.

```
res_ward.D2 <- dist(train_df[, -1]) %>%
  hclust(method = "ward.D2")
```

이 때, 결과 데이터 `res_ward.D2`에서 워드 방법의 criterion을 나타내는 `height` 원소 (component)가 표현하는 값은 위에서 계산하였던 SSW와 다르다.

```
res_ward.D2$height
```

```
## [1] 1.414214 2.236068 2.236068 5.944185 6.244998 13.945131 26.813243
```

이는 `height`에서 표현하는 값은 전체 SSW가 아니라, 두 군집 i 와 j 를 하나로 묶을 때 추가로 증가하는 SSW 수치의 변환으로, 아래와 같이 계산되기 때문이다.

$$height = \sqrt{D^2(C_i \cup C_j) - (D^2(C_i) + D^2(C_j))} \tag{12.3}$$

따라서, 군집 i 와 j 를 하나로 묶을 때 증가하는 SSW의 수치 ΔSSW 는 아래와 같이 표현된다.

Table 12.4: hclust 함수 ward.D2 방법의 height와 SSW 관계

iteration	\$height\$	$\$\\Delta SSW = \\frac{1}{2} height^2$$	\$SSW = \\sum \\Delta SSW\$
1	1.414214	1.00000	1.00000
2	2.236068	2.50000	3.50000
3	2.236068	2.50000	6.00000
4	5.944185	17.66667	23.66667
5	6.244998	19.50000	43.16667
6	13.945131	97.23333	140.40000
7	26.813243	359.47500	499.87500

$$\Delta SSW = \frac{1}{2} height^2 \quad (12.4)$$

각 iteration에서 발생하는 ΔSSW 의 누적합이 위 12.4.2절에서 보였던 SSW 결과와 동일함을 아래와 같이 확인해보자.

```
tibble(
  iteration = c(1:(nrow(train_df) - 1)),
  height = res_ward.D2$height
) %>%
  mutate(
    delta_ssw = height ^ 2 / 2
) %>%
  mutate(
    ssw = cumsum(delta_ssw)
) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r', 'r'),
    col.names = c('iteration', '$height$', '$\\Delta SSW = \\frac{1}{2} height^2$', '$SSW = \\sum \\Delta SSW$'),
    caption = 'hclust 함수 ward.D2 방법의 height와 SSW 관계'
)
```

우선 method값을 “ward.D”로 설정하는 경우, dist 함수의 결과를 입력 거리행렬로 그대로 사용하면 아래와 같이 위 “ward.D2”와는 다른 height값을 출력하며, 이는 워드 방법의 criterion을 정확히 반영하지 못한다.

```
res_ward.D <- dist(train_df[, -1]) %>%
  hclust(method = "ward.D")

res_ward.D$height
```

```
## [1] 1.414214 2.236068 2.236068 6.452039 6.615990 17.358484 39.311447
```

이는 “ward.D2”는 워드 방법 수행 전 입력된 유클리드 거리행렬을 내부적으로 제곱하는 반면, “ward.D” 방법은 제곱 유클리드 거리행렬이 입력되는 것을 가정하기 때문이다.

Lance and Williams (1967) 은 군집 i 와 j 를 하나로 묶을 때, 새로 생성된 군집과 다른 군집들간의 거리는 원 두 군집들과 다른 군집들간의 거리로 아래와 같이 표현됨을 보였다. 이를 Lance-Williams update 공식이라 한다.

$$D(C_i \cup C_j, C_{h \notin \{i,j\}}) = \alpha_i D(C_i, C_h) + \alpha_j D(C_j, C_h) + \beta D(C_i, C_j) + \gamma |D(C_i, C_h) - D(C_j, C_h)| \quad (12.5)$$

이후 Wishart (1969) 에서 워드 방법을 위 Lance-Williams update 공식으로 표현하였다.

$$\begin{aligned} \alpha_i &= \frac{|C_i| + |C_h|}{|C_i| + |C_j| + |C_h|} \\ \alpha_j &= \frac{|C_j| + |C_h|}{|C_i| + |C_j| + |C_h|} \\ \beta &= -\frac{|C_h|}{|C_i| + |C_j| + |C_h|} \\ \gamma &= 0 \end{aligned} \quad (12.6)$$

이 때, 식 (12.6)가 기반한 식 (12.5)에서의 거리함수 D 는 제곱 유클리드 거리를 사용한다.

“ward.D” 방법은 제곱 유클리드 거리의 입력을 가정하며, 위의 경우와 같이 제곱 유클리드 거리가 아닌 일반 유클리드 거리행렬을 입력하였을 때, 오류 메시지를 출력하는 대신, 입력된 거리행렬이 제곱 유클리드 거리를 나타낸다 가정하고 Lance-Williams update를 수행한다. 따라서, 이 경우 `height`는 워드 방법의 criterion을 정확히 표현하지 못한다.

제곱 유클리드 거리를 “ward.D” 방법의 입력 거리행렬로 설정하고, 구해진 `height`를 출력해보자

```
res_ward.D <- dist(train_df[, -1])^2 %>%
  hclust(method = "ward.D")

res_ward.D$height
```

```
## [1] 2.00000 5.00000 5.00000 35.33333 39.00000 194.46667 718.95000
```

위 `height`값은 “ward.D2” 방법에서 출력된 값보다 크다. 위 값의 제곱근(square root)를 구하면 “ward.D2”에서의 `height`값과 동일한 값을 얻을 수 있다.

Table 12.5: hclust 함수 ward.D 방법의 height와 SSW 관계

iteration	\$height\$	$\Delta SSW = \frac{1}{2} \text{height}$	$SSW = \sum \Delta SSW$
1	2.00000	1.00000	1.00000
2	5.00000	2.50000	3.50000
3	5.00000	2.50000	6.00000
4	35.33333	17.66667	23.66667
5	39.00000	19.50000	43.16667
6	194.46667	97.23333	140.40000
7	718.95000	359.47500	499.87500

```
sqrt(res_ward.D$height)
```

```
## [1] 1.414214 2.236068 2.236068 5.944185 6.244998 13.945131 26.813243
```

제곱 유클리드 거리행렬을 입력한 “ward.D” 방법의 결과로 출력된 criterion `height`는 $2\Delta SSW$ 의 값에 해당하는 수치이며, 각 iteration 당 $\sum_i D(C_i)$ 의 값의 변화량이라고 볼 수 있다. (식 (12.2) 참조)

```
tibble(
  iteration = c(1:(nrow(train_df) - 1)),
  height = res_ward.D$height
) %>%
  mutate(
    delta_ssw = height / 2
) %>%
  mutate(
    ssw = cumsum(delta_ssw)
) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r', 'r'),
    col.names = c('iteration', '$height$', '$\Delta SSW = \frac{1}{2} height$', '$SSW = \sum \Delta SSW$'),
    caption = 'hclust 함수 ward.D 방법의 height와 SSW 관계'
)
```

즉, “ward.D2”와 “ward.D”의 가장 큰 차이는 입력될 거리행렬이 유클리드 거리 (ward.D2)인지 제곱 유클리드 거리 (ward.D)인지의 차이이다.

참고로, `cluster` 패키지의 `agnes` 함수도 워드 방법을 지원하며, 이 경우 파라미터 `method`의 값을 “ward”로 설정한 결과가 `hclust` 함수의 “ward.D2”의 경우와 동일하다. 본 절에서는 해당 함수의 자세한 사용법은 생략한다.

Table 12.6: DIANA 군집 대상 객체 데이터

객체번호	\$x_1\$	\$x_2\$
1	30	15
2	45	22
3	25	12
4	40	24
5	50	25
6	20	10
7	42	9

```
res_agnes_ward <- cluster::agnes(train_df[, -1], method = "ward")
sort(res_agnes_ward$height)

## [1] 1.414214 2.236068 2.236068 5.944185 6.244998 13.945131 26.813243
```

12.5 분리적 방법 - 다이아나

다이아나는 분리적 방법의 하나로, Kaufman and Rousseeuw (1990)에 의하여 제안된 것이다. 이는 전체의 객체를 하나의 군집으로 시작하여 매번 이분화하는 등 모든 군집이 단독 객체로 구성될 때까지 진행하는 방법이다. 이 때, 비유사성 척도로는 평균거리를 사용한다.

12.5.1 기본 R 스크립트

```
train_df <- tibble(
  id = c(1:7),
  x1 = c(30, 45, 25, 40, 50, 20, 42),
  x2 = c(15, 22, 12, 24, 25, 10, 9)
)

knitr::kable(train_df, booktabs = TRUE,
             align = c('r', 'r', 'r'),
             col.names = c('객체번호', '$x_1$', '$x_2$'),
             caption = 'DIANA 군집 대상 객체 데이터')
```

Table 12.6와 같이 두 변수 x_1, x_2 로 이루어진 7개의 객체 데이터에 대해 DIANA 방법에 의해 군집해를 아래와 같이 cluster 패키지의 diana 함수를 이용하여 간단히 구할 수 있다.

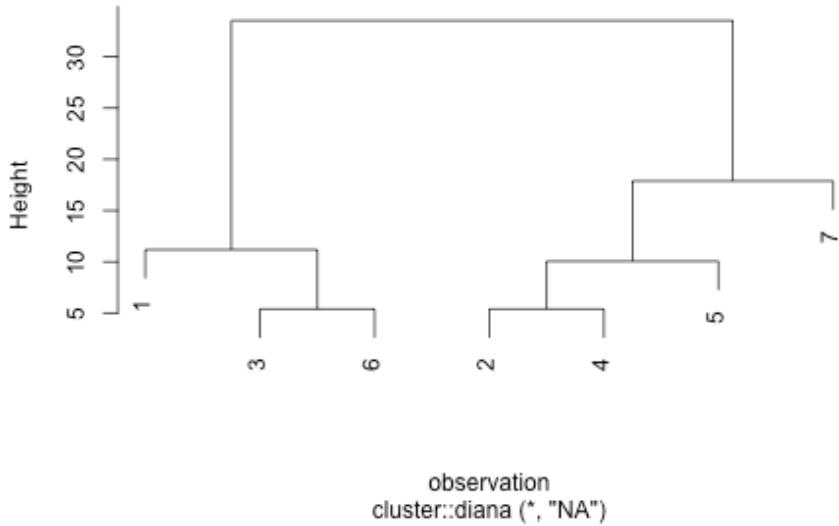


Figure 12.4: DIANA 방법에 의한 군집 텐드로그램

```
res_diana <- cluster::diana(train_df[, -1])
cluster::pltree(res_diana,
  main = NULL,
  xlab = "observation"
)
```

12.5.2 다이아나 알고리즘

가장 처음 이분화가 이루어질 때, 우선 타 객체와의 평균거리가 가장 큰 객체가 분파되어 새로운 군집을 형성한다. 그리고 다른 객체에 대하여, 군집에 남아있을 때의 평균거리와 새로운 군집으로 분리될 때의 평균거리를 산출하여, 현 군집에 잔류 또는 새로운 군집으로의 합류를 결정한다.

여기서 객체 i 와 군집 C 간의 평균거리는 다음과 같이 산출된다.

$$\bar{d}(i, C) = \begin{cases} \frac{1}{|C|-1} \sum_{j \in C} d(i, j) & \text{if } i \in C \\ \frac{1}{|C|} \sum_{j \in C} d(i, j) & \text{if } i \notin C \end{cases}$$

본 방법의 알고리즘은 다음과 같다.

1. 단계0: n 개의 객체를 하나의 군집으로 간주한다. ($k = 1$)
2. 단계1: 객체 간 거리가 가장 큰 두 객체를 포함한 군집을 이분화 대상으로 선정한다.
(이를 A 라 하고, $B \leftarrow \emptyset$ 로 둔다.)
3. 단계2: 다음 과정을 통하여 군집 A 를 이분화한다.
 1. 단계2-1: $i \leftarrow \arg \max_{i'} \bar{d}(i', A)$
 2. 단계2-2: $A \leftarrow A - \{i\}$, $B \leftarrow B \cup \{i\}$
 3. 단계2-3: $i \leftarrow \arg \max_{i' \in A} e(i') = \bar{d}(i', A) - \bar{d}(i', B)$
 4. 단계2-4: $e(i) > 0$ 이면 단계2-2로, $e(i) \leq 0$ 이면 단계3으로
4. 단계3
 1. $k \leftarrow k + 1$
 2. $k < n$ 이면 단계1로, $k = n$ 이면 Stop.

DIANA 알고리즘을 R script로 구현해보자.

우선, 단계1의 군집을 찾는 함수 `max_distance_cluster`를 구현하자. 이 함수는 아래 두 개의 데이터 프레임을 입력받는다.

- 입력
 - `df`: 관측 데이터. 각 열의 설명은 아래와 같다.
 - * `id`: 객체번호
 - * 나머지 열: 숫자형 변수
 - `cluster_label`: 각 객체의 현재 소속 군집을 나타내는 데이터 프레임
 - * `id`: 객체번호
 - * `cluster`: 군집명
- 함수값
 - `cluster`: 객체간 거리가 가장 큰 두 객체를 포함한 군집명
 - `distance`: 군집 내 객체간 최대 거리

```
max_distance_cluster <- function(df, cluster_label) {
  unique_cluster <- unique(cluster_label$cluster)

  cluster_df <- lapply(unique_cluster, function(x) {
    cluster_label %>%
      filter(cluster == x) %>%
      inner_join(df, by = "id") %>%
      select(-cluster, -id)
  })

  max_distance <- sapply(cluster_df,
    function(x) {
      if(nrow(x) == 1) return(0)
      max(dist(x))
    })
}
```

```

        }
    )

list(
  cluster = unique_cluster[which.max(max_distance)],
  distance = max(max_distance)
)
}
}

```

단계 2-1에서 군집 내 평균거리가 가장 큰 객체를 찾는 함수 `max_within_distance`를 아래와 같이 구현해보자. 이 때 입력변수인 `cluster_df`는 해당 군집의 객체 데이터로, 객체 번호를 나타내는 열 `id`와 객체의 각 숫자형 변수를 표현하는 열들로 구성된다.

```

max_within_distance <- function(cluster_df) {
  idx <- dist(subset(cluster_df, select = -id), upper = TRUE) %>%
    broom::tidy() %>%
    group_by(item1) %>%
    summarize(mean_distance = mean(distance)) %>%
    ungroup() %>%
    arrange(-mean_distance) %>%
    .[["item1"]] %>%
    .[1]

  cluster_df$id[idx]
}

```

이후 단계2-3에서 정의한 $e(i') = \bar{d}(i', A) - \bar{d}(i', B)$ 를 계산하는 함수 `e_score`를 아래와 같이 구현한다.

- `object`: 객체 번호(`id`)
- `A(B)`: 군집 $A(B)$ 의 객체 데이터. 행은 객체를 나타내며, `id` 열은 객체 번호, 이외의 열들은 변수를 나타낸다.

```

e_score <- function(object, A, B) {
  d_from_A <- proxy::dist(subset(A, id == object, -id),
                           subset(A, id != object, -id)) %>%
    mean()
  d_from_B <- proxy::dist(subset(A, id == object, -id),
                           B %>% select(-id)) %>%
    mean()
  return(d_from_A - d_from_B)
}

```

위 두 함수 `max_within_distance`와 `e_score`를 이용하여, 주어진 데이터 프레임을 두 군집으로 나누는 함수 `split_cluster`를 구현해보자.

- 입력: 객체 데이터를 나타내는 데이터 프레임 `cluster_df`. 행은 객체를 나타내며, 객체 번호를 나타내는 열 `id`와 객체의 각 숫자형 변수를 표현하는 열들로 구성된다.
- 함수값: 아래 두 개의 component를 지닌 리스트.
 - `idx_A`: 객체 데이터에서 행렬 `A`에 속하는 객체 번호
 - `idx_B`: 객체 데이터에서 행렬 `B`에 속하는 객체 번호

```
split_cluster <- function(cluster_df) {
  n <- nrow(cluster_df)

  idx_A <- cluster_df$id
  idx_B <- NULL

  # 단계2-1
  max_object <- max_within_distance(cluster_df)
  e_i <- Inf

  while(e_i > 0) {
    # 단계2-2
    idx_B <- c(idx_B, max_object)
    idx_A <- setdiff(idx_A, max_object)

    A <- cluster_df %>% filter(id %in% idx_A)
    B <- cluster_df %>% filter(id %in% idx_B)

    # 단계2-3
    if(nrow(A) > 1) {
      e_is <- sapply(A$id, function(x) e_score(x, A, B))
      max_object <- A$id[which.max(e_is)]
      e_i <- max(e_is)
    } else {
      e_i <- -Inf
    }
  }

  return(list(idx_A = idx_A, idx_B = idx_B))
}
```

단계1 함수 `max_distance_cluster`과 단계2 함수 `split_cluster`를 반복적으로 수행하며 각각의 객체가 군집에 될 때까지 군집을 분리해간다.

```
# 단계0
current_cluster <- tibble(
  id = train_df$id
)
current_cluster$cluster <- paste(1:nrow(current_cluster), collapse = ",")
```

```

i <- 0L
k <- 1L

while(k < nrow(train_df)) {
  i <- i + 1L

  # 단계1
  max_cluster <- max_distance_cluster(train_df, current_cluster)

  # 단계2
  new_split <- current_cluster %>%
    filter(cluster == max_cluster$cluster) %>%
    inner_join(train_df, by = "id") %>%
    select(-cluster) %>%
    split_cluster()

  # 군집해 업데이트
  current_cluster[
    current_cluster$id %in% new_split$idx_A,
    "cluster"] <- paste(new_split$idx_A, collapse = ", ")
  current_cluster[
    current_cluster$id %in% new_split$idx_B,
    "cluster"] <- paste(new_split$idx_B, collapse = ", ")

  # 군집해 출력
  k <- length(unique(current_cluster$cluster))
  cat("Iteration: ", i, ", k = ", k, ", clusters = ",
      paste0("{", unique(current_cluster$cluster), "}" ),
      ", height = ", max_cluster$distance, "\n")
}

## Iteration: 1 , k = 2 , clusters = {6,3,1} {2,4,5,7} , height = 33.54102
## Iteration: 2 , k = 3 , clusters = {6,3,1} {2,4,5} {7} , height = 17.88854
## Iteration: 3 , k = 4 , clusters = {1} {2,4,5} {3,6} {7} , height = 11.18034
## Iteration: 4 , k = 5 , clusters = {1} {2,4} {3,6} {5} {7} , height = 10.04988
## Iteration: 5 , k = 6 , clusters = {1} {2} {3,6} {4} {5} {7} , height = 5.385165
## Iteration: 6 , k = 7 , clusters = {1} {2} {3} {4} {5} {6} {7} , height = 5.385165

```

위 출력 결과에서 `height`는 해당 iteration에서 분리된 군집의 분리 전 지름(diameter)으로, 함수 `max_distance_cluster`에서 계산한 군집 내 객체간 최대 거리를 나타내며, 이는 R 패키지 `cluster`의 `diana` 함수 수행 시 함수값으로 출력되는 `height`값이다. Iteration이 진행됨에 따라 `height`의 값이 감소하는 것을 확인할 수 있다.

12.6 군집수의 결정

최적의 군집수를 결정하는 객관적인 방법은 존재하지 않는다. 계층적 군집방법에서는 텐드로그램을 참조하여 군집 간의 거리가 급격히 증가하는 계층에서 수평으로 절단하여, 그 이하의 그룹들을 하나의 군집으로 형성하는 방안을 널리 사용하고 있다. 이외에 군집수를 결정하는 데 통계량으로 다음과 같은 통계량들이 부수적으로 사용된다.

