Statistical_Mechine_Learning_Final

Hyeonho Lee

2018년 9월 9일

Question 1

It is well-known that ridge regression tends to give similar coeffcient values to correlated variables, whereas the lasso may give quite different coefficient values to correlated vari-ables. We will now explore this property in a very simple setting.

Suppose that $n = 2, p = 2, x_{11} = x_{12}, x_{21} = x_{22}$ Furthermore, suppose that $y_1 + y_2 = 0$ and $x_{11} + x_{21} = 0$ and $x_{12} + x_{22} = 0$, so that the estimate for the intercept in a least squares, ridge regression, or lasso model is zero: $\hat{\beta}_0$.

1. Write out the ridge regression optimiztion problem in this setting.

$Answer$:

A general form of Ridge regression optimization looks like :

$\sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \sum_{j=1}^{p}\hat{\beta}_j x_j^2 + \lambda\sum_{i=1}^{p}\hat{\beta}_j^2$

In this case, $\hat{\beta}_0 = 0$ and $n = p = 2$.

So, the optimization looks like $Minimize : (y_1 - \hat{\beta}_1 x_{11} - \hat{\beta}_2 x_{12})^2 + (y_2 - \hat{\beta}_1 x_{22})^2 + \lambda(\hat{\beta}_1^2 + \hat{\beta}_2^2)$

2. Argue that in this setting, the ridge coefficient estimates satisfy $\hat{\beta}_1 = \hat{\beta}_2$

$Answer$:

Argue that in this setting, the ridge coefficient estimates satisfy $\hat{\beta}_1 = \hat{\beta}_2$. Given the situations that $x_{11} = x_{12} = x_1, x_{21} = x_{22} = x_2$, take the derivatives of the expression in (a) with respect to both $\hat{\beta}_1$ and $\hat{\beta}_2$ and setting them equal zero, then we get

$\hat{\beta}_1^* = \frac{x_1 y_1 + x_2 y_2 - \hat{\beta}_2^*(x_1^2 + x_2^2)}{\lambda + x_1^2 + x_2^2}$

$\hat{\beta}_2^* = \frac{x_1 y_1 + x_2 y_2 - \hat{\beta}_1^*(x_1^2 + x_2^2)}{\lambda + x_1^2 + x_2^2}$

The symmetry form in the above formula suggests that $\hat{\beta}_1 = \hat{\beta}_2$

3. Write ou the lasso optimization problem in this setting.

$Answer$:

The optimization looks like

$Minimize : (y_1 - \hat{\beta}_1 x_{11} - \hat{\beta}_2 x_{12})^2 + (y_2 - \hat{\beta}_1 x_{21} - \hat{\beta}_2 x_{22})^2 + \lambda(|\hat{\beta}_1| + \hat{\beta}_2)$

4. Argue that in this setting, the lasso coefficients $\hat{\beta}_1$ and $\hat{\beta}_2$ are not unique-in other wores, there are man possible solutions to the optimization problem in 3. Describe these solutions.

$Answer$: The Lasso contraint takes the form $|\hat{\beta}_1| + |\hat{\beta}_2| < s$ which plotted takes the shape of a diamond centered at origin (0,0). Next consider the sdquared optimization constrain $(y_1 - \hat{\beta}_1 x_{11} - \hat{\beta}_2 x_{12})^2 + (y_2 - \hat{\beta}_1 x_{21} - \hat{\beta}_2 x_{22})^2$. We use the facts $x_{11} = x_{12}$, $x_{21} = x_{22}$, $x_{11} + x_{21} = 0$, $x_{12} + x_{22} = 0$, and $y_1 + y_2 = 0$ to simlify is to minimize: $2(y_1 - (\hat{\beta}_1 + \hat{\beta}_2) x_{11})^2$.