1. 새 군집의 RMS 표준편차(root-mean-square standard deviation of the new cluster; RMSSTD)

$$RMSSTD(C_i, C_j) = \sqrt{\frac{SS(C_i \cup C_j)}{p(|C_i| + |C_j| - 1)}}$$

2. Semipartial R-squared(SPR)

$$SPR(C_i, C_j) = \frac{SS(C_i \cup C_j) - (SS(C_i) + SS(C_j))}{SST}$$

where

$$SST = \sum_{i=1}^n \sum_{j=1}^p \left(x_{ji} - \frac{1}{n} \sum_{a=1}^n x_{ja} \right)^2$$

3. R-squared(R^2)

$$1 - \frac{\sum_{i=1}^k SS(C_i)}{SST}$$

위 12.4.2절에서 워드 군집 알고리즘으로 구현한 군집 과정에 대해 위 통계량을 계산해보자.

```
train_df <- tibble(
  id = c(1:8),
  x1 = c(4, 20, 3, 19, 17, 8, 19, 18),
  x2 = c(15, 13, 13, 4, 17, 11, 12, 6)
)

sst <- train_df %>%
  select(-id) %>%
  sapply(function(x) sum((x - mean(x))^2)) %>%
  sum()
```

```

#단계0
init_cluster <- tibble(
  id = train_df$id,
  cluster = as.character(1:nrow(train_df))
)
i <- 0L
current_clusters <- unique(init_cluster$cluster)
k <- length(current_clusters)
ssw <- calculate_ssw(train_df, init_cluster)
old_ssw <- NA_real_

#단계1
iteration <- vector("list", length = nrow(train_df) - 1)
while(k > 1) {
  i <- i + 1
  old_ssw <- ssw

  if(i == 1) {
    old_cluster <- init_cluster
  } else {
    old_cluster <- iteration[[i-1]]$new_cluster_label
  }

  iteration[[i]] <- best_merge_cluster(
    train_df,
    old_cluster
  )

  merged <- old_cluster %>%
    anti_join(iteration[[i]]$new_cluster_label, by = "cluster")

  current_clusters <- unique(iteration[[i]]$new_cluster_label$cluster)
  k <- length(current_clusters)
  ssw <- iteration[[i]]$new_ssw

  iteration[[i]]$rmsstd <- sqrt(
    merged %>%
      inner_join(train_df, by = "id") %>%
      select(-id, -cluster) %>%
      sapply(function(x) sum((x - mean(x))^2)) %>%
      sum() / (2 * (nrow(merged) - 1)))
  )

  iteration[[i]]$iter <- i
  iteration[[i]]$merge <- paste0("{", unique(merged$cluster), "}", collapse = ", ")
}

```

Table 12.7: 군집 과정에 따른 여러 통계량

Iteration	통합대상군집	통합 후 군집	\$RMSSTD\$	\$SPR\$
0	NA	{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}	NA	NA 1.
1	{2}, {7}	{1}, {2,7}, {3}, {4}, {5}, {6}, {8}	0.7071068	0.0020005 0.
2	{1}, {3}	{1,3}, {2,7}, {4}, {5}, {6}, {8}	1.1180340	0.0050013 0.
3	{4}, {8}	{1,3}, {2,7}, {4,8}, {5}, {6}	1.1180340	0.0050013 0.
4	{2,7}, {5}	{1,3}, {2,7,5}, {4,8}, {6}	2.1602469	0.0353422 0.
5	{1,3}, {6}	{1,3,6}, {2,7,5}, {4,8}	2.3452079	0.0390098 0.
6	{2,7,5}, {4,8}	{1,3,6}, {2,7,5,4,8}	3.8470768	0.1945153 0.
7	{1,3,6}, {2,7,5,4,8}	{1,3,6,2,7,5,4,8}	5.9753960	0.7191298 0.

```

iteration[[i]]$sol <- paste0("{", unique(current_clusters), "}", collapse = ", ")
iteration[[i]]$spr <- (ssw - old_ssw) / sst
iteration[[i]]$r_sq <- 1 - ssw / sst
}

cluster_statistic <- lapply(iteration, function(x) x[
  c("iter", "merge", "sol", "rmsstd", "spr", "r_sq")]) %>%
  bind_rows(
    tibble(
      iter = 0,
      sol = paste0("{", unique(init_cluster$cluster), "}", collapse = ", "),
      r_sq = 1
    )
  ) %>%
  arrange(iter)

cluster_statistic %>%
  knitr::kable(
    booktabs = TRUE,
    col.names = c('Iteration', '통합대상군집', '통합 후 군집',
                 '$RMSSTD$', '$SPR$', '$R^2$'),
    caption = '군집 과정에 따른 여러 통계량'
  )

cluster_statistic %>%
  mutate(
    rmsstd = if_else(is.na(rmsstd), 0, rmsstd),
    spr = if_else(is.na(spr), 0, spr)
  ) %>%
  ggplot(aes(x = iter)) +
  geom_line(aes(y = rmsstd, color = "RMSSTD")) +

```

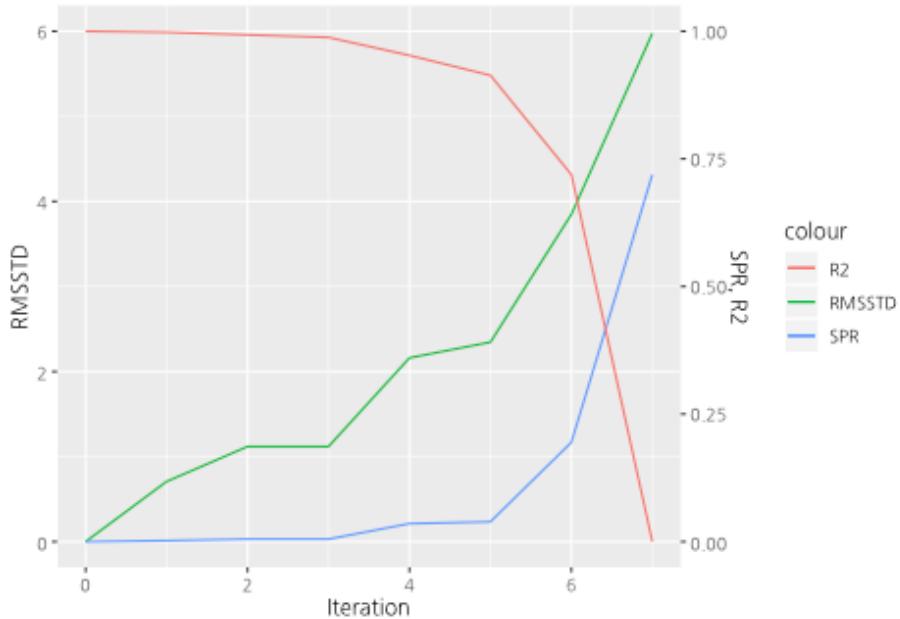


Figure 12.5: 군집 과정에 따른 통계량 추이

```

geom_line(aes(y = spr * 6, color = "SPR")) +
  geom_line(aes(y = r_sq * 6, color = "R2")) +
  scale_y_continuous(sec.axis = sec_axis(~ . / 6, name = "SPR, R2")) +
  ylab("RMSSTD") +
  xlab("Iteration")

```

그림 12.5에서 보듯이 Iteration 6부터 3가지 통계량 모두 급격하게 변화하는 것을 알 수 있다. 따라서 군집수는 Iteration 5까지 3개가 가장 적당하다고 하겠다.

Chapter 13

비계층적 군집방법

비계층적 군집방법(Nonhierarchical clustering)은 분할방법(Partitioning method)이라고도 하는데, 군집의 수 K 를 사전에 지정하고 대상 객체들을 적절한 군집에 배정하는 방법이다. 즉, 이 방법은 n 개의 객체를 K 개의 군집에 할당하는 최적화 문제로 간주할 수 있다. 본 장에서는 분할방법의 대표적인 K-means 알고리즘, K-medoids 군집방법, 퍼지 K-means 알고리즘, 그리고 모형기반 군집방법에 대하여 주로 알아본다.

13.1 필요 R package 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
stats	3.6.0
cluster	2.0.8
flexclust	1.4-0
mclust	5.4.3
mvtnorm	1.0-10

13.2 K-means 알고리즘

K-means 알고리즘은 비계층적 군집방법 중 가장 널리 사용되는 것으로 K 개 군집의 중심좌표를 고려하여 각 객체를 가까운 군집에 배정하는 반복적 알고리즘이다.

13.2.1 기본 R 스크립트

10명에 대한 PC의 사용경력(x_1)과 주당 사용시간(x_2)이 다음과 같다.

Table 13.1: PC 사용 데이터

객체번호	사용경력 (\$x_1\$)	사용시간 (\$x_2\$)
1	6	14
2	8	13
3	14	6
4	11	8
5	15	7
6	7	15
7	13	6
8	5	4
9	3	3
10	3	2

```

df <- tribble(
  ~id, ~x1, ~x2,
  1, 6, 14,
  2, 8, 13,
  3, 14, 6,
  4, 11, 8,
  5, 15, 7,
  6, 7, 15,
  7, 13, 6,
  8, 5, 4,
  9, 3, 3,
  10, 3, 2
)

df %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c(
      '객체번호',
      '사용경력($x_1$)', '사용시간($x_2$)'
    ),
    caption = 'PC 사용 데이터'
)

```

아래와 같이 `stats` 패키지의 `kmeans` 함수를 이용하여 K-means 알고리즘 수행 결과를 얻을 수 있다. 아래 스크립트는 군집 수가 $K = 3$ 라 가정하여 수행한 예이다.

```
set.seed(123)
kmeans_solution <- kmeans(x = df[, -1], centers = 3)
```

위 스크립트 실행 결과 도출된 군집 중심좌표는 위에서 얻어진 `kmeans` 클래스 객체의 `centers`값에 저장된다.

```
kmeans_solution$centers
```

```
##          x1      x2
## 1 13.250000  6.75
## 2 3.666667  3.00
## 3 7.000000 14.00
```

또한 각 학습 데이터가 속한 군집은 `cluster`값에 저장된다.

```
kmeans_solution$cluster
```

```
## [1] 3 3 1 1 1 3 1 2 2 2
```

객체의 군집결과는 아래와 같이 도식화하여 보일 수 있다.

```
df %>%
  mutate(cluster = as.factor(kmeans_solution$cluster)) %>%
  ggplot(aes(x = x1, y = x2)) +
  geom_text(aes(label = id, color = cluster))
```

13.2.2 알고리즘

K-means 알고리즘의 구체적 절차는 아래와 같다. Table 13.1에 대한 각 단계의 결과를 함께 살펴보자.

[단계 0] (초기 객체 선정) 어떤 규칙에 의하여 K 개의 객체의 좌표를 초기 군집의 중심좌표(centroid)로 선정한다. 군집 j 의 중심좌표를 $\mathbf{c}_j = \left(\bar{x}_1^{(j)}, \dots, \bar{x}_p^{(j)}\right)^T$ 라 하자. 초기 군집 중심좌표 $\mathbf{c}_1, \dots, \mathbf{c}_K$ 를 선정하는 방법은 예를 들어 다음과 같은 규칙이 사용된다.

- 무작위 방법: 대상 객체 중 무작위로 K 개를 선정한다.
- 외각 객체 선정: 전체 객체의 중심좌표에서 가장 멀리 위치하는 K 개의 객체를 선정한다.

[단계 0] 무작위 방법을 이용하여 Table 13.1으로부터 3개의 객체를 초기 군집 중심좌표로 선정하자.

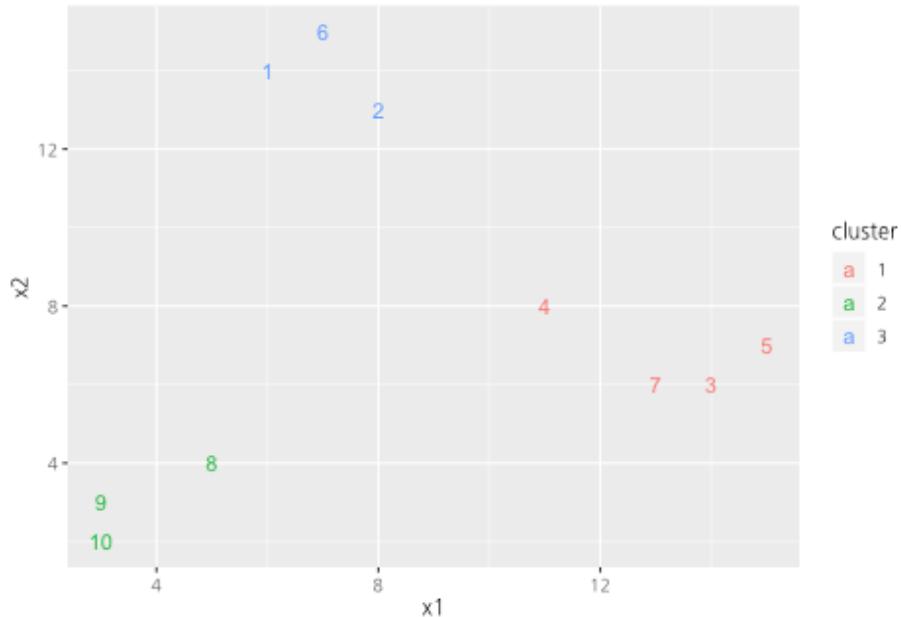


Figure 13.1: K-means 수행 결과

```
set.seed(123)

init_cluster <- function(df, k = 1) {
  k <- min(k, nrow(df))

  df %>%
    sample_n(k)
}

cluster_df <- init_cluster(df[, -1], 3)

cluster_df

## # A tibble: 3 x 2
##       x1     x2
##   <dbl> <dbl>
## 1     14     6
## 2      3     2
## 3      8    13
```

[단계 1] (객체의 군집 배정) 각 객체에 대하여 K 개의 군집 중심좌표(centroid)와의 거리 (주로 유clidean 거리 사용)를 산출한 후 가장 가까운 군집에 그 객체를 배정한다.

$$a_{ij} = \begin{cases} 1 & \text{if } j = \arg \max_k d(\mathbf{x}_i, \mathbf{c}_k) \\ 0 & \text{otherwise} \end{cases}, i = 1, \dots, n, j = 1, \dots, K$$

각 객체에 대하여 3개의 군집 중심좌표와의 거리를 산출해보면 아래와 같다.

```
flexclust::dist2(df[, -1], cluster_df)
```

```
##          [,1]      [,2]      [,3]
## [1,] 11.313708 12.369317 2.236068
## [2,] 9.219544 12.083046 0.000000
## [3,] 0.000000 11.704700 9.219544
## [4,] 3.605551 10.000000 5.830952
## [5,] 1.414214 13.000000 9.219544
## [6,] 11.401754 13.601471 2.236068
## [7,] 1.000000 10.770330 8.602325
## [8,] 9.219544 2.828427 9.486833
## [9,] 11.401754 1.000000 11.180340
## [10,] 11.704700 0.000000 12.083046
```

이에 각 객체들에 대해 거리가 가장 가까운 군집에 그 객체를 배정한다.

```
assign_cluster <- function(df, cluster_df) {
  cluster_ind <- flexclust::dist2(df, cluster_df) %>%
    apply(1, which.min)

  map(unique(cluster_ind), ~which(cluster_ind == .))
}

cluster_objects <- assign_cluster(df[, -1], cluster_df)

cluster_objects

## [[1]]
## [1] 1 2 6
##
## [[2]]
## [1] 3 4 5 7
##
## [[3]]
## [1] 8 9 10
```

[단계 2] (군집 중심좌표의 산출) 새로운 군집에 대한 중심좌표를 산출한다.

$$\bar{x}_l^{(j)} = \frac{\sum_i a_{ij}x_{li}}{\sum_i a_{ij}}, l = 1, \dots, p, j = 1, \dots, K$$

```

find_center <- function(df, cluster) {
  map_dfr(cluster, ~df[., ] %>% summarize_all(mean))
}

new_cluster_df <- find_center(df[, -1], cluster_objects)

new_cluster_df

## # A tibble: 3 x 2
##       x1     x2
##   <dbl> <dbl>
## 1     7    14
## 2  13.2   6.75
## 3   3.67    3

```

[단계 3] (수렴 조건 점검) 새로 산출된 중심좌표값과 이전 좌표값을 비교하여 수렴 조건 내에 들면 마치며, 그렇지 않으면 단계 1을 반복한다.

```
identical(cluster_df, new_cluster_df)
```

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

위의 경우, 군집 중심좌표가 다르므로 다음 iteration을 진행한다.

13.2.3 R 스크립트 구현

위의 과정을 군집해가 수렴할 때까지 반복하도록 아래와 같이 R 스크립트를 구현해보자.

```

init_cluster <- function(df, k = 1) {
  k <- min(k, nrow(df))

  df %>% sample_n(k)
}

assign_cluster <- function(df, cluster_df) {
  cluster_ind <- flexclust:::dist2(df, cluster_df) %>%
    apply(1, which.min)

  map(unique(cluster_ind), ~which(cluster_ind == .))
}

```

```

}

find_center <- function(df, cluster) {
  map_dfr(cluster, ~df[., ] %>% summarize_all(mean))
}

kmeans_cluster <- function(df, k = 1, verbose = FALSE) {
  k <- min(k, nrow(df))

  i <- 0L

  ## 단계 0
  cluster_df <- init_cluster(df, k)

  while(TRUE) {
    i <- i + 1L

    ## 단계 1
    cluster_objects <- assign_cluster(df, cluster_df)
    if (verbose) { # 군집해 출력
      cat("Iteration", i, ":", 
          map(cluster_objects, ~ str_c("{", str_c(., collapse = ", "), "}")) %>%
            str_c(collapse = ", "),
          "\n")
    }
  }

  ## 단계 2
  new_cluster_df <- find_center(df, cluster_objects)

  ## 단계 3
  if(identical(cluster_df, new_cluster_df)) break

  cluster_df <- new_cluster_df
}

res <- list(
  cluster_centers = cluster_df,
  assigned_objects = cluster_objects,
  n_iteration = i
)

return (res)
}

```

```
set.seed(123)

kmeans_solution <- kmeans_cluster(df[, -1], k = 3, verbose = TRUE)

## Iteration 1 : {1, 2, 6}, {3, 4, 5, 7}, {8, 9, 10}
## Iteration 2 : {1, 2, 6}, {3, 4, 5, 7}, {8, 9, 10}
```

위와 같이 2번째 Iteration에서 군집해가 수렴하였으며, 최종 군집해는 아래와 같다.

```
kmeans_solution$assigned_objects
```

```
## [[1]]
## [1] 1 2 6
##
## [[2]]
## [1] 3 4 5 7
##
## [[3]]
## [1] 8 9 10
```

13.3 K-medoids 군집방법

K-means 알고리즘에서는 각 군집의 중심좌표(centroid)를 군집 중심으로 고려하고 있는 반면, K-medoids 군집방법에서는 각 군집의 대표객체를 군집 중심으로 고려한다.

K-medoids 군집방법의 알고리즘으로 잘 알려진 것에는 다음과 같은 것들이 있다.

1. PAM(Partitioning Around Medoids)
2. CLARA(Clustering LARge Applications)
3. CLARANS(Clustering Large Applications based on RANdomized Search)
4. K-means-like 알고리즘

13.3.1 PAM 알고리즘

PAM 알고리즘은 Kaufman and Rousseeuw (1990)에 의하여 발표된 것으로, 초기 대표객체를 선정하는 방법인 **BUILD**와 더 나은 군집해를 찾아나가는 과정인 **SWAP**의 두 부분으로 구성되어 있다.

13.3.1.1 기본 R 스크립트

Table 13.2: PC 사용자 데이터

객체번호	사용경력(\$x_1\$)	사용시간(\$x_2\$)
1	3	3
2	5	4
3	11	8
4	13	6
5	14	6
6	15	7

```
df <- tribble(
  ~id, ~x1, ~x2,
  1, 3, 3,
  2, 5, 4,
  3, 11, 8,
  4, 13, 6,
  5, 14, 6,
  6, 15, 7
)

df %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c(
      '객체번호',
      '사용경력($x_1$)', '사용시간($x_2$)'
    ),
    caption = 'PC 사용자 데이터'
  )
```

PAM 알고리즘은 `cluster` 패키지 내의 함수 `pam`을 이용하여 아래와 같이 간단하게 실행할 수 있다.

```
pam_solution <- cluster::pam(df[, -1], k = 2)
```

얻어진 `pam` 객체의 원소 `id.med`는 몇 번째 객체가 군집의 대표객체로 선정되었는지를 나타낸다.

```
pam_solution$id.med
```

```
## [1] 2 5
```

또한 pam 객체의 원소 clustering은 각 객체가 어떠한 군집에 할당되었는지를 보여준다.

```
pam_solution$clustering
```

```
## [1] 1 1 2 2 2 2
```

13.3.1.2 PAM 알고리즘

PAM 알고리즘의 각 단계를 Table 13.2의 예제 데이터에 적용하여 살펴보기로 하자.

13.3.1.2.1 BUILD

BUILD는 다음 절차들을 거쳐서 K 개의 초기 대표객체를 구하는 과정이다.

[단계 0] 우선 각 객체별로 다른 객체 간의 거리를 구한 후, 그 합이 가장 작은 객체 하나를 대표객체로 선정한다. 선정된 대표객체집합을 M 이라 하자.

```
pairwise_distance_df <- dist(df[, -1], upper = TRUE) %>%
  broom::tidy()

M_idx <- pairwise_distance_df %>%
  group_by(item1) %>%
  summarize(sum_distance = sum(distance)) %>%
  top_n(-1, sum_distance) %>%
  .\$item1

M <- df[M_idx, ]
```

```
## # A tibble: 1 x 3
##       id     x1     x2
##   <dbl> <dbl> <dbl>
## 1     4     13     6
```

[단계 1] 대표객체로 선정되지 않은 객체 j 에 대하여, 이전에 대표객체로 선정된 객체들 중 객체 j 에 가장 가까운 거리 D_j 를 구한다. 즉,

$$D_j = \min_{k \in M} d(j, k), j \notin M$$

```
D_j <- pairwise_distance_df %>%
  filter(
    !item1 %in% M$id,
    item2 %in% M$id
  ) %>%
  group_by(item1) %>%
  top_n(-1, distance) %>%
  ungroup() %>%
  rename(D_j = distance) %>%
  select(-item2)

D_j

## # A tibble: 5 x 2
##   item1     D_j
##   <int> <dbl>
## 1     1  10.4
## 2     2   8.25
## 3     3   2.83
## 4     5   1
## 5     6   2.24
```

그리고 대표객체로 선정되지 않은 두 객체 i, j 에 대하여 다음을 산출한다.

$$C_{ji} = \max(D_j - d(j, i), 0)$$

```
d_ji <- pairwise_distance_df %>%
  filter(
    !item1 %in% M$id,
    !item2 %in% M$id
  )

C_ji <- D_j %>%
  inner_join(d_ji, by = "item1") %>%
  mutate(C_ji = pmax(D_j - distance, 0))

C_ji
```

```
## # A tibble: 20 x 5
##   item1     D_j item2 distance   C_ji
##   <int> <dbl> <int>     <dbl> <dbl>
## 1     1  10.4      2     2.24  8.20
## 2     1  10.4      3     9.43  1.01
```

```

##  3   1 10.4    5   11.4  0
##  4   1 10.4    6   12.6  0
##  5   2 8.25     1   2.24 6.01
##  6   2 8.25     3   7.21 1.04
##  7   2 8.25     5   9.22 0
##  8   2 8.25     6   10.4  0
##  9   3 2.83     1   9.43 0
## 10   3 2.83     2   7.21 0
## 11   3 2.83     5   3.61 0
## 12   3 2.83     6   4.12 0
## 13   5 1         1   11.4  0
## 14   5 1         2   9.22 0
## 15   5 1         3   3.61 0
## 16   5 1         6   1.41 0
## 17   6 2.24     1   12.6  0
## 18   6 2.24     2   10.4  0
## 19   6 2.24     3   4.12 0
## 20   6 2.24     5   1.41 0.822

```

이는 객체 i 가 추가로 대표객체가 된다고 할 때, 객체 j 의 입장에서 거리 감소량이다.

[단계 2] 다음과 같이 거리감소량이 가장 큰 객체 m 을 대표객체에 포함시키고,

$$m = \arg \max_{i \notin M} \sum_{j \notin M} C_{ji}$$

```

m <- C_ji %>%
  group_by(item2) %>%
  summarize(sum_C_ji = sum(C_ji)) %>%
  top_n(1, sum_C_ji) %>%
  .$item2

```

m

```
## [1] 2
```

대표객체집합을 수정한다.

$$M \leftarrow M \cup \{m\}$$

```
M <- M %>%
  bind_rows(df[m, ])
```

[단계 3] K 개의 대표객체가 선정되었으면 Stop, 그렇지 않은 경우에는 [단계 1]로 되돌아간다.

13.3.1.2.2 SWAP

SWAP은 대표객체로 선정되어 있는 객체 i 와 선정되지 않은 객체 h 를 교환할 때 목적함수의 변화량을 산출해 더 나은 목적함수값을 찾아가는 과정이다.

[단계 1] 객체 i 와 객체 h 를 교환할 때 목적함수의 변화량을 산출하기 위하여 우선, 대표 객체로 선정되지 않은 임의의 객체 $j \neq h$ 에서의 변화량을 다음과 같이 산출한다.

$$\begin{aligned} C_{jih} &= (i \text{와 } h \text{를 교환 후 객체 } j \text{와 대표객체와의 거리}) \\ &\quad - (\text{교환 전 객체 } j \text{와 대표객체와의 거리}), \\ &(j \notin M, i \in M, h \notin M) \end{aligned}$$

```
# 교환 전 각 객체와 대표 객체와의 거리
D_j <- pairwise_distance_df %>%
  filter(
    !item1 %in% M$id,
    item2 %in% M$id
  ) %>%
  group_by(item1) %>%
  top_n(-1, distance) %>%
  ungroup() %>%
  rename(j = item1) %>%
  select(-item2)

D_j
```

```
## # A tibble: 4 x 2
##       j   distance
##   <int>     <dbl>
## 1     1     2.24
## 2     3     2.83
## 3     5     1
## 4     6     2.24
```

```
# 교환할 객체
swap_ids <- tibble(
  i = rep(M$id, each = nrow(df) - nrow(M)),
  h = rep(setdiff(df$id, M$id), nrow(M))
)

# 교환 후 각 객체와 대표 객체와의 거리
update_distance <- function(i, h, distance_df, center_ids) {
  new_center_ids <- c(setdiff(center_ids, i), h)
```

```

res <- distance_df %>%
  filter(
    !item1 %in% c(center_ids, h),
    item2 %in% new_center_ids
  ) %>%
  group_by(item1) %>%
  top_n(-1, distance) %>%
  ungroup() %>%
  rename(j = item1) %>%
  select(-item2) %>%
  mutate(
    i = i,
    h = h
  )

  return(res)
}

D_jih <- pmap_dfr(
  swap_ids,
  update_distance,
  distance_df = pairwise_distance_df,
  center_ids = M$id
)

D_jih

## # A tibble: 24 x 4
##       j distance     i     h
##   <int>    <dbl> <dbl> <dbl>
## 1     3    7.21     4     1
## 2     5    9.22     4     1
## 3     6   10.4      4     1
## 4     1    2.24     4     3
## 5     5    3.61     4     3
## 6     6    4.12     4     3
## 7     1    2.24     4     5
## 8     3    3.61     4     5
## 9     6    1.41     4     5
## 10    1    2.24     4     6
## # ... with 14 more rows

# 거리의 변화량
C_jih <- D_j %>%
  inner_join(D_jih, by = "j", suffix = c("", "_new")) %>%

```

```

  mutate(diff_distance = distance_new - distance)

C_jih

## # A tibble: 24 x 6
##       j distance distance_new     i     h diff_distance
##   <int>    <dbl>      <dbl> <dbl> <dbl>      <dbl>
## 1     1     2.24      2.24     4     3        0
## 2     1     2.24      2.24     4     5        0
## 3     1     2.24      2.24     4     6        0
## 4     1     2.24      9.43     2     3      7.20
## 5     1     2.24     10.4      2     5      8.20
## 6     1     2.24     10.4      2     6      8.20
## 7     3     2.83      7.21     4     1      4.38
## 8     3     2.83      3.61     4     5      0.777
## 9     3     2.83      4.12     4     6      1.29
## 10    3     2.83      2.83     2     1        0
## # ... with 14 more rows

```

[단계 2] 대표객체 i 를 h 로 교환하는 경우 총 변화량은 다음과 같다.

$$T_{ih} = \sum_j C_{jih}$$

```

T_ih <- C_jih %>%
  group_by(i, h) %>%
  summarize(total_diff_distance = sum(diff_distance))

T_ih

```

```

## # A tibble: 8 x 3
## # Groups:   i [2]
##       i     h total_diff_distance
##   <dbl> <dbl>      <dbl>
## 1     2     1        0
## 2     2     3      7.20
## 3     2     5      7.38
## 4     2     6      8.20
## 5     4     1     20.8
## 6     4     3      4.49
## 7     4     5     -0.0447
## 8     4     6      1.71

```

○ 때 $\min_{i,h} T_{ih}$ 에 대응하는 객체 i^* 와 h^* 를 찾아 $T_{i^*h^*} < 0$ 이면 교환한 후에 다시 [단계 1]으로 돌아가고, $T_{i^*h^*} \geq 0$ 이면 교환하지 않고 stop.

```
swap_ih <- T_ih %>%
  filter(total_diff_distance < 0) %>%
  top_n(-1, total_diff_distance)

swap_ih

## # A tibble: 1 x 3
## # Groups:   i [1]
##       i     h total_diff_distance
##   <dbl> <dbl>             <dbl>
## 1     4     5            -0.0447
```

본 예제의 경우 객체 4 대신 객체 5가 새로운 대표객체로 선택되고, 다시 [단계 1]로 넘어간다.

```
M <- M %>%
  anti_join(df[swap_ih$i, ], ) %>%
  bind_rows(df[swap_ih$h, ], )

## Joining, by = c("id", "x1", "x2")
```

M

```
## # A tibble: 2 x 3
##       id     x1     x2
##   <dbl> <dbl> <dbl>
## 1     2     5     4
## 2     5    14     6
```

SWAP 과정이 종료된 후, 최종 군집해는 각 객체를 가장 가까운 대표객체가 속한 군집에 할당함으로써 얻어진다.

13.3.1.3 R 스크립트 구현

일련의 과정을 함수로 구현해보자.

```
# 각 객체로부터 가장 가까운 대표객체까지의 거리
distance_from_medoids <- function(distance_df, medoids_ids) {
  distance_df %>%
```

```

filter(
  !item1 %in% medoids_ids,
  item2 %in% medoids_ids
) %>%
group_by(item1) %>%
arrange(distance) %>%
slice(1) %>%
ungroup() %>%
rename(j = item1) %>%
select(-item2)
}

# k개의 초기 대표객체를 선정
build_medoids <- function(distance_df, k = 1L) {
  k <- min(k, length(unique(distance_df$item1)))

  # 첫 번째 대표객체 선정
  medoids_ids <- distance_df %>%
    group_by(item1) %>%
    summarize(sum_distance = sum(distance)) %>%
    arrange(sum_distance) %>%
    slice(1) %>%
    .$item1

  while (length(medoids_ids) < k) {
    D_j <- distance_from_medoids(distance_df, medoids_ids)

    d_ji <- distance_df %>%
      filter(
        !item1 %in% medoids_ids,
        !item2 %in% medoids_ids
      ) %>%
      rename(j = item1, i = item2)

    # 거리 감소량
    C_ji <- D_j %>%
      inner_join(d_ji, by = "j",
                 suffix = c("", "_new")) %>%
      mutate(diff_distance = pmax(distance - distance_new, 0))

    # 거리 감소량이 가장 큰 객체 선택
    m <- C_ji %>%
      group_by(i) %>%
      summarize(total_diff_distance = sum(diff_distance)) %>%
      arrange(desc(total_diff_distance)) %>%
      slice(1)
    medoids_ids <- bind_rows(medoids_ids, m)
  }
}

```

```

    slice(1) %>%
      .\$i

    # 대표객체에 추가
    medoids_ids <- c(medoids_ids, m)
  }

  return(medoids_ids)
}

# 교환 후 각 객체와 대표 객체와의 거리
update_distance <- function(i, h, distance_df, medoids_ids) {
  new_medoids_ids <- c(setdiff(medoids_ids, i), h)

  res <- distance_from_medoids(distance_df, new_medoids_ids) %>%
    filter(j != h) %>%
    mutate(
      i = i,
      h = h
    )

  return(res)
}

# 교환할 medoid 선택
swap_medoids <- function(distance_df, medoids_ids) {
  observation_ids <- unique(distance_df$item1)

  # 교환 전 각 객체와 대표 객체와의 거리
  D_j <- distance_from_medoids(distance_df, medoids_ids)

  # 교환할 객체
  swap_ids <- tibble(
    i = rep(medoids_ids,
            each = length(observation_ids) - length(medoids_ids)),
    h = rep(setdiff(observation_ids, medoids_ids),
            length(medoids_ids))
  )

  D_jih <- pmap_dfr(
    swap_ids,
    update_distance,
    distance_df = distance_df,
    medoids_ids = medoids_ids
  )
}

```

```

# 거리의 변화량
C_jih <- D_j %>%
  inner_join(D_jih, by = "j", suffix = c("", "_new")) %>%
  mutate(diff_distance = distance_new - distance)

T_ih <- C_jih %>%
  group_by(i, h) %>%
  summarize(total_diff_distance = sum(diff_distance))

swap_ih <- T_ih %>%
  filter(total_diff_distance < 0) %>%
  arrange(total_diff_distance) %>%
  slice(1)

return(list(remove = swap_ih$i, add = swap_ih$h))
}

# 전체 PAM 알고리즘
pam_medoids <- function(distance_df, k = 1L) {
  # BUILD
  medoids_ids <- build_medoids(distance_df, k)
  k <- length(medoids_ids)

  # SWAP
  while (TRUE) {
    swap_medoids <- swap_medoids(distance_df, medoids_ids)
    if (is_empty(swap_medoids$remove)) {
      break
    } else {
      medoids_ids <- c(
        setdiff(medoids_ids, swap_medoids$remove),
        swap_medoids$add
      )
    }
  }

  return (medoids_ids)
}

```

위 구현한 함수를 이용하여 아래와 같이 PAM을 실행해보자.

```

pairwise_distance_df <- dist(df[, -1], upper = TRUE) %>%
  broom::tidy()

medoids_ids <- pam_medoids(pairwise_distance_df, k = 2)

```

```
df[medoids_ids, ]
```

```
## # A tibble: 2 x 3
##   id     x1     x2
##   <dbl> <dbl> <dbl>
## 1     2     5     4
## 2     5    14     6
```

위와 같이 2개의 대표객체 2, 5가 선정된다.

최종 군집해는 각 객체를 가장 가까운 대표객체가 속한 군집에 할당함으로써 얻어진다.

```
assign_cluster <- function(distance_df, medoids_ids) {
  distance_df %>%
    filter(
      !item1 %in% medoids_ids,
      item2 %in% medoids_ids
    ) %>%
    group_by(item1) %>%
    arrange(distance) %>%
    slice(1) %>%
    ungroup() %>%
    bind_rows(
      tibble(
        item1 = medoids_ids,
        item2 = medoids_ids,
        distance = 0
      )
    ) %>%
    rename(object = item1) %>%
    arrange(object) %>%
    group_by(item2) %>%
    mutate(cluster = str_c("{", str_c(object, collapse = ", "), "}")]
    ungroup() %>%
    select(-item2)
  }

  assign_cluster(pairwise_distance_df, medoids_ids)

## # A tibble: 6 x 3
##   object distance cluster
##   <int>     <dbl> <chr>
## 1     1     2.24 {1, 2}
## 2     2       0     {1, 2}
```

```
## 3      3      3.61 {3, 4, 5, 6}
## 4      4      1     {3, 4, 5, 6}
## 5      5      0     {3, 4, 5, 6}
## 6      6      1.41 {3, 4, 5, 6}
```

13.3.2 CLARA 알고리즘

PAM 알고리즘은 SWAP 부분에서 모든 가능한 경우를 고려하기 때문에, 전체 객체 수가 많은 경우 계산 시간이 매우 길다는 단점이 있다. 이를 보완하기 위해 CLARA는 적절한 수의 객체를 샘플링한 후 이들에 대해 PAM 알고리즘을 적용하여 중심객체를 선정하는 방법이다. 이러한 샘플링을 여러 번 한 후, 이 중 가장 좋은 결과를 택하는 것인데, 반복수는 5번으로 충분한 것으로 분석되고 있다.

자세한 내용은 교재 (전치혁, 2012) 참조

13.3.2.1 기본 R 스크립트

```
clara_solution <- cluster::clara(df[, -1], k = 2)
```

얻어진 clara 객체의 원소 `i.med`는 몇 번째 객체가 군집의 대표객체로 선정되었는지를 나타낸다.

```
clara_solution$i.med
```

```
## [1] 2 5
```

또한 clara 객체의 원소 `clustering`은 각 객체가 어떠한 군집에 할당되었는지를 보여 준다.

```
clara_solution$clustering
```

```
## [1] 1 1 2 2 2 2
```

13.3.3 CLARANS 알고리즘

교재 (전치혁, 2012) 참조

13.3.4 K-means-like 알고리즘

본 알고리즘은 PAM 알고리즘의 단점을 보완하고자 Park and Jun (2009)에 의해 제안된 것으로, 대표 객체를 반복적으로 수정하는데 K-means 알고리즘의 작동 원리를 모방한 K-medoids 군집 방법이다. 따라서 간단하며 계산 시간이 빠른 것이 장점이라 하겠다. 이 알고리즘은 다음과 같이 3단계로 구성되어 있다. 알고리즘의 각 단계를 Table 13.2의 예제 데이터에 적용하여 살펴보기로 하자.

[단계 1] (초기 대표 객체 선정) K 개의 초기 대표 객체를 선정하며, 각 객체를 가장 가까운 대표 객체에 배정하여 초기 군집해를 얻는다.

객체들 간의 거리 $d(i, j)$, $i, j = 1, \dots, n$ 를 구한다.

```
pairwise_distance_df <- dist(df[, -1], upper = TRUE) %>%
  broom::tidy()
```

각 객체 $j = 1, \dots, n$ 에 대하여 다음을 산출한다.

$$v_j = \sum_{i=1}^n \frac{d(i, j)}{\sum_{k=1}^n d(i, k)}$$

```
v_j <- pairwise_distance_df %>%
  group_by(item2) %>%
  mutate(prop = distance / sum(distance)) %>%
  ungroup() %>%
  group_by(item1) %>%
  summarize(sum_prop = sum(prop)) %>%
  rename(j = item1)

v_j
```

```
## # A tibble: 6 x 2
##       j   sum_prop
##   <int>     <dbl>
## 1     1     1.67
## 2     2     1.33
## 3     3     0.781
## 4     4     0.661
## 5     5     0.713
## 6     6     0.849
```

이후 v_j 값들을 오름차순으로 정렬하여 가장 작은 K 개의 값을 초기 대표 객체로 선정한다. 본 예에서는 $K = 2$ 로 가정하자.

```
medoids_ids <- v_j %>%
  arrange(sum_prop) %>%
  slice(1:2) %>%
  .\$j

medoids_ids
```

```
## [1] 4 5
```

이후 객체를 배정하여 군집해를 얻는다. 위에서 정의했던 `assign_cluster` 함수를 재사용하자.

```
cluster_solution <- assign_cluster(
  pairwise_distance_df,
  medoids_ids
)

cluster_solution
```

```
## # A tibble: 6 x 3
##   object distance cluster
##     <int>    <dbl> <chr>
## 1       1     10.4 {1, 2, 3, 4}
## 2       2      8.25 {1, 2, 3, 4}
## 3       3      2.83 {1, 2, 3, 4}
## 4       4       0    {1, 2, 3, 4}
## 5       5       0    {5, 6}
## 6       6     1.41 {5, 6}
```

[단계 2] (대표객체의 수정) 현재의 군집에 배정된 객체들의 대표객체를 구하여 새로운 대표객체로 삼는다. 새로운 대표객체는 같은 군집에 배정된 다른 객체들로부터의 거리의 합이 최소가 되는 객체이다.

```
find_medoids <- function(distance_df, ids) {
  distance_df %>%
    filter(
      item1 %in% ids,
      item2 %in% ids
    ) %>%
    group_by(item1) %>%
    summarize(total_distance = sum(distance)) %>%
    arrange(total_distance) %>%
    slice(1) %>%
```

```

    .\$item1
}

cluster_objects <- cluster_solution %>%
  split(.\$cluster) %>%
  map(~ .\$object)

medoids_ids <- map_int(
  cluster_objects,
  find_medoids,
  distance_df = pairwise_distance_df
)

medoids_ids

## {1, 2, 3, 4}      {5, 6}
##             2          5

```

[단계 3] (객체의 배정) 각 객체를 가장 가까운 대표객체에 배정하여 군집해를 얻는다.
군집해가 이전과 동일하면 Stop하고, 그렇지 않으면 [단계 2]를 반복한다.

```

new_cluster_solution <- assign_cluster(
  pairwise_distance_df,
  medoids_ids
)

is_converge <- near(
  1,
  # clusteval::cluster_similarity(
  #   as.factor(cluster_solution\$cluster),
  #   as.factor(new_cluster_solution\$cluster),
  #   similarity = "rand"
  # )
  flexclust::randIndex(
    as.factor(cluster_solution\$cluster),
    as.factor(new_cluster_solution\$cluster)
  )
)

print(is_converge)

##    ARI
## FALSE

```

위의 경우 첫 번째 iteration에서 군집해가 수정되었으므로 다음 iteration을 수행한다.

13.3.4.1 R 스크립트 구현

위 일련의 과정들을 수행하는 R 함수 스크립트를 구현해보자.

```
init_medoids <- function(distance_df, k = 1) {
  object_ids <- unique(distance_df$item1)
  k <- min(k, length(object_ids))

  v_j <- distance_df %>%
    group_by(item2) %>%
    mutate(prop = distance / sum(distance)) %>%
    ungroup() %>%
    group_by(item1) %>%
    summarize(sum_prop = sum(prop)) %>%
    rename(j = item1)

  medoids_ids <- v_j %>%
    arrange(sum_prop) %>%
    slice(1:k) %>%
    .\$j

  return(medoids_ids)
}

assign_cluster <- function(distance_df, medoids_ids) {
  distance_df %>%
    filter(
      !item1 %in% medoids_ids,
      item2 %in% medoids_ids
    ) %>%
    group_by(item1) %>%
    arrange(distance) %>%
    slice(1) %>%
    ungroup() %>%
    bind_rows(
      tibble(
        item1 = medoids_ids,
        item2 = medoids_ids,
        distance = 0
      )
    ) %>%
    rename(object = item1) %>%
    arrange(object) %>%
    group_by(item2) %>%
    mutate(cluster = str_c("{", str_c(object, collapse = ", "), "})) %>%
    ungroup() %>%
```

```

    select(-item2)
}

find_medoids <- function(distance_df, ids) {
  distance_df %>%
    filter(
      item1 %in% ids,
      item2 %in% ids
    ) %>%
    group_by(item1) %>%
    summarize(total_distance = sum(distance)) %>%
    arrange(total_distance) %>%
    slice(1) %>%
    .\$item1
}

kmeans_like_kmedoids <- function(distance_df, k = 1) {
  medoids_ids <- init_medoids(distance_df, k)

  while (TRUE) {
    cluster_solution <- assign_cluster(distance_df, medoids_ids)

    cluster_objects <- cluster_solution %>%
      split(.\$cluster) %>%
      map(~ .\$object)

    medoids_ids <- map_int(
      cluster_objects,
      find_medoids,
      distance_df = distance_df
    )

    new_cluster_solution <- assign_cluster(distance_df, medoids_ids)

    is_converge <- near(
      1,
      flexclust::randIndex(
        as.factor(cluster_solution\$cluster),
        as.factor(new_cluster_solution\$cluster)
      )
    )

    if (is_converge) break
  }
}

```

```

    return(medoids_ids)
}

```

위에서 정의한 함수 `kmeans_like_kmedoids`를 Table 13.2의 데이터에 적용한 군집결과 및 군집 대표객체는 아래와 같이 얻어진다.

```

pairwise_distance_df <- dist(df[, -1], upper = TRUE) %>%
  broom::tidy()

medoids_ids <- kmeans_like_kmedoids(pairwise_distance_df, k = 2)

medoids_ids

##      {1, 2} {3, 4, 5, 6}
##          1           5

```

13.4 페지 K-means 알고리즘

이 방법은 K-means 알고리즘과 유사하나, 하나의 객체가 여러 군집에 속할 가능성을 허용하는 확률 또는 이를 확장한 페지(fuzzy) 개념을 도입한 것이다. 객체 i 가 군집 j 에 속할 확률 P_{ij} 를 구하는 문제이다.

13.4.1 기본 R 스크립트

```

df <- tibble(
  id = c(1:10),
  x1 = c(6, 8, 14, 11, 15, 7, 13, 5, 3, 3),
  x2 = c(14, 13, 6, 8, 7, 15, 6, 4, 3, 2)
)

df %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c(
      '객체번호',
      '사용경력($x_1$)',
      '사용시간($x_2$)'
    ),
    caption = 'PC 사용자 데이터'
)

```

Table 13.3: PC 사용자 데이터

객체번호	사용경력 (\$x_1\$)	사용시간 (\$x_2\$)
1	6	14
2	8	13
3	14	6
4	11	8
5	15	7
6	7	15
7	13	6
8	5	4
9	3	3
10	3	2

```

cluster::fanny(df[, -1], k = 3, metric = "SqEuclidean")

## Fuzzy Clustering object of class 'fanny' :
## m.ship.expon.      2
## objective       18.37061
## tolerance        1e-15
## iterations       12
## converged        1
## maxit            500
## n                10
## Membership coefficients (in %, rounded):
##      [,1] [,2] [,3]
## [1,]   98    1    1
## [2,]   96    3    2
## [3,]    1   99    1
## [4,]   12   80    8
## [5,]    2   96    2
## [6,]   98    1    1
## [7,]    1   99    1
## [8,]    3    3   94
## [9,]    0    0   99
## [10,]   1    1   98
## Fuzzyness coefficients:
## dunn_coeff normalized
## 0.9225653 0.8838480
## Closest hard clustering:
## [1] 1 1 2 2 2 1 2 3 3 3
##
## Available components:

```

```
## [1] "membership"   "coeff"          "memb.exp"      "clustering"    "k.crisp"
## [6] "objective"     "convergence"    "diss"         "call"         "silinfo"
## [11] "data"
```

13.4.2 알고리즘

[단계 0] 초기 K 개의 군집을 임의로 결정한다.

$$P_{ij} = \begin{cases} 1 & \text{if object } i \text{ belongs to cluster } j \\ 0 & \text{otherwise} \end{cases}$$

```
init_cluster <- function(df, k = 1) {
  k <- min(k, nrow(df))

  while (TRUE) {
    cluster_ind <- sample.int(k, size = nrow(df), replace = TRUE)
    if (length(unique(cluster_ind)) == k) break
  }

  map_dfc(unique(cluster_ind), ~ as.double(cluster_ind == .))
}

set.seed(4000)

cluster_membership <- init_cluster(df[, -1], k = 3)

cluster_membership

## # A tibble: 10 x 3
##       V1     V2     V3
##   <dbl> <dbl> <dbl>
## 1     1     0     0
## 2     2     0     0
## 3     3     1     0
## 4     4     0     0
## 5     5     0     1
## 6     6     0     1
## 7     7     0     1
## 8     8     1     0
## 9     9     0     0
## 10    10    0     0
```

[단계 1] 각 군집의 중심좌표를 산출한다.

$$\mathbf{c}_j = \frac{\sum_{i=1}^n P_{ij}^m \mathbf{x}_i}{\sum_{i=1}^n P_{ij}^m}$$

여기에서 상수 m 은 1보다 큰 값을 사용한다.

```
find_center <- function(df, p, m) {
  wt <- p ^ m
  df %>%
    summarize_all(weighted.mean, w = wt)
}

cluster_df <- map_dfr(cluster_membership, find_center, df = df[, -1], m = 2)

cluster_df

## # A tibble: 3 x 2
##       x1     x2
##   <dbl> <dbl>
## 1  6.2    8
## 2  8.67   8.33
## 3  14      6.5
```

[단계 2] 군집 membership 계수 P_{ij} 를 업데이트한다.

$$P_{ij} = \frac{d(\mathbf{x}_i, \mathbf{c}_j)^{-\frac{1}{m-1}}}{\sum_{a=1}^K d(\mathbf{x}_i, \mathbf{c}_a)^{-\frac{1}{m-1}}}$$

여기에서 거리함수 $d()$ 는 제곱 유클리드 거리를 사용한다.

```
update_membership <- function(df, cluster_df, m) {
  distance_mat <- flexclust::dist2(df, cluster_df) ^ 2

  p <- distance_mat ^ (-1 / (m - 1)) %>%
    `/`(rowSums(.)) %>%
    as_tibble(.name_repair = "minimal")

  p
}

update_membership(df[, -1], cluster_df, m = 2)
```

A tibble: 10 x 3

```

##      ``    ``    ``
##      <dbl>  <dbl> <dbl>
## 1 0.451   0.414  0.135
## 2 0.380   0.483  0.137
## 3 0.00381 0.00730 0.989
## 4 0.139   0.576  0.285
## 5 0.0152  0.0285 0.956
## 6 0.406   0.427  0.166
## 7 0.0231  0.0479 0.929
## 8 0.574   0.311  0.115
## 9 0.542   0.315  0.143
## 10 0.508   0.325  0.166

```

13.4.3 R 스크립트 구현

```

fuzzy_kmeans <- function(df, k = 1, m = 2, max_iter = 1000L, tol = 1e-9) {
  k <- min(k, nrow(df))
  i <- 0L

  cluster_membership <- init_cluster(df, k)

  while (i < max_iter) {
    i <- i + 1L

    cluster_df <- map_dfr(cluster_membership, find_center, df = df, m = m)

    new_cluster_membership <- update_membership(df, cluster_df, m)

    if (max(abs(cluster_membership - new_cluster_membership)) < tol) break

    cluster_membership <- new_cluster_membership
  }

  res <- list(
    n_iteration = i,
    center = cluster_df,
    membership = cluster_membership %>% as.matrix()
  )

  return (res)
}

```

```

set.seed(4000)

fuzzy_kmeans_solution <- fuzzy_kmeans(df[, -1], k = 3, m = 2)

fuzzy_kmeans_solution$n_iteration

## [1] 16

fuzzy_kmeans_solution$center

## # A tibble: 3 x 2
##       x1     x2
##   <dbl> <dbl>
## 1 3.64  2.98
## 2 7.00 14.0 
## 3 13.4   6.63

fuzzy_kmeans_solution$membership

##
##  [1,]  0.007809655 0.983147737 0.009042608
##  [2,]  0.015726915 0.957515486 0.026757600
##  [3,]  0.006068358 0.006268614 0.987663029
##  [4,]  0.078772454 0.120672309 0.800555237
##  [5,]  0.017165082 0.022105992 0.960728926
##  [6,]  0.006532987 0.984345339 0.009121674
##  [7,]  0.005945712 0.005766360 0.988287928
##  [8,]  0.939278603 0.026072757 0.034648640
##  [9,]  0.993661877 0.002989486 0.003348637
## [10,]  0.981125198 0.008481303 0.010393499

fuzzy_kmeans_solution$membership %>%
  apply(1, which.max)

## [1] 2 2 3 3 3 2 3 1 1 1

```

13.5 모형기반 군집방법

모형기반 군집방법(model-based clustering)에서는 각 객체가 혼합분포(mixture)를 따른다고 가정하여 객체의 군집배정변수를 통계적으로 추정하는 것이다.