This optimization problem has a simple solution: $\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_{11}}$. this is a line parallel to the edge of Lasso-diamond $\hat{\beta}_1 + \hat{\beta}_2 = s$. Now the soluitions to the original Lasso optimization problem are contours of the function $(y_1 - (\hat{\beta}_1 + \hat{\beta}_2) x_{11})^2$ that touch the Lasso-diamond $\hat{\beta}_1 + \hat{\beta}_2 = s$. Finally, as $\hat{\beta}_1$ and $\hat{\beta}_2$ vary along the line $\hat{\beta}_1 + \hat{\beta}_2 = \frac{y_1}{x_{11}}$, these coniours touch the Lasso-diamond edge $\hat{\beta}_1 + \hat{\beta}_2 = s$ at different points. As a result, the enrire edge $\hat{\beta}_1 + \hat{\beta}_2 = s$ isd a potential solution to the Lasso optimization problem!

Similar argument can be made for the opposite Lasso-diamond edge: $\hat{\beta}_1 + \hat{\beta}_2 = -s$.

Thus, the Lasso problem does not have a unique solution. The general form of solution is

$\hat{\beta}_1 + \hat{\beta}_2 = s; \hat{\beta}_1 \geq 0; \hat{\beta}_2 \geq 0;$ and $\hat{\beta}_1 + \hat{\beta}_2 = -s; \hat{\beta}_1 \leq 0; \hat{\beta}_2 \leq 0.$

Question 2

Suppose we have a data set with five predictors, $X1 = GPA$, $X2 = IQ$, $X3 = Gender$ (1 for Female and 0 for Male), $X4$ = Interaction between GPA and IQ, and $X5$ = Interaction between GPA and $Gender$. The response is starting salary after graduation (in thousands of dollars). Suppose we use least squares to fit the model, and get $\hat{\beta}_0 = 50$, $\hat{\beta}_1 = 20$, $\hat{\beta}_3 = 35$, $\hat{\beta}_4 = 0.01$, $\hat{\beta}_5$

1. Which answer is correct, and why?

a. For a fixed value of IQ and GPA, males earn more on average than females.
b. For a fixed value of IQ and GPA, females earn more on average than males.
c. For a fixed value of IQ and GPA, males earn more on average than females provided that the GPA is high enough.
d. For a fixed value of IQ and GPA, females earn more on average than males provided that the GPA is high enough.

c. For a fixed value of IQ and GPA, males earn more on average than females provided that the GPA is high enough.

The least squares regression line is :

$Y = 50 + 20GPA + 0.07IQ + 35Gender + 0.01(GPA * IQ) - 10(GPA * Gender)$

$(35 - 10GPA)Gender$

If Male = 0 is our baseline, we find that males with a GPA higher than 3.5 will earn more on average than females.

2. Predict the salary of a female with IQ of 110 and a GPA of 4.0.

$Y = 50 + 20(4) + 0.07(110) + 35(1) + 0.01(4 \times 110) - 10(4 \times 1) = 137.1$

The predicted salary would be $137,100.

3. True or false: Since the coefficient for the GPA/IQ interaction term is very small, there is very little evidence of an interaction effect. Justify your answer.

False - The size of the coefficient for the interaction term does not necessarily imply little evidence of an interatcion effect. The p value will help us determine significance of the term in the model, and the size of the coefficients of th GPA and IQ main effects will give us a relative scale of which we will see the actual effects of the interaction.

```
library(caret)
library(randomForest)
library(mice)
library(dplyr)
library(VIM)
library(varhandle)
library(factoextra)
library(ape)
library(fpc)
```

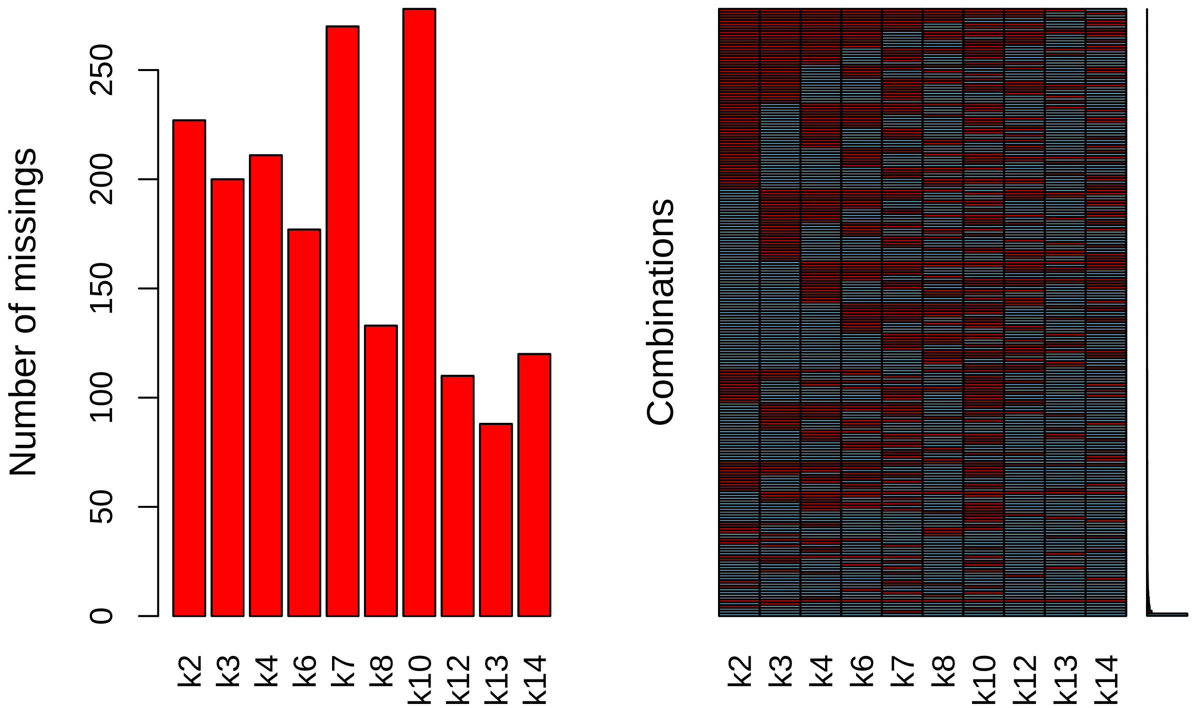
Question 3

강의 홈에 제공된 'data3.xlsx'에 대하여 다른 변수들을 이용하여 'ideo_self'를 예측하는 모형을 구축 하고자 한다. 본인이 찾은 최적의 예측모형을 서술하고 10-fold cross-validation을 이용한 시험오차(testing error)에 대해 혼동행렬(confusion matrix)을 계산하여라

```
dat = read.csv('data3.csv')
dat = dat[,-c(1,3)]
for (i in c(1,3:17))
{
  dat[,i] = as.factor(dat[,i])
}
```

변수의 성격에 맞게 str을 변환하였다.

```
aggr(dat[,7:16], prop=FALSE, numbers=TRUE)
```



결측치의 개수와 결측치의 패턴을 보기 위해 위의 함수를 사용하였다. 패턴은 없다고 판단하였다. 결측치를 모두 채운 후 ideo_self를 예측하는 모델링을 할 것이기 때문에, 결측치가 가장 적은 K13부터 가장많은 K10순서대로 값을 채울 것을 결정하였다.

```
miceresult <- mice(dat, seed = 123, m = 5)

a=complete(miceresult, 1)
b=complete(miceresult, 2)
c=complete(miceresult, 3)
d=complete(miceresult, 4)
e=complete(miceresult, 5)
dat_t=complete(miceresult, 5)


for (j in 7:(ncol(a)-1))
{
  dat_t[,j] <- unfactor(dat_t[,j])
  for (i in 1:nrow(a))
  {
    dat_t[i,j] = round(mean(unfactor(a[i,j]),unfactor(b[i,j]),unfactor(c[i,j]),
                        unfactor(d[i,j]),unfactor(e[i,j])))
  }
  dat_t[,j] <- as.factor(dat_t[,j])
}

completedData <- dat_t
```

결측치의 값을 처리하기 위해 2가지 방법을 사용하였는데, 그 중 첫번째이다. mice패키지를 사용하여 결측값을 대체하였다.