Table 13.4: 모형기반 군집 학습 데이터

객체번호	\$x_1\$	\$x_2\$
1	4	12
2	6	13
3	6	15
4	10	4
5	11	3
6	12	2
7	12	5

13.5.1 기본 R script

```
df <- tibble(
  id = c(1:7),
  x1 = c(4, 6, 6, 10, 11, 12, 12),
  x2 = c(12, 13, 15, 4, 3, 2, 5)
)

df %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c(
      '객체번호',
      '$x_1$',
      '$x_2$'
    ),
    caption = '모형기반 군집 학습 데이터'
)
```

Table 13.4의 7개의 객체를 두 개의 군집에 배정하는 간단한 R 스크립트는 아래와 같다. 여기에서 `mclust::meVII`는 각 군집의 분산-공분산 행렬은 서로 다르되, 각 변수의 분산이 동일하며 공분산은 0이라 가정한다.

```
set.seed(1024)

# 군집 개수
K <- 2

# z_ik 값 초기화
init_z <- matrix(runif(nrow(df) * K), ncol = K) %>%
  `/`(apply(., 1, sum))
```

```

# EM 알고리즘 - 분산-공분산 행렬: unequal volume (V), spherical (II)
sol <- mclust::meVII(data = df[, -1], z = init_z)

# 군집 사후확률
sol$z

##          [,1]      [,2]
## [1,] 7.596402e-28 1.000000e+00
## [2,] 8.254183e-27 1.000000e+00
## [3,] 9.463816e-36 1.000000e+00
## [4,] 1.000000e+00 6.832084e-20
## [5,] 1.000000e+00 1.473491e-25
## [6,] 1.000000e+00 4.876251e-31
## [7,] 1.000000e+00 1.480388e-20

# 혼합분포
sol$parameters

## $pro
## [1] 0.5714286 0.4285714
##
## $mean
##      [,1]      [,2]
## x1 11.25  5.333333
## x2  3.50 13.333333
##
## $variance
## $variance$modelName
## [1] "VII"
##
## $variance$d
## [1] 2
##
## $variance$G
## [1] 2
##
## $variance$sigma
## , , 1
##
##      x1      x2
## x1 0.96875 0.00000
## x2 0.00000 0.96875
##
## , , 2

```

```

##          x1      x2
## x1 1.222222 0.000000
## x2 0.000000 1.222222
##
## $variance$sigmasq
## [1] 0.968750 1.222222
##
## $variance$scale
## [1] 0.968750 1.222222
##
## $Vinv
## NULL

```

13.5.2 EM 알고리즘

우선, 필요한 기호를 다음과 같이 정의하자.

- $f_k(\mathbf{x} | \theta_k)$: 군집 k 에 속하는 객체 \mathbf{x} 의 확률밀도함수 (θ_k 는 관련 파라미터)
- τ_k : 임의의 객체가 군집 k 에 속할 사전확률 ($\tau_k \geq 0, \sum_{k=1}^K \tau = 1$)

이 때, 임의의 객체 \mathbf{x} 는 다음과 같은 혼합 확률밀도함수를 갖는다.

$$f(\mathbf{x} | \boldsymbol{\theta}) = \sum_{k=1}^K \tau_k f_k(\mathbf{x} | \theta_k)$$

본 장에서는 f_k 가 다변량 정규분포를 나타낸다고 가정하자.

추가로, 각 객체가 속하는 군집에 대한 지시변수 z_{ik} 를 아래와 같이 정의하자.

$$z_{ik} = \begin{cases} 1 & \text{if object } i \text{ belongs to cluster } k \\ 0 & \text{otherwise} \end{cases}$$

이 z_{ik} 변수는 실제값이 관측되지 않는 변수이므로, 최우추정법을 이용하여 그 기대값, 즉 객체 i 가 군집 k 에 속할 확률을 추정한다. 보다 자세한 설명은 교재 (전치혁, 2012) 참조.

앞의 13.5.1장에서 수행했던 예제를 단계별로 살펴보기로 하자.

[단계 0] \hat{z}_{ik} 를 초기화한다.

```

set.seed(1024)

# 군집 개수
K <- 2

# z_ik 추정값 초기화
z_hat <- matrix(runif(nrow(df) * K), ncol = K) %>%
  `/~(apply(., 1, sum)) %>%
  as_tibble() %>%
  `names<-` (str_c("C", 1:K))

z_hat

## # A tibble: 7 x 2
##       C1     C2
##   <dbl> <dbl>
## 1 0.406  0.594
## 2 0.623  0.377
## 3 0.372  0.628
## 4 0.956  0.0444
## 5 0.0214 0.979
## 6 0.681  0.319
## 7 0.264  0.736

```

[단계 1] (M-step) \hat{z}_{ik} 를 바탕으로 파라미터를 추정한다.

우선 τ_k 와 μ_k 를 다음과 같이 추정한다.

$$\hat{\tau}_k = \frac{\sum_{i=1}^n \hat{z}_{ik}}{n}$$

```

tau_hat <- map(z_hat, mean)
tau_hat

```

```

## $C1
## [1] 0.4746762
##
## $C2
## [1] 0.5253238

```

$$\hat{\mu}_k = \frac{\sum_{i=1}^n \hat{z}_{ik} \mathbf{x}_i}{\sum_{i=1}^n \hat{z}_{ik}}$$

```
mu_hat <- map(z_hat, ~colSums(. * df[, -1]) / sum(.))  
mu_hat
```

```
## $C1  
##      x1      x2  
## 8.644042 7.559792  
##  
## $C2  
##      x1      x2  
## 8.777757 7.853884
```

분산-공분산 행렬 Σ_k 의 추정은 분산-공분산 구조를 어떻게 가정하느냐에 따라 다르다. 우선, 일반적으로 분산-공분산 행렬 Σ_k 는 아래와 같이 decompose할 수 있다 (Banfield and Raftery, 1993).

$$\Sigma_k = \lambda_k \mathbf{D}_k \mathbf{A}_k \mathbf{D}_k^\top$$

여기에서

- $\lambda_k = \det \Sigma_k^{1/2}$; 군집 k 이 크기와 관련
- \mathbf{D}_k : matrix of eigenvectors of Σ_k ; 군집 k 의 방향(orientation)과 관련
- \mathbf{A}_k : diagonal matrix s.t. $\det \mathbf{A}_k = 1$; 군집 k 의 형태와 관련

이다. λ_k , \mathbf{D}_k 및 \mathbf{A}_k 에 적용되는 제약조건에 따라 분산-공분산 모형을 정의할 수 있다. 아래는 본 장에서 다룰 세 가지 모형이다.

```
tribble(  
  ~model, ~sigma,  
  "VII", "$\\lambda_k \\mathbf{I}$",  
  "VEI", "$\\lambda_k \\mathbf{A}$",  
  "VEE", "$\\lambda_k \\mathbf{D} \\mathbf{A} \\mathbf{D}^\\top$"  
) %>%  
  knitr::kable(  
    booktabs = TRUE,  
    align = c('c', 'c'),  
    col.names = c(  
      '분산-공분산 모형',  
      '$\\boldsymbol\\Sigma_k$'  
    ),  
    caption = 'Within-group 분산-공분산 모형'  
)
```

즉,

Table 13.5: Within-group 분산-공분산 모형

분산-공분산 모형	$\$\\boldsymbol\\Sigma_k\$$
VII	$\$\\lambda_k \\mathbf{I} \$$
VEI	$\$\\lambda_k \\mathbf{A} \$$
VEE	$\$\\lambda_k \\mathbf{D} \\mathbf{A} \\mathbf{D}^\\top \$$

- “VII”
 - for all k , $\mathbf{D}_k = \mathbf{I}$
 - for all k , $\mathbf{A}_k = \mathbf{I}$
- “VEI”
 - for all k , $\mathbf{D}_k = \mathbf{I}$
 - for all k , $\mathbf{A}_k = \mathbf{A}$
- “VEE”
 - for all k , $\mathbf{D}_k = \mathbf{D}$
 - for all k , $\mathbf{A}_k = \mathbf{A}$

Celeux and Govaert (1995)에 각 분산-공분산 모형에 대한 추정값을 얻는 반복적 알고리즘이 소개되어 있으며, 이는 교재 (전치혁, 2012)에도 설명되어 있다.

우선, 각 군집 k 내에서의 scatter matrix를 아래와 같이 계산한다.

$$\mathbf{W}_k = \sum_{i=1}^n \hat{z}_{ik} (\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^\top$$

```
p <- ncol(df[, -1])
W <- map(mu_hat,
  ~pmap_dfc(list(x = df[, -1], mu = .),
    function(x, mu) x - mu)) %>%
  map(as.matrix, nrow = p, ncol = p) %>%
  map2(z_hat, ~ t(.x) %*% (.y * .x))
```

W

```
## $C1
##           x1          x2
## x1  28.22794 -44.46516
## x2 -44.46516  82.36848
##
## $C2
##           x1          x2
## x1  37.16942 -53.17491
## x2 -53.17491  92.90912
```

이후 분산-공분산 모형에 따라 아래와 같이 각 군집의 분산-공분산 행렬을 추정한다.

[VII의 경우] 반복 업데이트의 과정 없이 closed-form으로 해가 존재한다.

$$\lambda_k = \frac{Tr(\mathbf{W}_k)}{p \sum_{i=1}^n \hat{z}_{ik}}$$

$$\Sigma_k = \lambda_k \mathbf{I}$$

이 때 p 는 관측값의 차원수이다 ($\mathbf{x} \in \mathbb{R}^p$).

[VEI의 경우]

VEI-0. 행렬 $\mathbf{B} = \mathbf{I}$ 로 초기화한다.

VEI-1. λ_k 값을 아래와 같이 계산한다.

$$\lambda_k = \frac{Tr(\mathbf{W}_k \mathbf{B}^{-1})}{p \sum_{i=1}^n \hat{z}_{ik}}$$

VEI-2. 행렬 \mathbf{B} 를 아래와 같이 업데이트한다.

$$\mathbf{B} = \frac{diag\left(\sum_{k=1}^K \frac{\mathbf{W}_k}{\lambda_k}\right)}{\left(\det diag\left(\sum_{k=1}^K \frac{\mathbf{W}_k}{\lambda_k}\right)\right)^{1/p}}$$

VEI-3. 결과가 수렴하면 종료, 그렇지 않으면 VEI-1로 돌아간다. 최종 수렴한 결과를 통해 각 군집의 분산-공분산 행렬을 아래와 같이 얻는다.

$$\Sigma_k = \lambda_k \mathbf{B}$$

[VEE의 경우]

VEE-0. 행렬 $\mathbf{C} = \mathbf{I}$ 로 초기화한다.

VEE-1. λ_k 값을 아래와 같이 계산한다.

$$\lambda_k = \frac{Tr(\mathbf{W}_k \mathbf{C}^{-1})}{p \sum_{i=1}^n \hat{z}_{ik}}$$

VEE-2. 행렬 \mathbf{C} 를 아래와 같이 업데이트한다.

$$\mathbf{C} = \frac{\sum_{k=1}^K \frac{\mathbf{W}_k}{\lambda_k}}{\left(\det \sum_{k=1}^K \frac{\mathbf{W}_k}{\lambda_k}\right)^{1/p}}$$

VEE-3. 결과가 수렴하면 종료, 그렇지 않으면 VEE-1로 돌아간다. 최종 수렴한 결과를 통해 각 군집의 분산-공분산 행렬을 아래와 같이 얻는다.

$$\boldsymbol{\Sigma}_k = \lambda_k \mathbf{C}$$

위 각 분산-공분산 모형을 추정 과정에 대한 설명은, 보다 일반화된 분산-공분산 구조 $\lambda_k \mathbf{D}_k \mathbf{A}_k \mathbf{D}_k^\top$ 을 추정하는 과정을 각 모형에 추가되는 제약에 따라 조금 더 단순하게 표현한 것이다. 보다 자세한 내용은 Celeux and Govaert (1995) 참조.

아래와 같이 군집 내 분산-공분산 행렬을 추정하는 함수 `estimate_mixture_cov`를 구현해보자. 해당 함수는 3개의 입력변수를 사용한다.

- `modelName`: 분산-공분산 모형 이름. “VII”, “VEI”, “VEE” 중 하나를 선택한다.
- `W`: K 개의 scatter matrix ($\mathbf{W}_1, \dots, \mathbf{W}_K$)를 원소로 지니는 리스트 (list)
- `z`: z_{ik} 값을 지닌 데이터 프레임 (data.frame). n 개의 행과 K 개의 열로 이루어진다. 즉, 행은 각 관측객체를 나타내며, 열은 각 군집을 나타낸다.

```
estimate_mixture_cov <- function(modelName, W, z) {
  K <- ncol(z)
  p <- ncol(W[[1]])

  # 초기화
  D <- map(1:K, ~ diag(1, p))
  A <- map(1:K, ~ diag(1, p))

  get_lambda <- function(W, z, D, A, p) {
    pmap(
      list(W, z, D, A),
      function(W, z, D, A, p)
        sum(diag(W %*% D %*% solve(A) %*% t(D))) / (sum(z) * p),
      p = p
    )
  }

  objective_value <- function(W, z, lambda, D, A, p) {
    pmap_dbl(
      list(W, z, lambda, D, A),
      function(W, z, lambda, D, A, p)
        sum(diag(W %*% D %*% solve(A) %*% t(D))) / lambda +
        p * sum(z) * log(lambda),
      p = p
    ) %>%
      sum()
  }
}
```

```

i <- 0L
obj <- Inf

while(TRUE) {
  i <- i + 1L

  lambda <- get_lambda(W, z, D, A, p)
  new_obj <- objective_value(W, z, lambda, D, A, p)

  if (obj - new_obj < 1e-9) break

  if (modelName == "VII") {

    } else if (modelName == "VEI") {
      B <- map2(W, lambda, ~ diag(.x) / .y) %>%
        reduce(`+`) %>%
        diag() %>%
        `/`(det(.) ^ (1 / p))
      A <- map(1:K, ~ B)
    } else if (modelName == "VEE") {
      C <- map2(W, lambda, ~ .x / .y) %>%
        reduce(`+`) %>%
        `/`(det(.) ^ (1 / p))

      s <- svd(C)
      A <- map(1:K, ~ diag(s$d))
      D <- map(1:K, ~ s$u)
    } else {
      stop("Model ", modelName, " is not supported yet.")
    }

    obj <- new_obj
  }

  Sigma <- pmap(
    list(lambda, D, A),
    function(lambda, D, A) lambda * (D %*% A %*% t(D))
  )

  return (list(
    volume = lambda,
    shape = A,
    orientation = D,
    Sigma = Sigma
  ))
}

```

```
}
```

초기값으로 주어진 \hat{z}_{ik} 를 이용하여 “VII” 구조의 분산-공분산 행렬을 구해보자.

```
estimate_mixture_cov("VII", W, z_hat)
```

```
## $volume
## $volume$C1
## [1] 16.64239
##
## $volume$C2
## [1] 17.68685
##
##
## $shape
## $shape[[1]]
## [,1] [,2]
## [1,] 1 0
## [2,] 0 1
##
## $shape[[2]]
## [,1] [,2]
## [1,] 1 0
## [2,] 0 1
##
##
## $orientation
## $orientation[[1]]
## [,1] [,2]
## [1,] 1 0
## [2,] 0 1
##
## $orientation[[2]]
## [,1] [,2]
## [1,] 1 0
## [2,] 0 1
##
##
## $Sigma
## $Sigma$C1
## [,1] [,2]
## [1,] 16.64239 0.00000
## [2,] 0.00000 16.64239
##
## $Sigma$C2
```

```
##          [,1]      [,2]
## [1,] 17.68685  0.00000
## [2,]  0.00000 17.68685
```

위 `estimate_mixture_cov` 함수에서 “VII”보다 일반화된 “VEI”와 “VEE” 분산-공분산 모형에 대한 추정도 구현되어 있다.

```
# estimate_mixture_cov("VEI", W, z_hat)
# estimate_mixture_cov("VEE", W, z_hat)
```

위 일련의 파라미터 추정을 하나의 함수 `GMM_Mstep`으로 아래와 같이 구성해보자.

```
GMM_Mstep <- function(df, z, modelName = "VII") {
  tau <- map(z, mean)
  mu <- map(z, ~colSums(. * df) / sum(.))

  p <- ncol(df)
  W <- map(mu, ~pmap_dfc(
    list(x = df, mu = .), function(x, mu) x - mu)) %>%
    map(as.matrix, nrow = p, ncol = p) %>%
    map2(z, ~ t(.x) %*% (.y * .x))

  var_cov <- estimate_mixture_cov(modelName, W, z)

  res <- list(
    tau = tau,
    mu = mu,
    Sigma = var_cov$Sigma
  )

  return (res)
}

Mstep_res <- GMM_Mstep(df[, -1], z_hat)

Mstep_res

## $tau
## $tau$C1
## [1] 0.4746762
##
## $tau$C2
## [1] 0.5253238
##
```

```

## 
## $mu
## $mu$C1
##      x1      x2
## 8.644042 7.559792
##
## $mu$C2
##      x1      x2
## 8.777757 7.853884
##
## 
## $Sigma
## $Sigma$C1
##      [,1]      [,2]
## [1,] 16.64239  0.00000
## [2,]  0.00000 16.64239
##
## $Sigma$C2
##      [,1]      [,2]
## [1,] 17.68685  0.00000
## [2,]  0.00000 17.68685

```

[단계 2] (E-step) M-step에서의 파라미터 추정치를 바탕으로 \hat{z}_{ik} 를 산출한다.

$$\hat{z}_{ik} = \frac{\hat{\tau}_k f_k(\mathbf{x}_i | \hat{\boldsymbol{\mu}}_k, \hat{\boldsymbol{\Sigma}}_k)}{\sum_{l=1}^K \hat{\tau}_l f_l(\mathbf{x}_i | \hat{\boldsymbol{\mu}}_l, \hat{\boldsymbol{\Sigma}}_l)}$$

```

GMM_Estep <- function(df, tau, mu, Sigma) {
  pmap_dfc(list(tau = tau, mu = mu, Sigma = Sigma),
            function(df, tau, mu, Sigma)
              tau * mvtnorm::dmvnorm(df, mean = mu, sigma = Sigma),
              df = df) %>%
  `/`(rowSums(.))
}

new_z_hat <- GMM_Estep(df[, -1], Mstep_res$tau, Mstep_res$mu, Mstep_res$Sigma)

new_z_hat

##          C1      C2
## 1 0.4626878 0.5373122
## 2 0.4568716 0.5431284
## 3 0.4373551 0.5626449
## 4 0.4964082 0.5035918

```

```
## 5 0.4934276 0.5065724
## 6 0.4886741 0.5113259
## 7 0.4870085 0.5129915
```

[단계 3] 수렴조건을 만족하면 stop, 그렇지 않으면 [단계 1]을 반복한다. 일반적으로는 우도함수값의 변화량을 수렴조건으로 사용하나, 본 장에서는 간단하게 z_{ik} 값의 변화량을 기준으로 수렴을 판단하도록 하자.

```
max(abs(z_hat - new_z_hat)) < 1e-9
```

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

위 수식의 값이 TRUE이면 수렴, FALSE이면 [단계 1]을 반복한다.