5번을 반복한 결과의 평균을 내고 반올림하는 식으로 5개의 값을 모두 이용하여 결측값을 대체하였다.

```
miceresult$method
```

```
##       sex      age1       age      area       edu    income        k2
##        ""        ""        ""        ""        ""        ""  "logreg"
##        k3        k4        k6        k7        k8       k10       k12
##  "logreg"  "logreg"  "logreg"  "logreg"  "logreg"  "logreg"  "logreg"
##       k13       k14 ideo_self
##  "logreg"  "logreg"        ""
```

각각의 변수에 대해 어떤 함수를 사용하여 결측치를 채웠는 지 확인 할 수 있다.

```
Fold_index <- createFolds(1:nrow(completedData), k = 10)

result = c()
result_acc = c()
result_flex = c()
for(k in 1:10){
  Train <- completedData[-Fold_index[[k]],]
  Test <- completedData[Fold_index[[k]],]

  out <- randomForest(ideo_self~., data = Train)
  pred <- predict(out, Test)

  result[[k]] = table(pred, Test$ideo_self)
  print(result[[k]])

  result_acc[[k]] = sum(diag(result[[k]]))/sum(result[[k]])

  pred_flex <- matrix(result[[k]], 11, 11)
  acc = rep(0, 11)

  for(i in 1:ncol(pred_flex))
  {
    if(i == 1)
    {
      acc[i] <- pred_flex[i,i] + pred_flex[i, i+1]
    }
    else if(i == 11)
    {
      acc[i] <- pred_flex[i,i] + pred_flex[i, i-1]
    }
    else
    {
      acc[i] <- pred_flex[i,i] + pred_flex[i, i-1] + pred_flex[i, i+1]
    }
  }
  result_flex[[k]] = sum(acc)/sum(pred_flex)
}
```

```
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0  0  0  0  1  0  0  0  0  0  0  0
##    1  0  0  0  0  0  0  0  0  0  0  0
##    2  0  0  0  0  0  0  0  0  0  0  0
```

```
##    3    0  0  0  2  2  4  2  0  0  0  0
##    4    0  1  1  1  2  3  1  0  0  0  0
##    5    3  2  3  6  7 24  6  6  2  1  3
##    6    0  0  0  1  0  0  3  1  0  1  1
##    7    0  0  0  0  0  1  0  3  2  0  2
##    8    0  0  0  0  1  1  0  0  2  1  1
##    9    0  0  0  0  0  0  0  0  0  0  0
##    10   0  0  0  0  0  1  0  0  0  0  1
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0    0  0  0  0  0  0  0  0  0  0  0
##    1    1  0  0  0  0  0  0  0  0  0  0
##    2    0  0  0  1  0  0  0  0  0  0  0
##    3    0  0  2  1  1  3  3  0  0  0  0
##    4    0  0  1  0  1  1  2  0  0  0  0
##    5    2  2  1  9  8 30  7  6  2  1  2
##    6    0  0  0  1  0  1  0  1  1  0  0
##    7    0  0  0  0  0  1  2  2  2  0  0
##    8    0  0  0  0  0  0  0  1  1  2  0
##    9    0  0  0  0  0  1  0  0  0  0  0
##    10   0  0  0  0  0  0  1  0  0  0  1
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0    0  0  0  0  0  0  0  0  0  0  0