13.5.3 R 스크립트 구현

위 일련의 과정들을 포함하는 하나의 함수 GMM_EM을 아래와 같이 구현해보자. 앞에서 정의했던 함수 GMM_Mstep 및 GMM_Estep을 재사용한다.

```
GMM_EM <- function(df, K, modelName = "VII", tol = 1e-9) {
  K <- min(K, nrow(df))

  i <- 0L

  # [단계 0] z_ik 추정값 초기화
  z_hat <- matrix(runif(nrow(df) * K), ncol = K) %>%
    `/^(apply(., 1, sum)) %>%
    as_tibble() %>%
    `names<-`(str_c("C", 1:K))

  while (TRUE) {
    i <- i + 1L

    # [단계 1] M-step
    Mstep_res <- GMM_Mstep(df, z_hat, modelName)

    # [단계 2] E-step
    new_z_hat <- GMM_Estep(df, Mstep_res$tau, Mstep_res$mu, Mstep_res$Sigma)

    # [단계 3] 수렴조건 확인
    if (max(abs(z_hat - new_z_hat)) < tol) break

    z_hat <- new_z_hat
  }
}
```

```
    }

    return (list(z = z_hat,
                  parameters = Mstep_res,
                  n_iteration = i))
}
```

위 함수를 학습데이터 Table 13.4에 적용한 결과는 아래와 같다.

```
VII_res <- GMM_EM(df[, -1], 2, "VII")  
  
# 군집 멤버쉽  
VII_res$z
```

```

##          C1          C2
## 1 1.000000e+00 6.786307e-28
## 2 1.000000e+00 8.771316e-27
## 3 1.000000e+00 8.895007e-36
## 4 5.251505e-17 1.000000e+00
## 5 7.046079e-22 1.000000e+00
## 6 1.843399e-26 1.000000e+00
## 7 1.544788e-17 1.000000e+00

```

```
# 혼합분포  
VII_res$parameters
```

```

## $tau
## $tau$C1
## [1] 0.4285714
##
## $tau$C2
## [1] 0.5714286
##
## $mu
## $mu$C1
##           x1           x2
## 5.3333333 13.3333333
##
## $mu$C2
##           x1           x2
## 11.25    3.50
##
##
```

```

## $Sigma
## $Sigma$C1
##      [,1]      [,2]
## [1,] 1.222222 0.000000
## [2,] 0.000000 1.222222
##
## $Sigma$C2
##      [,1]      [,2]
## [1,] 0.96875 0.00000
## [2,] 0.00000 0.96875

```

13.5.4 R 패키지 내 모형기반 군집분석

R 패키지 `mclust`를 통해, 위에서 살펴본 VII, VEI, VEE 외에 보다 다양한 분산-공분산 모형을 가정한 군집분석을 수행할 수 있다 (Scrucca et al., 2016). 다음은 “EEI” 구조에 대한 수행 예이다.

```
res_mclust_EEI <- mclust::meEEI(df[, -1], z_hat)
```

위 수행 결과 객체는 리스트 형태이며, 그 원소 중 `z`는 \hat{z}_{ik} 값을, `parameters`는 혼합분포 파라미터 추정값 ($\hat{\pi}_k$, $\hat{\mu}_k$, $\hat{\Sigma}_k$)을 나타낸다.

```
res_mclust_EEI$z
```

```

##      [,1]      [,2]
## [1,] 6.198742e-26 1.000000e+00
## [2,] 2.193775e-22 1.000000e+00
## [3,] 1.432979e-28 1.000000e+00
## [4,] 1.000000e+00 3.497777e-20
## [5,] 1.000000e+00 1.350932e-26
## [6,] 1.000000e+00 5.217649e-33
## [7,] 1.000000e+00 9.883335e-24

```

```
res_mclust_EEI$parameters
```

```

## $pro
## [1] 0.5714286 0.4285714
##
## $mean
##      [,1]      [,2]
## x1 11.25 5.333333
## x2 3.50 13.333333
##
```

```

## $variance
## $variance$modelName
## [1] "EEI"
##
## $variance$d
## [1] 2
##
## $variance$G
## [1] 2
##
## $variance$sigma
## , , 1
##
##           x1      x2
## x1 0.7738095 0.000000
## x2 0.0000000 1.380952
##
## , , 2
##
##           x1      x2
## x1 0.7738095 0.000000
## x2 0.0000000 1.380952
##
##
## $variance$Sigma
##           x1      x2
## x1 0.7738095 0.000000
## x2 0.0000000 1.380952
##
## $variance$scale
## [1] 1.033728
##
## $variance$shape
## [1] 0.7485618 1.3358950
##
##
## $Vinv
## NULL

```

`mclust` 패키지 내의 함수 `densityMclust`는 Bayesian information criterion (BIC)을 기준으로 최적의 혼합분포를 찾는 함수이다. 즉, 내부적으로 여러가지 분산-공분산 모형과 군집 수의 조합에 대한 군집분석을 수행한 뒤, 각 조합의 최종결과에서 얻어진 BIC 값을 기준으로 최적의 조합을 선정한다.

```
res_mclust_opt <- mclust::densityMclust(df[, -1], verbose = FALSE)
```

함수 수행결과 객체의 BIC 원소가 나타내는 결과는 아래와 같다.

```
res_mclust_opt$BIC
```

```
## Bayesian Information Criterion (BIC):
##      EII      VII     EEI     VEI     EVI     VVI     EEE
## 1 -85.40006 -85.40006 -85.70851 -85.70851 -85.70851 -85.70851 -75.27433
## 2 -62.00992 -63.86240 -63.37677 -65.22485 -65.32204 -67.17013 -64.66516
## 3 -65.47796      NA -66.01911      NA      NA      NA -67.87781
## 4 -69.04346      NA -70.17240      NA      NA      NA -67.31282
## 5 -72.02226      NA -73.96817      NA      NA      NA      NA
## 6 -62.27998      NA -64.22589      NA      NA      NA      NA
## 7      NA      NA      NA      NA      NA      NA      NA
##      EVE      VEE      VVE     EEV     VEV     EVV     VVV
## 1 -75.27433 -75.27433 -75.27433 -75.27433 -75.27433 -75.27433 -75.27433
## 2 -65.06811 -66.60332 -66.92481 -65.42506 -67.34154 -66.55375 -68.44666
## 3      NA      NA      NA -71.48895      NA      NA      NA
## 4      NA      NA      NA      NA      NA      NA      NA
## 5      NA      NA      NA      NA      NA      NA      NA
## 6      NA      NA      NA      NA      NA      NA      NA
## 7      NA      NA      NA      NA      NA      NA      NA
##
## Top 3 models based on the BIC criterion:
##      EII,2     EII,6     EEI,2
## -62.00992 -62.27998 -63.37677
```

수행 결과 가장 큰 BIC 값을 지닌 최적의 조합은 EII 분산-공분산 모형으로 2개의 군집을 가정했을 때 얻어진다.

densityMclust 함수 수행 결과 얻어진 혼합분포를 plotting해보자.

```
plot(res_mclust_opt, what = "density",
      data = df[, -1], points.cex = 0.5)
```

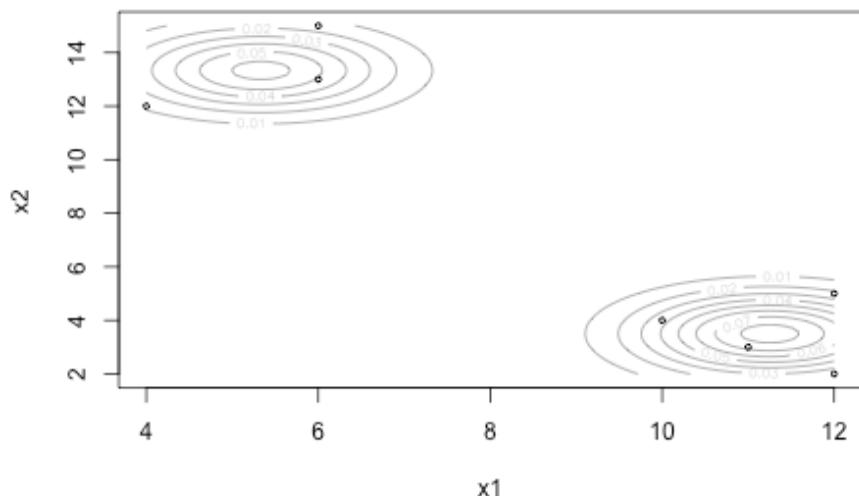


Figure 13.2: mclust::densityMclust 수행 결과 얻어진 혼합분포

Chapter 14

군집해의 평가 및 해석

군집분석 수행 시에는 분류분석과 달리 각 객체가 속하는 군집에 대하여 알려진 학습표본이 없기 때문에 어떤 군집해의 성능을 평가하기가 곤란하다. 각 객체가 이차원 또는 삼차원의 변수로 이루어진 경우에는 각 객체를 좌표축에 나타내어 도식화함으로써 어느 정도 군집해의 타당성을 정성적으로 평가가 가능할 것이다. 이보다 높은 차원의 경우에는 도식화가 불가능하다. 따라서 응용분야에 따라 전문가의 견해가 군집해의 평가에 필요할 수도 있다.

14.1 필요 R package 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
cluster	2.0.8
flexclust	1.4-0
clValid	0.6-6

14.2 군집해의 평가

군집해의 정량적인 평가척도에 대한 연구는 지속적으로 이루어지고 있으며, 크게 외부평 가지수(external index)와 내부평 가지수(internal index)로 구분되고 있다.

14.2.1 외부평 가지수

이미 잘 알려진 군집해가 있다고 가정할 때, 새로 제안된 군집해를 기준에 알려진 군집해와 비교하는 평가척도이다.

n 개의 객체에 대하여 알려진 기준 군집해를 다음과 같다고 하자.

$$\mathbf{U} = \{U_1, U_2, \dots, U_r\}$$

즉, 기준 군집해 \mathbf{U} 는 r 개의 군집으로 구성되며, k 번째 군집을 U_k 로 나타낸다.

유사하게, 비교 대상의 군집해를 다음과 같이 s 개의 군집으로 구성된 \mathbf{V} 로 나타내자.

$$\mathbf{V} = \{V_1, V_2, \dots, V_s\}$$

14.2.1.1 기본 R 스크립트

다음의 두 군집해 간의 유사도를 랜드지수 및 수정랜드지수를 이용하여 표현할 수 있다.

$$\begin{aligned} U &= \{\{1, 2, 3, 4\}, \{5, 6, 7\}, \{8, 9, 10\}\} \\ V &= \{\{1, 2, 5, 8\}, \{3, 6, 9\}, \{4, 7, 10\}\} \end{aligned}$$

랜드지수 및 수정랜드지수는 `flexclust` 패키지의 `randIndex` 함수를 호출하여 계산할 수 있으며, 랜드지수를 계산할 때는 `correct = FALSE`, 수정랜드지수를 계산할 때는 `correct = TRUE`로 파라미터 값을 지정하여야 한다. 파라미터 값을 지정하지 않을 때는 기본값으로 수정랜드지수를 계산한다.

```
sol_1 <- c(1, 1, 1, 1, 2, 2, 2, 3, 3, 3)
sol_2 <- c(1, 1, 2, 3, 1, 2, 3, 1, 2, 3)
map(c(FALSE, TRUE),
  ~ flexclust::randIndex(x = sol_1, y = sol_2, correct = .x))
```

```
## [[1]]
##          RI
## 0.5111111
##
## [[2]]
##      ARI
## -0.25
```

14.2.1.2 랜드지수

두 군집해 U, V 내에서 각 객체가 속한 군집을 나타내기 위한 `membership` 변수를 아래와 같이 정의하자.

$$u_{ik} = \begin{cases} 1 & \text{if object } i \text{ belongs to cluster } U_k, \text{ i.e. } i \in U_k, i = 1, \dots, n, k = 1, \dots, r \\ 0 & \text{otherwise} \end{cases}$$

$$v_{ik} = \begin{cases} 1 & \text{if object } i \text{ belongs to cluster } V_k, \text{ i.e. } i \in V_k \\ 0 & \text{otherwise} \end{cases}, i = 1, \dots, n, k = 1, \dots, s$$

랜드지수를 산출하기 위해, 우선 다음과 같은 네 가지 값을 정의한다.

$$\begin{aligned} a &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^r \sum_{l=1}^s u_{ik} u_{jk} v_{il} v_{jl} \\ b &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^r \sum_{l=1}^s u_{ik} u_{jk} v_{il} (1 - v_{jl}) \\ c &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^r \sum_{l=1}^s u_{ik} (1 - u_{jk}) v_{il} v_{jl} \\ d &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \sum_{k=1}^r \sum_{l=1}^s u_{ik} (1 - u_{jk}) v_{il} (1 - v_{jl}) \end{aligned}$$

이 때, Rand (1971) 가 제안한 랜드지수 (Rand index) 는 다음과 같이 정의된다.

$$RI = \frac{a + d}{a + b + c + d} \quad (14.1)$$

여기서, $a + b + c + d$ 는 객체 쌍의 전체 수를 의미하므로 다음과 같다.

$$a + b + c + d = \binom{n}{2}$$

식 (14.1)은 0에서 1 사이의 값을 갖게 되며, 0에 가까울수록 두 군집해가 일치하지 않음을, 1에 가까울수록 두 군집해가 일치함을 나타낸다. 또한 랜드지수는 랜덤하게 작성한 두 군집해 간의 유사도의 기대값이 0보다 크다.

Hubert and Arabie (1985) 는 수정랜드지수 (adjusted Rand index) 를 아래와 같이 제안하였다.

$$RI_{adj} = \frac{2(ad - bc)}{(a + b)(b + d) + (a + c)(c + d)} \quad (14.2)$$

식 (14.2)은 랜덤한 군집해 간의 비교의 경우 0에 가까운 값을 갖는다.

아래 구현한 `rand_index` 함수는 임의의 두 군집해 `u` 와 `v`에 대한 랜드지수 및 수정랜드지수를 계산하는 함수이다.

- u, v : 서로 비교할 두 개의 군집해 벡터. 각 원소값은 각 객체가 속한 군집을 나타낸다.
- 다음과 같은 두 개의 component를 지닌 list를 리턴한다.
 - ri : 랜드지수
 - adj_ri : 수정랜드지수

```

rand_index <- function(u, v) {
  if (!is_vector(u) || !is_vector(v)) {
    stop("Input needs to be vector")
  } else if (length(u) != length(v)) {
    stop("Vectors u and v must have the same length.")
  }

  U <- tibble(
    i = seq_along(u),
    cluster = u
  ) %>%
    inner_join(
      rename(., j = i),
      by = "cluster"
    ) %>%
    select(-cluster) %>%
    filter(i < j)

  V <- tibble(
    i = seq_along(v),
    cluster = v
  ) %>%
    inner_join(
      rename(., j = i),
      by = "cluster"
    ) %>%
    select(-cluster) %>%
    filter(i < j)

  a <- nrow(inner_join(U, V, by = c("i", "j")))
  b <- nrow(anti_join(U, V, by = c("i", "j")))
  c <- nrow(anti_join(V, U, by = c("i", "j")))
  d <- choose(length(u), 2) - (a + b + c)

  ri <- (a + d) / (a + b + c + d)
  adj_ri <- 2 * (a * d - b * c) /
    ((a + b) * (b + d) + (a + c) * (c + d))

  return(list(ri = ri, adj_ri = adj_ri))
}

```

```
}
```

50개의 객체에 대해 랜덤하게 할당된 군집해($K = 3$) 두 개를 비교하여 랜드지수와 수정랜드지수를 구해보자.

```
random_rand_index <- function(n, r, s) {
  u <- sample.int(r, size = n, replace = TRUE)
  v <- sample.int(s, size = n, replace = TRUE)
  rand_index(u, v)
}

set.seed(500)

rerun(200, random_rand_index(50, 3, 3)) %>%
  bind_rows() %>%
  gather(key = "metric", value = "value") %>%
  ggplot(aes(x = value, fill = metric)) +
  geom_histogram(binwidth = 0.05) +
  scale_x_continuous(limits = c(-1, 1)) +
  scale_fill_discrete(
    name = "index",
    breaks = c("ri", "adj_ri"),
    labels = c("Rand Index", "adjusted Rand Index")
  ) +
  labs(
    title = "Distribution of index values from 200 random assignments",
    x = "index value",
    y = "frequency"
  )

## Warning: Removed 4 rows containing missing values (geom_bar).
```

Figure 14.1에 보이듯이, 랜덤한 군집해 간 비교에서 랜드지수는 0보다 큰 값을 나타내는 반면, 수정랜드지수는 0을 중심으로 분포되어 있다.

다음의 두 군집해에 대하여 랜드지수와 수정랜드지수를 구해보자.

$$\begin{aligned} U &= \{(1, 2, 3, 4), (5, 6, 7), (8, 9, 10)\} \\ V &= \{(1, 2, 5, 8), (3, 6, 9), (4, 7, 10)\} \end{aligned}$$

```
rand_index(
  c(1, 1, 1, 1, 2, 2, 2, 3, 3, 3),
  c(1, 1, 2, 3, 1, 2, 3, 1, 2, 3)
)
```

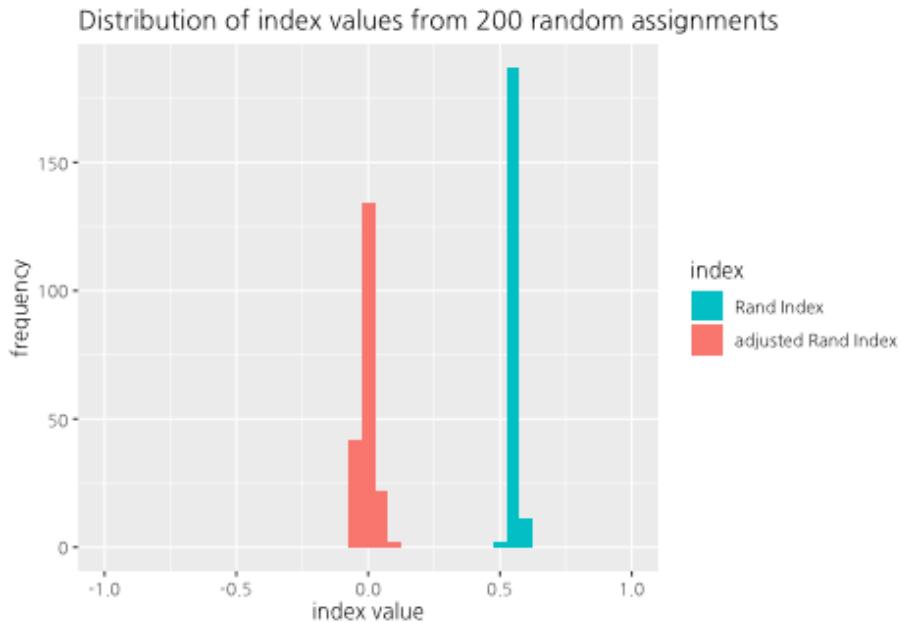


Figure 14.1: 랜덤한 군집해 간 비교: 랜드지수 및 수정랜드지수

```
## $ri
## [1] 0.5111111
##
## $adj_ri
## [1] -0.25
```

14.2.2 내부평가지수

군집해에 대한 내부평가지수는 외부 정보의 도움 없이 입력 데이터만으로 군집해를 평가하는 척도로써, 주로 밀집성(compactness), 연결성(connectedness), 분리성(spatial separation) 등 세 가지 관점에서 평가한다.

우선 K 개의 군집으로 이루어진 군집해 C 가 다음과 같다고 하자.

$$C = \{C_1, C_2, \dots, C_K\}$$

14.2.2.1 기본 R 스크립트

아래와 같이 두 개의 변수 x_1 및 x_2 로 표현되는 객체 데이터에 대해 군집을 찾고자 한다.

Table 14.1: 군집 대상 데이터

객체번호	\$x_1\$	\$x_2\$
1	4	15
2	20	13
3	3	13
4	19	4
5	17	17
6	8	11
7	19	12
8	18	6

```
df <- tribble(
  ~id, ~x1, ~x2,
  1, 4, 15,
  2, 20, 13,
  3, 3, 13,
  4, 19, 4,
  5, 17, 17,
  6, 8, 11,
  7, 19, 12,
  8, 18, 6
)

df %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('r', 'r', 'r'),
    col.names = c(
      '객체번호',
      '$x_1$', '$x_2$'
    ),
    caption = '군집 대상 데이터'
  )
```

위 데이터에 대해 다음과 같은 두 개의 다른 군집해를 얻었다고 하자.

$$\begin{aligned} U &= \{\{1, 3, 6\}, \{2, 5, 7\}, \{4, 8\}\} \\ V &= \{\{1, 3, 6\}, \{2, 4, 5, 7, 8\}\} \end{aligned} \tag{14.3}$$

```
sol_1 <- c(1, 2, 1, 3, 2, 1, 2, 3)
sol_2 <- c(1, 2, 1, 2, 2, 1, 2, 2)
```

본 장에서는 4가지 내부 평가지수를 다룬다. 아래는 R 패키지들에 속한 각각의 함수를 실행하여 그 결과를 리스트 형태로 리턴하는 사용자 정의 함수 `cluster_eval`를 구현한 것이다. 해당 함수에서 호출하는 R 패키지 함수들은 아래와 같다.

- Dunn index (Dunn, 1973): `clValid::dunn`
- CH index (Caliński and Harabasz, 1974): `fpc::calinhara`
- Connectivity (Handl and Knowles, 2005): `clValid::connectivity`
- Silhouettes (Rousseeuw, 1987): `cluster::silhouette`

각각의 평가지수에 대한 자세한 설명은 다음 장에서 하기로 한다.

아래 구현한 사용자 정의함수는 아래와 같은 입력 파라미터를 요구한다.

- `cluster`: 군집 해를 나타내는 길이 n , 최대값 K 의 정수형 벡터
- `df`: 군집 데이터
- `dist_method`: 거리 척도; Dunn, Connectivity, Silhouettes 지수 측정에 사용된다.
- `nn`: 최근 객체 수 (≥ 2); Connectivity 측정에 사용된다.

```
cluster_eval <- function(cluster, df, dist_method = "euclidean", nn = 2) {
  dunn_index <- clValid::dunn(
    Data = df,
    clusters = cluster,
    method = dist_method
  )

  ch_index <- fpc::calinhara(
    x = df,
    clustering = cluster
  )

  connectivity <- clValid::connectivity(
    Data = df,
    clusters = cluster,
    neighborSize = nn
  )

  asw <- cluster::silhouette(
    x = cluster,
    dist = dist(df, method = dist_method)
  )[ , "sil_width"] %>% mean()
```

```

    return(list(
      dunn_index = dunn_index,
      ch_index = ch_index,
      connectivity = connectivity,
      asw = asw
    ))
}

map_dfr(list(sol_1, sol_2), cluster_eval, df = df[, -1], .id = "solution")

## # A tibble: 2 x 5
##   solution dunn_index ch_index connectivity   asw
##   <chr>        <dbl>     <dbl>       <dbl> <dbl>
## 1 1            1.08      26.5        1  0.651
## 2 2            0.822     15.4        0  0.586

```

14.2.2.2 대표 내부평가지수

Dunn (1973) 은 아래와 같은 지수를 제안하였다.

$$DI = \frac{\min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j, 1 \leq i \neq j \leq K} d(\mathbf{x}, \mathbf{y})}{\max_{\mathbf{x} \in C_i, \mathbf{y} \in C_i, 1 \leq i \leq K} d(\mathbf{x}, \mathbf{y})} \quad (14.4)$$

여기서 분자는 군집 간의 분리성(클수록 분리성이 큼), 분모는 군집의 밀집성(작을수록 밀집성이 높음)을 반영하는 것이라 볼 수 있다. 따라서 분리성과 밀집성이 높을 때 식 (14.4)는 큰 값을 갖게 되며 해당 군집해가 상대적으로 좋게 평가된다.

군집해로부터 식 (14.4)를 계산하는 함수 `dunn_index`를 아래와 같이 구현해보자. 입력 파라미터는 아래와 같다.

- `cluster`: 각 객체가 속한 군집을 나타내는 길이 n 의 벡터
- `df`: 객체 데이터를 나타내는 n 행의 프레임
- `dist_method`: `base::dist` 함수의 `method` 파라미터값으로 사용할 거리 척도

```

dunn_index <- function(cluster, df, dist_method = "euclidean") {
  # 객체 수
  n <- nrow(df)

  # 각 객체의 군집해
  cluster_df <- tibble(
    id = 1:n,
    cluster = cluster
  )
}
```

```

# 각 객체 간 거리 데이터 프레임
# 위 군집해 데이터 프레임과 `inner_join`을 통해
# 두 객체가 동일 군집에 속하는지 서로 다른 군집에 속하는지 표현
dist_df <- dist(df, method = dist_method, upper = TRUE) %>%
  broom::tidy() %>%
  inner_join(
    cluster_df %>% rename(item1 = id, item1_cluster = cluster),
    by = "item1"
  ) %>%
  inner_join(
    cluster_df %>% rename(item2 = id, item2_cluster = cluster),
    by = "item2"
  )

# 서로 다른 군집에 속한 객체 쌍 중
# 가장 가까운 객체 간의 거리
numerator <- dist_df %>%
  filter(item1_cluster != item2_cluster) %>%
  top_n(-1, distance) %>%
  slice(1) %>%
  .\$distance

# 서로 같은 군집에 속한 객체 쌍 중
# 가장 먼 객체 간의 거리
denominator <- dist_df %>%
  filter(item1_cluster == item2_cluster) %>%
  top_n(1, distance) %>%
  slice(1) %>%
  .\$distance

res <- numerator / denominator

return(res)
}

map_dbl(list(sol_1, sol_2), dunn_index, df = df[, -1])

```

```
## [1] 1.075291 0.822375
```

Caliński and Harabasz (1974) 는 다음과 같은 지수를 제안하였다.

$$CH = \frac{\frac{1}{K-1} \sum_{k=1}^K n_k (\mathbf{c}_k - \mathbf{c})^\top (\mathbf{c}_k - \mathbf{c})}{\frac{1}{n-K} \sum_{k=1}^K \sum_{i \in C_k} (\mathbf{x}_i - \mathbf{c}_k)^\top (\mathbf{x}_i - \mathbf{c}_k)} \quad (14.5)$$

여기에서 \mathbf{c}_k 는 군집 k 의 중심좌표(centroid), \mathbf{c} 는 전체 객체들의 중심좌표, n_k 는 군집 C_k 내의 객체 수, n 은 전체 객체 수를 나타낸다. 식 (14.5) 또한 분자는 분리성, 분모는 밀집성을 평가하며, 값이 클수록 좋은 군집해로 평가된다.

```
ch_index <- function(cluster, df) {
  # 전체 데이터 중심
  centroid <- df %>% summarize_all(mean)

  cluster_df <- df %>%
    mutate(cluster = cluster) %>%
    group_by(cluster) %>%
    nest()

  # 군집 중심
  cluster_centroid_df <- map_dfr(cluster_df$data, ~ summarize_all(., mean))

  # 각 군집 중심과 전체 데이터 중심 간 제곱 유clidean 거리
  centroid_dist <- flexclust::dist2(
    cluster_centroid_df,
    centroid
  )^2

  # 각 군집 크기
  cluster_size <- map_dbl(cluster_df$data, nrow)

  numerator <- sum(centroid_dist * cluster_size) / (nrow(cluster_df) - 1)

  denominator <- map_dbl(
    cluster_df$data,
    # 각 군집 내의 객체와 군집 중심 간 제곱 유clidean 거리의 합
    ~ flexclust::dist2(., summarize_all(., mean))^2 %>% sum()
  ) %>%
    sum() %>%
    `/~(nrow(df) - nrow(cluster_df))

  res <- numerator / denominator

  return(res)
}

map_dbl(list(sol_1, sol_2), ch_index, df = df[, -1])

## [1] 26.45029 15.36218
```

Handl and Knowles (2005) 은 연결성을 반영한 아래와 같은 지수를 제시하고 있다.

$$Conn = \sum_{i=1}^n \sum_{j=1}^L v_{i,nn_i(j)} \quad (14.6)$$

이 때, $nn_i(j)$ 는 객체 i 의 j 번째 최근 객체(nearest neighbor)를 나타내며, L 은 연결성 척도 측정을 위한 사용자 지정 파라미터값이다. 또한 변수 $v_{i,nn_i(j)}$ 는 아래와 같이 정의 된다.

$$v_{i,nn_i(j)} = \begin{cases} 0 & \text{if } \exists C_k : i, nn_i(j) \in C_k \\ 1/j & \text{otherwise} \end{cases}$$

즉, 식 (14.6)은 각 객체가 $j (\leq L)$ 번째 최근 객체와 다른 객체에 속하면 $1/j$ 의 벌점을 부여하는 방식으로, 작은 값일수록 좋은 군집으로 평가될 수 있다.

```
connectivity <- function(cluster, df, dist_method = "euclidean", n_neighbor = 1) {
  n <- nrow(df)

  cluster_df <- tibble(
    id = 1:n,
    cluster = cluster
  )

  # 객체간 거리
  distance_df <- dist(df, method = dist_method, upper = TRUE) %>%
    broom::tidy()

  # 최근 객체
  nn_df <- distance_df %>%
    group_by(item1) %>%
    mutate(nearest = rank(distance, ties.method = "random")) %>%
    filter(nearest <= n_neighbor) %>%
    ungroup() %>%
    inner_join(
      cluster_df %>% rename(item1 = id, item1_cluster = cluster),
      by = "item1"
    ) %>%
    inner_join(
      cluster_df %>% rename(item2 = id, item2_cluster = cluster),
      by = "item2"
    )

  # 연결성 계산
  nn_df %>%
    filter(item1_cluster != item2_cluster) %>%
```

```

  mutate(v = 1 / nearest) %>%
  .$v %>%
  sum()

}

map_dbl(list(sol_1, sol_2), connectivity, df = df[, -1], n_neighbor = 1)

## [1] 0 0

map_dbl(list(sol_1, sol_2), connectivity, df = df[, -1], n_neighbor = 2)

## [1] 1 0

```

Rousseeuw (1987) 은 실루엣(silhouettes)이라는 내부평가지수를 제안하였다. 우선 다음과 같은 기호를 정의하자.

- $a(i)$: 객체 i 와 동일 군집에 속한 다른 객체들과의 평균 거리
- $d(i, C_k)$: 객체 i 와 다른 군집 C_k 에 속한 모든 객체들과의 평균 거리, $i \notin C_k$
- $b(i) = \min_{k:i \notin C_k} d(i, C_k)$

객체의 군집 멤버쉽 변수 z_{ik} 를

$$z_{ik} = \begin{cases} 1 & \text{if } i \in C_k \\ 0 & \text{otherwise} \end{cases}$$

라 정의하면, 위 $a(i)$ 와 $b(i)$ 를 아래와 같이 수식화할 수 있다.

$$\begin{aligned} a(i) &= \sum_{k=1}^K z_{ik} \frac{\sum_{j \neq i} z_{jk} d(\mathbf{x}_i, \mathbf{x}_j)}{\sum_{j \neq i} z_{jk}} \\ b(i) &= \max_{k:z_{ik} \neq 1} \frac{\sum_{j \neq i} z_{jk} d(\mathbf{x}_i, \mathbf{x}_j)}{\sum_{j \neq i} z_{jk}} \end{aligned}$$

이 때 객체 i 에 대한 실루엣은 아래와 같이 정의된다.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \quad (14.7)$$

식 (14.7)은 -1과 1 사이의 값을 갖는데, 1에 가까울수록 객체 i 가 비슷한 객체들과 군집된 것으로, -1에 가까울수록 먼 객체들과 군집된 것으로 판단할 수 있다.

객체 i 가 어떠한 다른 객체와도 같은 군집에 속하지 않는 경우가 발생할 수 있다 ($i \in C_k$, $|C_k| = 1$). 이 경우 $a(i)$ 가 정의되지 않으므로 $s(i)$ 값이 식 (14.7)에 의해서 정의되지 않는다. 이러한 경우에는 $s(i) = 0$ 이라고 실루엣을 정의한다.

$$s(i) = \begin{cases} \frac{b(i)-a(i)}{\max\{a(i), b(i)\}} & \text{if } a(i) \text{ is defined} \\ 0 & \text{otherwise} \end{cases} \quad (14.8)$$

이후 군집해의 평가지표로써 평균실루엣(overall average silhouette width; ASW)을 다음과 같이 정의하여 사용한다.

$$ASW = \frac{1}{n} \sum_{i=1}^n s(i) \quad (14.9)$$

```
asw <- function(cluster, df, dist_method = "euclidean") {
  n <- nrow(df)

  cluster_df <- tibble(
    id = 1:n,
    cluster = cluster
  )

  dist_df <- dist(df, method = dist_method, upper = TRUE) %>%
    broom::tidy() %>%
    inner_join(
      cluster_df %>% rename(item1 = id, item1_cluster = cluster),
      by = "item1"
    ) %>%
    inner_join(
      cluster_df %>% rename(item2 = id, item2_cluster = cluster),
      by = "item2"
    )

  dist_df <- dist_df %>%
    group_by(item1, item1_cluster, item2_cluster) %>%
    summarize(
      avg_distance = mean(distance)
    )

  a <- dist_df %>%
    filter(item1_cluster == item2_cluster) %>%
    .$avg_distance

  b <- dist_df %>%
    filter(item1_cluster != item2_cluster) %>%
    .$avg_distance
}
```

```

top_n(-1, avg_distance) %>%
  slice(1) %>%
  .$avg_distance

s <- map2_dbl(a, b, ~ (.y - .x) / max(.x, .y))

mean(s)
}

map_dbl(list(sol_1, sol_2), asw, df = df[, -1], dist_method = "euclidean")

## [1] 0.6507364 0.5864226

```

14.3 군집해의 해석

군집분석의 주목적을 달성하기 위해서는 군집해를 얻은 후 이에 대한 해석이 가능하여야 할 것이다. 즉, 각 군집의 특성을 파악할 수 있어야 실제 문제에 적용할 수 있을 것이다. 이를 위해서는 특정 응용분야의 전문가 지식을 요하는 경우가 많다. 그러나 첫 출발은 각 군집의 중심좌표, 즉 군집 별 변수별 평균치를 산출하는 것이다. 대부분의 경우 변수별 평균치로 군집들을 비교함으로써 군집의 특성을 파악할 수 있다. 다변량을 처리할 수 있는 그래프 역시 도움이 되며, 특히 다변량인 경우 요인분석(factor analysis)를 활용하기도 한다. 최종적으로 각 군집에 대한 특성이 파악되면 명명(naming)하는 것이 추천된다.

Part IV

4부 - 연관규칙

Chapter 15

연관규칙

```
library(tidyverse)
```

연관규칙 (association rule) 이란 간단히 말하면 항목들 간의 조건-결과 식으로 표현되는 유용한 패턴을 말한다. 연관규칙 탐사는 기업의 활동, 특히 마케팅에서 가장 널리 사용되고 있다.

15.1 필요 R package 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
arules	1.6-3

15.2 연관규칙의 정의 및 성능척도

데이터베이스가 총 n 개의 트랜잭션 데이터로 구성되며, 전체 m 개의 항목을 포함한다 하자. 전체 항목집합 I 를 다음과 같이 정의하자.

$$I = \{i_1, \dots, i_m\}$$

이 때, 각 트랜잭션 Z_j 는 I 의 부분집합이 된다.

$$Z_j \subseteq I, j = 1, \dots, n$$

연관규칙 R 은 조건부 X 와 결과부 Y 로 구성되어 ($X, Y \subseteq I$, $X \cap Y = \emptyset$) “ X 가 일어나면 Y 도 일어난다”는 의미로 아래와 같이 표현된다.

$$R : X \Rightarrow Y \quad (15.1)$$

식 (15.1)의 연관규칙 R 에 대한 성능척도로 지지도(support), 신뢰도(confidence) 및 개선도(lift)가 널리 사용된다.

15.2.1 지지도

지지도는 전체 트랜잭션 중 관심있는 항목집합을 포함하는 트랜잭션의 비율을 나타낸다.

항목 $X \subseteq I$ 에 대한 지지도는 아래와 같이 계산된다.

$$supp(X) = \frac{1}{n} \sum_{j=1}^n \mathbb{I}(X \subseteq Z_j)$$

이 때, $\mathbb{I}(a)$ 는 지시함수로 a 가 참일 때 1, 거짓일 때 0의 함수값을 가진다.

식 (15.1)의 연관규칙 R 에 대한 지지도는 아래와 같이 정의된다.

$$supp(R) = supp(X \cup Y)$$

다음과 같은 5개의 트랜잭션을 고려해보자.

```
transaction_df <- tribble(
  ~transaction_id, ~item,
  1, "b",
  1, "c",
  1, "g",
  2, "a",
  2, "b",
  2, "d",
  2, "e",
  2, "f",
  3, "a",
  3, "b",
  3, "c",
  3, "g",
  4, "b",
  4, "c",
  4, "e",
  4, "f",
  5, "b",
```

Table 15.1: 트랜잭션 데이터

트랜잭션	항목
1	b, c, g
2	a, b, d, e, f
3	a, b, c, g
4	b, c, e, f
5	b, c, e, f, g

```

5, "c",
5, "e",
5, "f",
5, "g"
)

transaction_df %>%
  group_by(transaction_id) %>%
  summarize(items = str_c(item, collapse = ", ")) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('c', 'c'),
    col.names = c('트랜잭션', '항목'),
    caption = '트랜잭션 데이터'
)

```

이 때, 전체 항목집합 I 는 $\{a, b, c, d, e, f, g\}$ 이다. 다음과 같은 규칙을 적용해보자.

$$R : \{b, c\} \Rightarrow \{g\}$$

이 때, 조건부 $X = \{b, c\}$ 에 대한 지지도는 아래와 같이 산출된다.

```

support <- function(group_df, item, set) {
  if(is_empty(set)) return(1)

  group_df %>%
    unique() %>%
    summarize(n = sum(!rlang::sym(item) %in% set)) %>%
    mutate(is_support = (n == length(set))) %>%
    {mean(.\$is_support)}
}

X <- c("b", "c")
group_transaction_df <- transaction_df %>% group_by(transaction_id)

```

```
support(group_transaction_df, item = "item", set = X)
```

```
## [1] 0.8
```

또한, 규칙 R 에 대한 지지도는 아래와 같이 산출할 수 있다.

```
Y <- c("g")
support(group_transaction_df, item = "item", set = union(X, Y))
```

```
## [1] 0.6
```

15.2.2 신뢰도

연관규칙 R 의 가치를 평가할 때, 통상 다음과 같이 정의되는 신뢰도를 사용한다.

$$conf(R) = \frac{supp(R)}{supp(X)} = \frac{supp(X \cup Y)}{supp(X)}$$

이 신뢰도는 조건부 확률의 개념으로, 집합 X (조건부)가 발생할 때 집합 Y (결과부)도 동시에 발생할 확률을 의미한다.

```
rule_confidence <- function(group_df, item, x, y) {
  support(group_df, item, union(x, y)) / support(group_df, item, x)
}
```

```
rule_confidence(group_transaction_df, item = "item", x = X, y = Y)
```

```
## [1] 0.75
```

15.2.3 개선도

결과가 단독으로 발생할 가능성에 비추어 조건과 연계하여 결과가 발생할 가능성의 빈도 비율로 정의한다.

$$lift(R) = \frac{conf(R)}{supp(Y)} = \frac{supp(X \cup Y)}{supp(X)supp(Y)}$$

```
rule_lift <- function(group_df, item, x, y) {
  rule_confidence(group_df, item, x, y) / support(group_df, item, y)
}
```

```
rule_lift(group_transaction_df, item = "item", x = X, y = Y)
```

```
## [1] 1.25
```

즉, 항목 b, c 가 발생할 때 g 가 발생하는 빈도가 25% 높아진다.

15.3 연관규칙의 탐사

연관규칙의 탐사는 결국 신뢰도 또는 개선도가 높은 규칙 R 을 트랜잭션 데이터로부터 도출하는 과정이다. 알고리즘으로 가장 널리 사용되는 것이 Apriori 알고리즘(Agrawal et al., 1994)이다.

15.3.1 빈발항목집합 생성

빈발항목집합(large itemsets)이란 미리 결정한 최소 지지도 s_{min} 이상의 지지도를 같은 모든 항목집합들을 뜻한다.

빈발항목집합 생성 과정은 아래와 같다.

1. k 개의 항목을 지닌 빈발항목집합 후보군 C_k 중 최소지지도 s_{min} 이상의 지지도를 같은 모든 항목집합들을 빈발항목집합 L_k 라 한다.
2. 빈발항목집합들 L_k 내의 각 쌍에 대해 합집합을 구하여 그 합집합의 크기(항목의 수)가 $k + 1$ 인 항목집합들을 빈발항목집합 후보군 C_{k+1} 라 한다.

k 를 1부터 증가시키면서 더 이상 빈발항목집합을 찾을 수 없을 때까지 위 과정을 반복한다. 이 때, 빈발항목집합 후보군을 생성하는 함수 `apriori_gen`을 아래와 같이 구현해보자.

```
apriori_gen <- function(L) {
  if(length(L) < 2) return(NULL)

  n_sets <- length(L)
  n_item <- unique(map_dbl(L, length))

  if(length(n_item) > 1) stop("All itemsets must be the same length.")

  C <- combn(L, m = 2, simplify = TRUE) %>%
    t() %>%
    `colnames<-`(`c("set1", "set2")`) %>%
    as_tibble() %>%
    pmap(function(set1, set2) {
      if(length(intersect(set1, set2)) != n_item - 1) return(NULL)
      sort(union(set1, set2))
    })
}
```

```

}) %>%
compact() %>%
unique()

C
}

```

위 함수를 이용하여, Table 15.1에서 최소 지지도 $s_{min} = 0.4$ 를 기준으로 빈발항목집합을 찾아보자.

```

s_min <- 0.4
group_transaction_df <- transaction_df %>% group_by(transaction_id)

candidate_itemsets <- as.list(sort(unique(transaction_df$item)))
large_itemsets <- vector("list", length = length(candidate_itemsets))

for(i in seq_along(large_itemsets)) {
  itemset_support <- map_dbl(candidate_itemsets,
    support,
    group_df = group_transaction_df,
    item = "item")

  large_itemsets[[i]] <- candidate_itemsets[itemset_support >= s_min]

  candidate_itemsets <- apriori_gen(large_itemsets[[i]])

  if(is.null(candidate_itemsets)) break
}

large_itemsets <- compact(large_itemsets)

```

찾아진 빈발항목집합들은 아래와 같다.

```

map(large_itemsets,
  ~map_chr(.x, ~str_c("{", str_c(.x, collapse = ", "), "}")))
## [[1]]
## [1] "{a}"  "{b}"  "{c}"  "{e}"  "{f}"  "{g}"
##
## [[2]]
## [1] "{a, b}"  "{b, c}"  "{b, e}"  "{b, f}"  "{b, g}"  "{c, e}"  "{c, f}"  "{c, g}"
## [9] "{e, f}"
##
## [[3]]

```

```
## [1] "{b, c, e}" "{b, c, f}" "{b, c, g}" "{b, e, f}" "{c, e, f}"
## [[4]]
## [1] "{b, c, e, f}"
```

15.3.2 규칙의 탐사

도출된 빈발항목집합 각각(L)을 조건부(X)와 결과부($Y = L \setminus A$)로 나눌 때 미리 결정된 최소 신뢰도 c_{\min} 이상의 신뢰도를 지닌 규칙 R 을 찾는다.

$$R : X \Rightarrow L \setminus X$$

우선, 빈발항목집합 L 으로부터 가능한 규칙들을 생성하는 함수 `generate_rules`를 아래와 같이 구현해보자.

```
generate_rules <- function(L, n_min_item = 1) {
  n_item <- length(L)
  if(n_item < n_min_item) return(NULL) # 항목 최소개수 제한

  X <- map(seq_len(n_item), ~combn(L, m = .x - 1, list)) %>% flatten()
  Y <- map(X, ~setdiff(L, .x))

  tibble(X = X, Y = Y)
}
```

앞 장에서 추출한 모든 빈발항목집합들로부터 규칙을 생성해보자.

```
rule_list <- map_dfr(
  large_itemsets %>% flatten(),
  generate_rules
)
```

각각의 규칙에 대하여 신뢰도를 계산하여, 그 값이 최소 신뢰도 c_{\min} 이상인 규칙만을 `conf_rule_list`라는 데이터 프레임으로 저장하자.

```
rule_list$confidence <- pmap_dbl(rule_list, function(X, Y)
  rule_confidence(group_transaction_df, item = "item", x = X, y = Y)
)

c_min <- 0.7
conf_rule_list <- rule_list %>% filter(confidence >= c_min)
```

이 결과 최종적으로 도출된 규칙들은 아래와 같다.

```
conf_rule_list %>%
  rowwise() %>%
  mutate(
    X = str_c("{", str_c(unlist(X), collapse = ", "), "}"),
    Y = str_c("{", str_c(unlist(Y), collapse = ", "), "}")
  ) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('c', 'c', 'c'),
    col.names = c('조건부 $X$', '결과부 $Y$', '신뢰도'),
    caption = '최종 연관규칙'
  )
)
```

항목의 수가 많은 경우, 생성 가능한 규칙의 수가 매우 많아, 보다 효율적인 탐사를 수행이 필요할 수 있다. 자세한 방법에 대해서는 교재 (전치혁, 2012) 참조.

15.3.3 R 패키지 내 Apriori

R 패키지 arules의 apriori 함수를 이용하여 위에서 살펴본 연관규칙 탐사를 수행할 수 있다.

우선, 15.2.1 절에서 생성한 데이터 프레임 transaction_df를 arules 패키지 내에 정의된 transactions 클래스의 데이터로 변환한다.

```
requireNamespace("arules")
transaction_df2 <- as(
  split(transaction_df$item, transaction_df$transaction_id),
  "transactions"
)
```

이후, apriori 함수를 호출하여 연관규칙 탐사를 수행한다.

```
rule_results <- arules::apriori(
  transaction_df2,
  parameter = list(
    support = 0.4,
    confidence = 0.7,
    target = "rules"
  ),
  control = list(
    verbose = FALSE
  )
)
```

결과로 얻어지는 `rules` 클래스 객체에서 필요한 정보를 추출하여 데이터 프레임으로 저장하자.

- `lhs`: 조건부
- `rhs`: 결과부
- `quality`: 평가척도 (지지도, 신뢰도, 개선도, 관측수)

```
rule_results_df <- tibble(
  X = as(rule_results@lhs, "list"),
  Y = as(rule_results@rhs, "list")
) %>%
  bind_cols(rule_results@quality)
```

해당 데이터 프레임은 아래와 같다.

```
rule_results_df %>%
  rowwise() %>%
  mutate(
    X = str_c("{", str_c(unlist(X), collapse = ", "), "}"),
    Y = str_c("{", str_c(unlist(Y), collapse = ", "), "}")
) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c('c', 'c', 'c', 'c', 'c', 'c'),
    col.names = c('조건부 $X$', '결과부 $Y$', '지지도',
                  '신뢰도', '개선도', '관측수'),
    caption = '최종 연관규칙 - arules::apriori'
)
```

위 Table 15.3를 살펴보면, 결과부에는 오직 하나의 항목만 존재하는 것을 알 수 있다. 이는 Apriori 알고리즘이 제안된 원 논문 (Agrawal et al., 1993)에 따른 것이며, 위 15.3.2 절에서 여러 개의 항목이 결과부에 존재하는 방식은 이 Apriori 알고리즘을 보다 일반화한 것이라 생각할 수 있겠다.

15.4 순차적 패턴의 탐사

순차적 패턴(sequential pattern)이란 고객들의 시간에 따른 구매 행태를 말하는데, 예를 들어 “냉장고를 구입한 후 김치냉장고를 구매한다”는 식이다.

순차적 패턴의 탐사를 위해서는 고객별, 시간별 트랜잭션 데이터가 필요하다. 항목 집합을 순서적으로 나열한 리스트를 시퀀스(sequence)라 하는데, A_j 를 j 번째의 항목집합이라 할 때, 시퀀스는 다음과 같이 표기한다.

$$s = \langle A_1, A_2, \dots, A_n \rangle$$

시퀀스에 포함된 항목집합의 수를 시퀀스의 길이라 하며, 길이가 k 인 시퀀스를 k -시퀀스라 한다.

$$\text{length}(< A_1, A_2, \dots, A_n >) = n$$

두 시퀀스 $s_1 = < A_1, A_2, \dots, A_n >$ 과 $s_2 = < B_1, B_2, \dots, B_m >$ 에 대하여 ($n \leq m$),

$$A_1 \subseteq B_{i_1}, A_2 \subseteq B_{i_2}, \dots, A_n \subseteq B_{i_n}$$

이 성립하는 $i_1 < i_2 < \dots < i_n$ 이 존재할 때, s_1 은 s_2 에 포함된다고 하며, 이 때 s_1 은 s_2 의 부분 시퀀스라 하며, 아래와 같이 표현한다.

$$s_1 \prec s_2$$

시퀀스 s 가 어떤 다른 시퀀스에 포함되지 않을 경우 최대 시퀀스(maximal sequence)라 한다.

N 명의 고객 각자에 대한 트랜잭션 시퀀스를 고객 시퀀스(customer sequence)라 하며, i 번째 고객에 대한 고객 시퀀스를 s_i 라 할 때 ($i = 1, \dots, N$), 임의의 시퀀스 s 에 대한 지지도를 다음과 같이 정의한다.

$$\text{supp}(s) = \frac{1}{N} \sum_{i=1}^N I(s \prec s_i)$$

그리고, 미리 정한 최소 지지도 이상을 갖는 시퀀스를 빈발 시퀀스(large sequence)라 한다. 따라서, 순차적 패턴 탐사 문제는 빈발 시퀀스 중 최대 시퀀스(maximal sequence)들을 찾는 것이라 할 수 있다.

15.4.1 AprioriAll 알고리즘

AprioriAll 알고리즘은 빈발 시퀀스를 탐색하나, 탐색된 시퀀스가 최대 빈발 시퀀스임을 보장하지는 못한다. 따라서, 후에 최대화 단계를 요한다.

아래와 같은 고객 시퀀스가 존재한다고 하자.

```
sequential_transaction_df <- tribble(
  ~customer_id, ~transaction_seq, ~item,
  1, 1, "a",
  1, 2, "b",
  2, 1, "c",
  2, 1, "d",
  2, 2, "a",
  2, 3, "e",
```

```

2, 3, "f",
2, 3, "g",
3, 1, "a",
3, 1, "h",
3, 1, "g",
4, 1, "a",
4, 2, "e",
4, 2, "g",
4, 3, "b",
5, 1, "b"
)

sequential_transaction_df %>%
  group_by(customer_id, transaction_seq) %>%
  summarize(itemset = str_c("{", str_c(item, collapse = ", "), "}") %>%
  summarize(sequence = str_c("<", str_c(itemset, collapse = ", "), ">")) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c("c", "c"),
    col.names = c("고객ID ($i$)", "고객 시퀀스 ($s_i$)"),
    caption = "고객별 시퀀스"
)

```

고객 시퀀스의 항목집합 또는 이의 부분집합 중 최소 지지도 이상인 것들을 빈발항목 집합으로 도출한다.

우선 시퀀스가 특정 패턴을 포함하는지 여부를 판단하는 사용자 정의 함수 `is_contained` 와 고객 시퀀스 집합의 특정 패턴에 대한 지지도를 산출하는 사용자 정의 함수 `support_sequence`를 아래와 같이 구현해보자.

```

is_contained <- function(x, pattern) {
  n_x <- length(x)
  n_pattern <- length(pattern)
  if (n_pattern == 0) return (TRUE)

  rtn <- FALSE

  location <- rep(NA_integer_, n_pattern)

  if (n_x >= n_pattern) {
    j <- 1L
    for(i in seq_len(n_x)) {
      if (is_empty(setdiff(pattern[[j]], x[[i]]))) {
        location[j] <- i
        j <- j + 1
        if (j > n_pattern) {

```

```

        rtn <- TRUE
        break
    }
}
}

rtn
}

support_sequence <- function(sequence_list, pattern) {
  map_lgl(sequence_list, is_contained, pattern = pattern) %>% mean()
}

```

앞 15.3.1절에서와 같이 사용자 정의 함수 `apriori_gen`을 이용하여 빈발항목집합(시퀀스 지지도 기준)을 아래와 같이 얻는다.

```

s_min <- 0.4

customer_sequence <- sequential_transaction_df %>%
  group_by(customer_id, transaction_seq) %>%
  summarize(itemset = list(item)) %>%
  summarize(sequence = list(itemset))

candidate_itemsets <- map(sort(unique(sequential_transaction_df$item)), ~list(.x))
large_itemsets <- vector("list", length = length(candidate_itemsets))

for (i in seq_along(large_itemsets)) {
  large_itemsets[[i]] <- candidate_itemsets[
    map_dbl(candidate_itemsets,
            ~support_sequence(customer_sequence$sequence, pattern = .x)) >= s_min
  ] %>% flatten()

  candidate_itemsets <- map(apriori_gen(large_itemsets[[i]]), ~list(.x))
}

large_itemsets <- large_itemsets %>% flatten()

```

위 결과, 아래와 같은 빈발항목집합이 얻어진다.

```
large_itemsets
```

```

## [[1]]
## [1] "a"

```

```
##  
## [[2]]  
## [1] "b"  
##  
## [[3]]  
## [1] "e"  
##  
## [[4]]  
## [1] "g"  
##  
## [[5]]  
## [1] "e" "g"
```

위 빈발항목집합에 일련번호를 부여한 뒤, 고객 시퀀스를 해당 일련번호를 이용한 시퀀스로 변환한다. 우선 아래와 같이 일련번호를 부여해보자.

```
large_itemset_df <- tibble(  
  itemset = large_itemsets,  
  mapped_to = seq_along(large_itemsets)  
)  
  
large_itemset_df %>%  
  rowwise() %>%  
  mutate(itemset = str_c("{", str_c(itemset, collapse = ", "), "}") ) %>%  
  knitr::kable(  
    booktabs = TRUE,  
    align = c("c", "c"),  
    col.names = c("빈발항목집합", "일련번호"),  
    caption = "고객 시퀀스 빈발항목집합"  
)
```

원 고객 시퀀스에 대해, 각 항목집합이 위 빈발항목집합을 포함하는 경우, 해당 일련번호가 항목으로 포함되는 형태로 변환 시퀀스를 생성한다.

```
customer_sequence$transformed_sequence <- customer_sequence$sequence %>%  
  map(~map(.x, function(x) {  
    large_itemset_df$mapped_to[  
      map_lgl(large_itemset_df$itemset, ~is_contained(x, .x))  
    ]  
  })) %>% compact()
```

원 고객 시퀀스와 변환 시퀀스는 아래와 같이 표현될 수 있다.

```

print_sequence <- function(sequence) {
  str_c("<", str_c(
    map(sequence, function(x)
      str_c(map_chr(x, ~str_c("{", str_c(.x, collapse = ", ")), "}")),
      collapse = ", "
    )), ">"))
}

customer_sequence %>%
  group_by(customer_id) %>%
  mutate(
    sequence = print_sequence(sequence),
    transformed_sequence = print_sequence(transformed_sequence)
  ) %>%
  knitr::kable(
    booktabs = TRUE,
    align = c("c", "c", "c"),
    col.names = c("고객ID", "고객 시퀀스", "변환 시퀀스"),
    caption = "고객 시퀀스의 변환"
  )

```

변환 시퀀스를 기준으로, 길이가 1인 빈발 시퀀스를 구한다. 최소 지지도를 앞에서 빈발 항목집합을 구할 때와 동일하게 설정할 때, 길이가 1인 빈발 시퀀스는 빈발항목집합(의 변환된 일련번호)과 동일하다.

우선, 최대 시퀀스 길이를 구하자.

```
max_sequence_length <- max(map_int(customer_sequence$transformed_sequence, ~length(.x)))
```

1-시퀀스인 빈발시퀀스는 빈발항목집합과 같다.

```

large_sequences <- vector("list", length = max_sequence_length)
large_sequences[[1]] <- map(large_itemset_df$mapped_to, ~list(.x))

```

같은 길이의 두 시퀀스를 이용해서 길이가 1 증가한 새로운 시퀀스를 생성하는 함수 generate_sequence를 아래와 같이 구현해보자. 새로운 시퀀스는 첫 번째 시퀀스 후에 두 번째 시퀀스의 가장 마지막 트랜잭션을 추가한 시퀀스이다.

```

generate_sequence <- function(seq1, seq2) {
  if (length(seq1) != length(seq2)) stop("Two sequences must be the same length.")

  # two k-sequences needs to be the same for first k-1 items to generate new sequence
  new_sequence <- NULL
  k <- length(seq1)

```

```

if (identical(seq1[seq_len(k - 1)], seq2[seq_len(k - 1)])) {
  new_sequence <- c(seq1, seq2[[k]])
}

new_sequence
}

```

빈발 k -시퀀스들로부터 $(k + 1)$ -시퀀스들을 생성하는 함수 `apriori_seq_gen`을 아래와 같이 구현해보자.

```

apriori_seq_gen <- function(L) {
  n_seqs <- length(L)
  n_item <- unique(map_dbl(L, length))

  if (length(n_item) > 1) stop("All sequences must be the same length.")

  k <- n_item

  # generate large new sequences with length (k+1)
  C <- vector("list", length = n_seqs * n_seqs)
  for (i in seq_along(L)) {
    for (j in seq_along(L)) {
      candidate_sequence <- generate_sequence(L[[i]], L[[j]])

      # check whether all subsequences with length k are element of L
      if (
        all(map_lgl(seq_len(k), function(x)
          any(map_lgl(L, ~identical(.x, candidate_sequence[-x]))))
      )
    ) {
      C[[n_seqs * (i - 1) + j]] <- candidate_sequence
    }
  }
}

compact(C)
}

```

위 `apriori_seq_gen` 함수 수행결과로 얻어지는 $(k + 1)$ -시퀀스들이 모두 빈발 시퀀스라는 보장은 없으므로, 새로 생성된 각 시퀀스가 최소 지지도 이상의 지지도를 갖는지 검토하여, 빈발 시퀀스만을 남기기로 하자. 앞에서 정의했던 함수 `support_sequence`를 활용하여, 새로운 함수 `get_large_sequence`를 정의하자.

```
get_large_sequence <- function(sequence_list, C, s_min) {
  is_large <- map_lgl(
    C, ~ support_sequence(sequence_list, .x) >= s_min
  )

  C[is_large]
}
```

위 빈발 k -시퀀스를 과정을 k 값을 1씩 증가시켜가며 더 이상 빈발 시퀀스를 찾을 수 없을 때까지 반복한다.

```
s_min <- 0.4

large_sequences <- vector("list", length = max_sequence_length)
large_sequences[[1]] <- map(large_itemset_df$mapped_to, ~list(.x))

for(i in seq_len(max_sequence_length - 1)) {
  large_sequences[[i + 1]] <- get_large_sequence(
    customer_sequence$transformed_sequence,
    apriori_seq_gen(large_sequences[[i]]),
    s_min
  )

  if(is_empty(large_sequences[[i + 1]])) break
}

large_sequences <- flatten(compact(large_sequences))
```

결과적으로 찾아진 빈발 시퀀스들은 아래와 같다.

```
map_chr(large_sequences,
~str_c("<", str_c(str_c("{", unlist(.x), "}"), collapse = ", "), ">"))

## [1] "<{1}>"      "<{2}>"      "<{3}>"      "<{4}>"      "<{5}>"
## [6] "<{1}, {2}>"  "<{1}, {3}>"  "<{1}, {4}>"  "<{1}, {5}>"
```

이후 최대화 단계를 통해, 빈발 시퀀스 중 최대 시퀀스들만 추출한다.

```
maximal_large_sequences <- large_sequences

for (i in seq(from = length(large_sequences), to = 2)) {
  if(!is_empty(maximal_large_sequences[i])) {
    is_subsequence <- map_lgl(maximal_large_sequences[seq_len(i - 1)],
```

```
~is_contained(maximal_large_sequences[i], .x))

  walk(which(is_subsequence), function(x) maximal_large_sequences[[x]] <- list())
}

maximal_large_sequences <- compact(maximal_large_sequences)
```

최대 빈발 시퀀스(변환 시퀀스 기준)은 아래와 같다.

```
map_chr(maximal_large_sequences,
~str_c("<", str_c(str_c("{", unlist(.x), "}"), collapse = ", "), ">"))

## [1] "<{1}, {2}>" "<{1}, {3}>" "<{1}, {4}>" "<{1}, {5}>"
```

이외에도 AprioriSome 알고리즘, DynamicSome 알고리즘 등의 시퀀스 탐사 알고리즘이 존재한다. 보다 자세한 내용은 전치혁 (2012) 및 Agrawal et al. (1995) 참고.

Table 15.2: 최종 연관규칙

조건부 \$X\$	결과부 \$Y\$	신뢰도
{}	{b}	1.00
{}	{c}	0.80
{a}	{b}	1.00
{}	{b, c}	0.80
{b}	{c}	0.80
{c}	{b}	1.00
{e}	{b}	1.00
{f}	{b}	1.00
{g}	{b}	1.00
{c}	{g}	0.75
{g}	{c}	1.00
{e}	{f}	1.00
{f}	{e}	1.00
{c, e}	{b}	1.00
{c, f}	{b}	1.00
{c}	{b, g}	0.75
{g}	{b, c}	1.00
{b, c}	{g}	0.75
{b, g}	{c}	1.00
{c, g}	{b}	1.00
{e}	{b, f}	1.00
{f}	{b, e}	1.00
{b, e}	{f}	1.00
{b, f}	{e}	1.00
{e, f}	{b}	1.00
{c, e}	{f}	1.00
{c, f}	{e}	1.00
{c, e}	{b, f}	1.00
{c, f}	{b, e}	1.00
{b, c, e}	{f}	1.00
{b, c, f}	{e}	1.00
{c, e, f}	{b}	1.00

Table 15.3: 최종 연관규칙 - arules::apriori

조건부 \$X\$	결과부 \$Y\$	지지도	신뢰도	개선도	관측수
{}	{c}	0.8	0.80	1.000000	4
{}	{b}	1.0	1.00	1.000000	5
{a}	{b}	0.4	1.00	1.000000	2
{g}	{c}	0.6	1.00	1.250000	3
{c}	{g}	0.6	0.75	1.250000	3
{g}	{b}	0.6	1.00	1.000000	3
{e}	{f}	0.6	1.00	1.666667	3
{f}	{e}	0.6	1.00	1.666667	3
{e}	{b}	0.6	1.00	1.000000	3
{f}	{b}	0.6	1.00	1.000000	3
{c}	{b}	0.8	1.00	1.000000	4
{b}	{c}	0.8	0.80	1.000000	4
{c, g}	{b}	0.6	1.00	1.000000	3
{b, g}	{c}	0.6	1.00	1.250000	3
{b, c}	{g}	0.6	0.75	1.250000	3
{c, e}	{f}	0.4	1.00	1.666667	2
{c, f}	{e}	0.4	1.00	1.666667	2
{e, f}	{b}	0.6	1.00	1.000000	3
{b, e}	{f}	0.6	1.00	1.666667	3
{b, f}	{e}	0.6	1.00	1.666667	3
{c, e}	{b}	0.4	1.00	1.000000	2
{c, f}	{b}	0.4	1.00	1.000000	2
{c, e, f}	{b}	0.4	1.00	1.000000	2
{b, c, e}	{f}	0.4	1.00	1.666667	2
{b, c, f}	{e}	0.4	1.00	1.666667	2

Table 15.4: 고객별 시퀀스

고객ID (\$i\$)	고객 시퀀스 (\$s_i\$)
1	<{a}, {b}>
2	<{c, d}, {a}, {e, f, g}>
3	<{a, h, g}>
4	<{a}, {e, g}, {b}>
5	<{b}>

Table 15.5: 고객 시퀀스 빈발항목집합

빈발항목집합	일련번호
{a}	1
{b}	2
{e}	3
{g}	4
{e, g}	5

Table 15.6: 고객 시퀀스의 변환

고객ID	고객 시퀀스	변환 시퀀스
1	<{a}, {b}>	<{1}, {2}>
2	<{c, d}, {a}, {e, f, g}>	<{1}, {3, 4, 5}>
3	<{a, h, g}>	<{1, 4}>
4	<{a}, {e, g}, {b}>	<{1}, {3, 4, 5}, {2}>
5	<{b}>	<{2}>

Chapter 16

추천시스템

```
library(tidyverse)
```

추천시스템(recommender system)은 상품, 웹페이지, 신문 기사 등에 대한 소비자의 성향을 파악하여 그에 부합하는 새로운 상품 등을 추천하고자 하는 목적으로 개발되며, 접근 방식에 따라 내용기반(content-based) 방법, 협업 필터링(collaborative filtering), 결합방식(hybrid) 등으로 분류된다.

16.1 필요 R package 설치

본 장에서 필요한 R 패키지들은 아래와 같다.

package	version
tidyverse	1.2.1
tidytext	0.2.0

16.2 내용기반 추천시스템

내용기반 추천시스템은 주로 문서 등의 추천에 활용되고 있다.

- N : 전체 문서의 수
- f_{ij} : 문서 j 에 나타난 단어 i 의 빈도수
- n_i : 단어 i 가 한 번 이상 나타난 문서의 수

우선 tidytext 패키지를 로드하자.

```
library(tidytext)
```

janeaustenr 패키지에 있는 Jane Austen의 6개 소설에 대한 텍스트 데이터를 로드하자. 해당 데이터는 책 내용이 담긴 `text`라는 컬럼과 책 제목인 `book` 컬럼으로 이루어진 데이터 프레임이다.

```
library(janeaustenr)
tidy_books <- austen_books()
head(tidy_books)
```

```
## # A tibble: 6 x 2
##   text                      book
##   <chr>                     <fct>
## 1 SENSE AND SENSIBILITY Sense & Sensibility
## 2 ""                        Sense & Sensibility
## 3 by Jane Austen           Sense & Sensibility
## 4 ""                        Sense & Sensibility
## 5 (1811)                   Sense & Sensibility
## 6 ""                        Sense & Sensibility
```

해당 데이터 프레임에 담긴 책의 수는 아래와 같다.

```
n_book <- nlevels(tidy_books$book)
print(n_book)
```

```
## [1] 6
```

책의 내용 `text`를 단어 단위로 나누어 각 행으로 저장하자.

```
tidy_words <- tidy_books %>% unnest_tokens(word, text)

head(tidy_words)
```

```
## # A tibble: 6 x 2
##   book                      word
##   <fct>                     <chr>
## 1 Sense & Sensibility sense
## 2 Sense & Sensibility and
## 3 Sense & Sensibility sensibility
## 4 Sense & Sensibility by
## 5 Sense & Sensibility jane
## 6 Sense & Sensibility austen
```

이 데이터 프레임을 기반으로, 단어 i 가 문서 j 에 나타난 단어 빈도수(term frequency)를 모든 단어 i 와 모든 문서 j 에 대해 계산하자.

$$TF_{ij} = \frac{f_{ij}}{\sum_k f_{kj}}$$

```
tf_results <- tidy_words %>%
  group_by(book, word) %>%
  summarize(n = n()) %>%
  mutate(tf = n / sum(n)) %>%
  select(-n) %>%
  ungroup() %>%
  complete(book, word, fill = list(tf = 0))
```

이 때, 단어 빈도수가 높은 단어들은 대체로 너무 흔한 단어들이어서 중요한 의미를 지니지 않은 경우가 많다. 아래와 같이, “the”, “to”, “and” 등의 단어들이 사용 빈도가 매우 높은 단어들이다.

```
tf_results %>% arrange(desc(tf)) %>% head(10)
```

```
## # A tibble: 10 x 3
##   book           word     tf
##   <fct>         <chr>   <dbl>
## 1 Northanger Abbey the    0.0409
## 2 Persuasion      the    0.0398
## 3 Mansfield Park the    0.0387
## 4 Pride & Prejudice the    0.0354
## 5 Sense & Sensibility to    0.0343
## 6 Sense & Sensibility the    0.0342
## 7 Mansfield Park  to    0.0341
## 8 Pride & Prejudice to    0.0341
## 9 Mansfield Park  and   0.0339
## 10 Persuasion     to    0.0336
```

따라서, 단어의 중요도를 정의할 때 단어 i 의 역문서 빈도수(inverse document frequency)를 함께 고려한다.

$$IDF_i = \log \frac{N}{n_i}$$

```
idf_results <- tf_results %>%
  filter(tf > 0) %>%
  group_by(word) %>%
  summarize(n = n()) %>%
```

```

mutate(idf = log(n_book / n)) %>%
select(-n)

idf_results %>% arrange(desc(idf)) %>% head(10)

```

```

## # A tibble: 10 x 2
##   word          idf
##   <chr>        <dbl>
## 1 _accepted_    1.79
## 2 _accident_    1.79
## 3 _adair_       1.79
## 4 _addition_    1.79
## 5 _advantages_  1.79
## 6 _affect_      1.79
## 7 _against_     1.79
## 8 _agreeable_   1.79
## 9 _air_         1.79
## 10 _allow_      1.79

```

최종적으로 단어의 중요도를 위해서 정의한 단어 빈도수와 역문서 빈도수의 곱으로 아래와 같이 구하며, 이를 TF-IDF 가중치라 한다.

$$w_{ij} = TF_{ij} \times IDF_i$$

```

tf_idf_results <- inner_join(tf_results, idf_results, by = "word") %>%
  mutate(tf_idf = tf * idf)

```

TF-IDF 가중치가 높은 단어들을 살펴보자.

```

tf_idf_results %>%
  arrange(desc(tf_idf)) %>%
  head(10)

```

```

## # A tibble: 10 x 5
##   book           word          tf      idf    tf_idf
##   <fct>        <chr>        <dbl>    <dbl>    <dbl>
## 1 Sense & Sensibility elinor    0.00519  1.79  0.00931
## 2 Sense & Sensibility marianne 0.00410  1.79  0.00735
## 3 Mansfield Park    crawford  0.00307  1.79  0.00551
## 4 Pride & Prejudice  darcy    0.00305  1.79  0.00547
## 5 Persuasion        elliot    0.00304  1.79  0.00544
## 6 Emma              emma     0.00488  1.10  0.00536
## 7 Northanger Abbey  tilney   0.00252  1.79  0.00452

```

```
## 8 Emma           weston    0.00242 1.79 0.00433
## 9 Pride & Prejudice bennet    0.00241 1.79 0.00431
## 10 Persuasion    wentworth 0.00228 1.79 0.00409
```

대체로 소설에 나타나는 인물의 이름이 높은 가중치를 보이는데, 이는 인물의 이름이 소설 한 권에 걸쳐 여러 번 나타나 단어 빈도수가 높으며, 또한 각각의 소설이 서로 다른 인물명을 등장시킴으로써 역문서 빈도수 또한 높기 때문이다.

위와 같은 TF-IDF 가중치 계산은 tidytext 패키지의 bind_tf_idf 함수를 이용하여 간편하게 구할 수 있다.

```
tidy_words %>%
  group_by(book, word) %>%
  summarize(n = n()) %>%
  ungroup() %>%
  bind_tf_idf(word, book, n) %>%
  arrange(desc(tf_idf)) %>%
  head(10)

## # A tibble: 10 x 6
##   book          word      n      tf     idf   tf_idf
##   <fct>        <chr>  <int>  <dbl>  <dbl>   <dbl>
## 1 Sense & Sensibility elinor  623 0.00519 1.79 0.00931
## 2 Sense & Sensibility marianne 492 0.00410 1.79 0.00735
## 3 Mansfield Park   crawford 493 0.00307 1.79 0.00551
## 4 Pride & Prejudice darcy   373 0.00305 1.79 0.00547
## 5 Persuasion       elliot   254 0.00304 1.79 0.00544
## 6 Emma            emma    786 0.00488 1.10 0.00536
## 7 Northanger Abbey tilney   196 0.00252 1.79 0.00452
## 8 Emma            weston   389 0.00242 1.79 0.00433
## 9 Pride & Prejudice bennet   294 0.00241 1.79 0.00431
## 10 Persuasion      wentworth 191 0.00228 1.79 0.00409
```

임의의 사용자 u 가 아래와 같이 다섯 가지 단어에 각기 다른 관심도 w_{iu} 를 지닌다고 하자.

```
words_of_interest <- tibble(
  word = c("kitty", "cottage", "judgment", "war", "sea"),
  weight = c(0.3, 0.3, 0.1, 0.1, 0.2)
)

words_of_interest %>%
  knitr::kable(
  booktabs = TRUE,
  align = c('c', 'c'),
  col.names = c('단어 ($i$)', '가중치 ($w_{iu}$)'),
```

Table 16.1: 목표 사용자의 관심 단어 및 가중치

단어 (\$i\$)	가중치 (\$w_{iu}\$)
kitty	0.3
cottage	0.3
judgment	0.1
war	0.1
sea	0.2

```
caption = '목표 사용자의 관심 단어 및 가중치'
)
```

이 때, 목표 사용자 u 의 문서 j 에 대한 유용도(utility)를 다음과 같이 코사인 유사성 척도(cosine similarity measure)로 산출한다.

$$u(a, j) = \frac{\sum_{i=1}^K w_{iu} w_{ij}}{\sqrt{\sum_{i=1}^K w_{iu}^2} \sqrt{\sum_{i=1}^K w_{ij}^2}}$$

```
utility_results <- tf_idf_results %>%
  inner_join(words_of_interest, by = "word") %>%
  group_by(book) %>%
  summarize(utility = sum(weight * tf_idf) /
            (sqrt(sum(weight ^ 2)) * sqrt(sum(tf_idf ^ 2)))) %>%
  arrange(desc(utility))

print(utility_results)
```

```
## # A tibble: 6 x 2
##   book           utility
##   <fct>        <dbl>
## 1 Persuasion    0.753
## 2 Emma          0.710
## 3 Sense & Sensibility 0.640
## 4 Pride & Prejudice   0.615
## 5 Northanger Abbey 0.529
## 6 Mansfield Park  0.404
```

위 결과 Persuasion이 목표 사용자의 관심에 가장 유용도 높은 문서로 추천된다.

교재 전치혁 (2012) 예제에 대한 R 스크립트를 구현해보자.

```
words_of_interest <- tribble(  
  ~word, ~weight,  
  "word1", 0.124,  
  "word2", 0.275,  
  "word3", 0.019,  
  "word4", 0.182,  
  "word5", 0.223  
)  
  
tf_idf_results <- tribble(  
  ~document, ~word, ~tf_idf,  
  "doc1", "word1", 0.0194,  
  "doc1", "word2", 0.0043,  
  "doc1", "word3", 0.0054,  
  "doc1", "word4", 0.0155,  
  "doc1", "word5", 0.0028,  
  "doc2", "word1", 0.0082,  
  "doc2", "word2", 0.0032,  
  "doc2", "word3", 0.0007,  
  "doc2", "word4", 0.0104,  
  "doc2", "word5", 0.0073,  
  "doc3", "word1", 0.0087,  
  "doc3", "word2", 0.0174,  
  "doc3", "word3", 0.0091,  
  "doc3", "word4", 0.0086,  
  "doc3", "word5", 0.0268,  
  "doc4", "word1", 0.0093,  
  "doc4", "word2", 0.0061,  
  "doc4", "word3", 0.0172,  
  "doc4", "word4", 0.0028,  
  "doc4", "word5", 0.0009,  
  "doc5", "word1", 0.0185,  
  "doc5", "word2", 0.0249,  
  "doc5", "word3", 0.0084,  
  "doc5", "word4", 0.0167,  
  "doc5", "word5", 0.0193,  
  "doc6", "word1", 0.0028,  
  "doc6", "word2", 0.0003,  
  "doc6", "word3", 0.0202,  
  "doc6", "word4", 0.0083,  
  "doc6", "word5", 0.0054  
)  
  
utility_results <- tf_idf_results %>%  
  inner_join(words_of_interest, by = "word") %>%
```

```

group_by(document) %>%
  summarize(utility = sum(weight * tf_idf) /
            (sqrt(sum(weight ^ 2)) * sqrt(sum(tf_idf ^ 2)))) %>%
  arrange(desc(utility))

print(utility_results)

## # A tibble: 6 x 2
##   document utility
##   <chr>      <dbl>
## 1 doc5        0.972
## 2 doc3        0.919
## 3 doc2        0.841
## 4 doc1        0.659
## 5 doc4        0.448
## 6 doc6        0.373

```

위 결과, 두 건의 문서를 추천할 경우 doc5, doc3를 추천할 수 있다.

16.3 협업 필터링

총 m 개의 상품에 대한 n 명의 소비자의 평점이 있다고 할 때, 관련 기호를 다음과 같이 정의하자.

- v_{ij} : 고객 i 의 상품 j 에 대한 평점
- I_i : 고객 i 가 평점을 매긴 상품집합
- $|I_i|$: 집합 I_i 에 포함된 상품 수

이 때, 고객 i 의 평균 평점은 다음과 같이 산출된다.

$$\bar{v}_i = \frac{1}{|I_i|} \sum_{j \in I_i} v_{ij}$$

이 때, 목표고객 a 와 i 번째 고객과의 유사성은 아래와 같이 평점에서 고객 평점을 뺀 (mean-centering) 값에 대한 코사인 유사성 척도를 이용하여 정의한다.

$$w(a, i) = \frac{\sum_{j \in I_a \cap I_i} (v_{aj} - \bar{v}_a)(v_{ij} - \bar{v}_i)}{\sqrt{\sum_{j \in I_a \cap I_i} (v_{aj} - \bar{v}_a)^2} \sqrt{\sum_{j \in I_a \cap I_i} (v_{ij} - \bar{v}_i)^2}}$$

이를 이용하여, 목표고객 a 가 아직 구매하지 않은 상품 j 에 매길 평점을 아래와 같이 추정한다.

$$\hat{v}_{aj} = \bar{v}_a + \frac{1}{\sum_{i=1}^n |w(a, i)|} \sum_{i=1}^n w(a, i)(v_{ij} - \bar{v}_i)$$

교재 @jun2012datamining 에 있는 예제에 대한 R 스크립트를 구현해보자.

```
rating_df <- tribble(
  ~customer, ~item, ~rating,
  "고객 1", "상품 1", 5,
  "고객 1", "상품 3", 4,
  "고객 1", "상품 5", 1,
  "고객 1", "상품 6", 0,
  "고객 1", "상품 7", 3,
  "고객 2", "상품 1", 4,
  "고객 2", "상품 2", 4,
  "고객 2", "상품 3", 4,
  "고객 2", "상품 7", 1,
  "고객 3", "상품 1", 5,
  "고객 3", "상품 2", 4,
  "고객 3", "상품 4", 1,
  "고객 3", "상품 5", 2,
  "고객 3", "상품 7", 3,
  "고객 4", "상품 1", 1,
  "고객 4", "상품 2", 2,
  "고객 4", "상품 3", 1,
  "고객 4", "상품 4", 4,
  "고객 4", "상품 5", 3,
  "고객 4", "상품 6", 5,
  "고객 4", "상품 7", 2,
  "고객 5", "상품 1", 0,
  "고객 5", "상품 2", 1,
  "고객 5", "상품 4", 3,
  "고객 5", "상품 5", 5,
  "고객 5", "상품 6", 5,
  "고객 6", "상품 2", 2,
  "고객 6", "상품 5", 4,
  "고객 6", "상품 6", 4,
  "고객 6", "상품 7", 2,
  "목표고객", "상품 1", 5,
  "목표고객", "상품 4", 1,
  "목표고객", "상품 7", 2
)
```

우선, 각 고객이 매긴 평균 평점 \bar{v}_i 을 각 아이템에 대한 평점 v_{ij} 에서 제외하여 mean_centered rating을 아래와 같이 구한다.

```

centered_rating_df <- rating_df %>%
  group_by(customer) %>%
  mutate(centered_rating = rating - mean(rating)) %>%
  ungroup()

print(centered_rating_df)

## # A tibble: 33 x 4
##   customer item    rating centered_rating
##   <chr>     <chr>    <dbl>           <dbl>
## 1 고객 1   상품 1     5            2.4
## 2 고객 1   상품 3     4            1.4
## 3 고객 1   상품 5     1           -1.6
## 4 고객 1   상품 6     0           -2.6
## 5 고객 1   상품 7     3            0.400
## 6 고객 2   상품 1     4            0.75
## 7 고객 2   상품 2     4            0.75
## 8 고객 2   상품 3     4            0.75
## 9 고객 2   상품 7     1           -2.25
## 10 고객 3  상품 1     5            2
## # ... with 23 more rows

```

목표 고객과 다른 고객들간의 유사성 척도를 계산한다.

```

similarity_df <- centered_rating_df %>% filter(customer == "목표고객") %>%
  inner_join(centered_rating_df %>% filter(customer != "목표고객"), by = "item") %>%
  group_by(customer.y) %>%
  summarize(similarity = sum(centered_rating.x * centered_rating.y) /
            (sqrt(sum(centered_rating.x ^ 2)) * sqrt(sum(centered_rating.y ^ 2)))) %
  rename(customer = customer.y)

print(similarity_df)

## # A tibble: 6 x 2
##   customer similarity
##   <chr>        <dbl>
## 1 고객 1      0.903
## 2 고객 2      0.565
## 3 고객 3      0.961
## 4 고객 4     -0.875
## 5 고객 5     -0.853
## 6 고객 6       1

```

유사성 척도의 절대값의 합이 1이 되도록 normalize한다.

```
normalized_similarity_df <- similarity_df %>%
  mutate(normalized_similarity = similarity / sum(abs(similarity)))

print(normalized_similarity_df)

## # A tibble: 6 x 3
##   customer      normalized_similarity
##   <chr>          <dbl>
## 1 고객 1        0.175
## 2 고객 2        0.109
## 3 고객 3        0.186
## 4 고객 4       -0.170
## 5 고객 5       -0.165
## 6 고객 6        0.194
```

이후 목표고객이 아직 평점을 매기지 않은 상품들에 대해 평점을 추정한다. 이 때, 상품 j 에 대해 평점을 매기지 않은 고객의 경우, $v_{ij} - \bar{v}_i = 0$ 이라 가정하자.

$$\hat{v}_{aj} = \bar{v}_a + \frac{1}{\sum_{i=1}^n |w(a, i)|} \sum_{i=1}^n w(a, i)(v_{ij} - \bar{v}_i)$$

```
items <- sort(setdiff(unique(rating_df$item),
                      rating_df$item[rating_df$customer == "목표고객"]))

target_mean <- mean(rating_df$rating[rating_df$customer == "목표고객"])

centered_rating_df %>%
  filter(item %in% items) %>%
  inner_join(normalized_similarity_df, by = "customer") %>%
  group_by(item) %>%
  summarize(predicted_rating = target_mean +
            sum(normalized_similarity * centered_rating)) %>%
  arrange(desc(predicted_rating))

## # A tibble: 4 x 2
##   item      predicted_rating
##   <chr>          <dbl>
## 1 상품 3        3.26
## 2 상품 2        3.14
## 3 상품 5        1.96
## 4 상품 6        1.63
```

이번에는, 목표상품 j 에 대한 평점을 추정할 때, 상품 j 에 대해 평점을 매긴 고객과의 유사성만을 아래와 같이 고려하기로 하자.

$$\hat{v}_{aj} = \bar{v}_a + \frac{1}{\sum_{i:j \in I_i} |w(a, i)|} \sum_{i:j \in I_i} w(a, i)(v_{ij} - \bar{v}_i)$$

```
centered_rating_df %>%
  filter(item %in% items) %>%
  inner_join(similarity_df, by = "customer") %>%
  group_by(item) %>%
  summarize(predicted_rating = target_mean +
            sum(similarity * centered_rating) / sum(abs(similarity))) %>%
  arrange(desc(predicted_rating))

## # A tibble: 4 x 2
##   item   predicted_rating
##   <chr>      <dbl>
## 1 상품 3      3.97
## 2 상품 2      3.24
## 3 상품 5      1.87
## 4 상품 6      1.19
```

16.4 시장바구니 데이터를 이용한 협업 필터링

아래와 같은 시장바구니 데이터가 있다.

$$v_{ij} = \begin{cases} 1 & \text{고객 } i \text{가 상품 } j \text{를 구매한 경우} \\ 0 & \text{그렇지 않은 경우} \end{cases}$$

```
market_basket_df <- tribble(
  ~customer, ~item, ~purchase,
  "고객 1", "상품 1", 1,
  "고객 1", "상품 3", 1,
  "고객 1", "상품 5", 1,
  "고객 1", "상품 7", 1,
  "고객 2", "상품 1", 1,
  "고객 2", "상품 2", 1,
  "고객 2", "상품 3", 1,
  "고객 2", "상품 7", 1,
  "고객 3", "상품 1", 1,
  "고객 3", "상품 2", 1,
  "고객 3", "상품 4", 1,
```

```

    "고객 3", "상품 5", 1,
    "고객 3", "상품 7", 1,
    "고객 4", "상품 1", 1,
    "고객 4", "상품 2", 1,
    "고객 4", "상품 3", 1,
    "고객 4", "상품 4", 1,
    "고객 4", "상품 6", 1,
    "고객 4", "상품 7", 1,
    "고객 5", "상품 2", 1,
    "고객 5", "상품 4", 1,
    "고객 5", "상품 5", 1,
    "고객 5", "상품 6", 1,
    "고객 6", "상품 2", 1,
    "고객 6", "상품 5", 1,
    "고객 6", "상품 6", 1,
    "고객 6", "상품 7", 1,
    "목표고객", "상품 1", 1,
    "목표고객", "상품 4", 1,
    "목표고객", "상품 7", 1
)

```

이 때, 총 상품의 개수를 m 이라 하고, 고객 i 에 대해

$$p_i = \frac{|I_i|}{m}$$

이라 정의하자. 즉, p_i 는 전체 상품 중 고객 i 가 구입한 상품의 비율을 뜻한다. 또한, 두 고객 i 와 k 가 공통적으로 구입한 상품의 비율을 아래와 같이 p_{ik} 라 정의하자.

$$p_{ik} = \frac{|I_i \cap I_k|}{m}$$

우선 아래와 같이 가중치 $w(a, i)$ 를 계산해보자. 이 가중치는 목표고객 a 과 각 고객 i 간의 유사성 척도이다.

$$w(a, i) = \frac{p_{ai} - p_a p_i}{\sqrt{p_a(1-p_a)} \sqrt{p_i(1-p_i)}}$$

```

m <- length(unique(market_basket_df$item))

n_df <- market_basket_df %>%
  group_by(customer) %>%
  summarize(p = n() / m)

common_df <- market_basket_df %>%

```

```

filter(customer == "목표고객") %>%
  inner_join(market_basket_df %>% filter(customer != "목표고객"),
             by = "item") %>%
  group_by(customer.y) %>%
  summarize(p = n() / m) %>%
  rename(customer = customer.y)

similarity_df <- crossing(
  target_customer = "목표고객",
  ref_customer = n_df$customer[n_df$customer != "목표고객"])
) %>%
  inner_join(n_df %>% rename(target_p = p),
             by = c("target_customer" = "customer")) %>%
  inner_join(n_df %>% rename(ref_p = p),
             by = c("ref_customer" = "customer")) %>%
  inner_join(common_df %>% rename(common_p = p),
             by = c("ref_customer" = "customer")) %>%
  mutate(
    similarity = (common_p - target_p * ref_p) /
      (sqrt(target_p * (1 - target_p)) * sqrt(ref_p * (1 - ref_p)))
  )

print(similarity_df)

## # A tibble: 6 x 6
##   target_customer ref_customer target_p ref_p common_p similarity
##   <chr>           <chr>        <dbl>  <dbl>    <dbl>      <dbl>
## 1 목표고객        고객 1       0.429  0.571    0.286     0.167
## 2 목표고객        고객 2       0.429  0.571    0.286     0.167
## 3 목표고객        고객 3       0.429  0.714    0.429     0.548
## 4 목표고객        고객 4       0.429  0.857    0.429     0.354
## 5 목표고객        고객 5       0.429  0.571    0.143     -0.417
## 6 목표고객        고객 6       0.429  0.571    0.143     -0.417

```

이후 목표고객이 아직 구매하지 않은 상품에 대해 평점을 추정한다. 목표고객 a 의 상품 j 에 대한 평점 추정치는 다음과 같이 산출한다.

$$P_{aj} = \frac{\sum_{i=1}^n w(a, i)v_{ij}}{\sum_{i=1}^n |w(a, i)|}$$

```

denom <- sum(abs(similarity_df$similarity))

pred_df <- similarity_df %>%
  inner_join(market_basket_df, by = c("ref_customer" = "customer")) %>%

```

```
anti_join(market_basket_df %>%
  filter(customer == "목표고객") %>%
  select(item),
  by = "item") %>%
group_by(item) %>%
summarize(est_score = sum(similarity * purchase) / denom) %>%
arrange(desc(est_score))

print(pred_df)

## # A tibble: 4 x 2
##   item    est_score
##   <chr>     <dbl>
## 1 상품 3     0.332
## 2 상품 2     0.113
## 3 상품 5    -0.0575
## 4 상품 6    -0.232
```


Bibliography

- Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, New York, NY, USA. ACM.
- Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499.
- Agrawal, R., Srikant, R., et al. (1995). Mining sequential patterns. In *icde*, volume 95, pages 3–14.
- Banfield, J. D. and Raftery, A. E. (1993). Model-based gaussian and non-gaussian clustering. *Biometrics*, pages 803–821.
- Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.
- Calinński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27.
- Cavanaugh, J., Hatch, R., and Sullivan, J. (1976). Models for the subjective effects of loss, noise, and talker echo on telephone connections. *Bell System Technical Journal*, 55(9):1319–1371.
- Celeux, G. and Govaert, G. (1995). Gaussian parsimonious clustering models. *Pattern recognition*, 28(5):781–793.
- Chang, C.-C. and Lin, C.-J. (2001). Training v-support vector classifiers: theory and algorithms. *Neural computation*, 13(9):2119–2147.
- Chang, C.-C. and Lin, C.-J. (2011). Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27.
- Czepiel, S. A. (2002). Maximum likelihood estimation of logistic regression models: theory and implementation. Available at czep.net/stat/mlelr.pdf.

- Dunn, J. C. (1973). A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters.
- Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005). Working set selection using second order information for training support vector machines. *Journal of machine learning research*, 6(Dec):1889–1918.
- Goldfarb, D. and Idnani, A. (1983). A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical programming*, 27(1):1–33.
- Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, pages 857–871.
- Handl, J. and Knowles, J. (2005). Exploiting the trade-off—the benefits of multiple objectives in data clustering. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 547–560. Springer.
- Hubert, L. and Arabie, P. (1985). Comparing partitions. *Journal of classification*, 2(1):193–218.
- Kaufman, L. and Rousseeuw, P. J. (1990). Finding groups in data: an introduction to cluster analysis.
- Lance, G. N. and Williams, W. T. (1967). A general theory of classificatory sorting strategies: 1. hierarchical systems. *The computer journal*, 9(4):373–380.
- Park, H.-S. and Jun, C.-H. (2009). A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341.
- Pratt, J. W. (1981). Concavity of the log likelihood. *Journal of the American Statistical Association*, 76(373):103–106.
- Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850.
- Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.
- Schölkopf, B., Smola, A. J., Williamson, R. C., and Bartlett, P. L. (2000). New support vector algorithms. *Neural computation*, 12(5):1207–1245.
- Scrucca, L., Fop, M., Murphy, T. B., and Raftery, A. E. (2016). mclust 5: Clustering, classification and density estimation using gaussian finite mixture models. *The R journal*, 8(1):289.
- Ward Jr, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244.

Wishart, D. (1969). 256. note: An algorithm for hierarchical classifications. *Biometrics*, pages 165–170.

전치혁 (2012). 데이터마이닝 기법과 응용. 한나래출판사.