##    1    0  0  0  0  0  0  1  0  0  0  0
##    2    0  0  1  0  0  0  0  0  0  0  0
##    3    0  1  0  4  1  4  0  1  2  0  0
##    4    0  0  0  2  3  0  0  0  0  0  0
##    5    1  3  4  4  7 24  5  7  1  4  3
##    6    0  0  0  0  0  3  1  1  0  1  0
##    7    0  0  0  1  0  1  1  2  3  0  0
##    8    0  0  0  0  0  0  1  2  2  1  0
##    9    0  0  0  0  0  0  0  0  0  0  0
##    10   0  0  0  0  1  2  0  0  0  0  0
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0    0  0  0  0  0  0  0  0  0  0  0
##    1    0  0  0  0  0  0  0  0  0  0  0
##    2    0  0  0  0  0  0  0  0  0  0  0
##    3    1  0  0  0  2  2  0  1  0  0  0
##    4    0  1  1  0  2  1  0  1  0  0  0
##    5    2  1  3  9  6 25  9  7  3  1  4
##    6    0  0  0  0  0  2  2  0  1  0  1
##    7    0  0  0  0  0  1  0  3  2  0  0
##    8    0  0  0  0  0  0  0  1  4  0  0
##    9    0  0  0  0  0  0  0  0  0  0  1
##    10   0  0  0  0  1  0  0  2  0  1  1
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0    0  0  0  0  0  0  0  0  0  0  0
##    1    0  0  0  0  0  1  0  0  0  0  0
##    2    0  0  0  0  0  0  0  0  0  0  0
##    3    0  0  3  4  6  6  0  0  0  0  0
##    4    0  0  1  3  0  5  0  0  0  0  0
```

```
##   5    3  3  2  4  6 19  3  7  1  0  4
##   6    0  0  0  1  1  0  0  0  0  0  1
##   7    0  0  0  0  0  1  0  3  1  0  2
##   8    0  0  0  0  0  2  2  2  0  1  3
##   9    0  0  0  0  0  1  0  0  0  0  0
##   10   0  0  0  0  0  2  1  0  0  1  0
##     
## pred  0  1  2  3  4  5  6  7  8  9 10
##   0    0  0  0  0  0  0  0  0  0  0  0
##   1    0  0  0  0  0  0  1  0  0  0  0
##   2    0  0  0  1  0  0  0  0  0  0  0
##   3    3  0  1  3  0  4  1  3  1  0  0
##   4    0  0  1  0  0  2  0  0  0  0  0
##   5    3  4  1  8  8 20  6  7  5  0  1
##   6    0  0  0  1  0  1  4  1  0  0  0
##   7    0  0  0  0  0  0  3  0  1  0  0
##   8    0  0  0  0  0  1  1  1  2  1  1
##   9    0  0  0  0  0  0  0  0  0  0  0
##   10   1  0  0  0  0  2  1  0  1  0  0
##     
## pred  0  1  2  3  4  5  6  7  8  9 10
##   0    0  0  0  0  0  0  0  0  0  0  0
##   1    0  0  1  0  1  1  0  0  0  0  0
##   2    1  0  0  0  0  0  0  0  0  0  0
##   3    0  0  2  2  1  2  0  0  0  0  0
##   4    0  1  1  2  1  1  1  0  1  0  0
##   5    2  0  3 10  9 17 10  2  2  1  6
##   6    0  0  0  1  1  1  1  1  1  0  0
##   7    0  0  0  0  0  2  2  1  1  0  1
##   8    0  0  0  0  0  1  1  2  1  2  1
##   9    0  0  0  0  0  0  0  0  0  0  1
##   10   0  0  0  0  0  0  0  0  0  0  2
##     
## pred  0  1  2  3  4  5  6  7  8  9 10
##   0    0  0  0  0  0  0  0  0  0  0  0
##   1    0  0  0  0  0  0  0  0  0  0  0
##   2    0  0  0  0  0  1  0  0  0  0  0
##   3    0  1  1  1  2  5  0  0  1  0  0
##   4    0  1  0  4  2  2  0  0  0  0  0
##   5    2  4  2  7 10 18  7  2  3  1  3
##   6    0  0  0  3  0  0  2  1  0  0  0
##   7    0  0  0  0  1  2  0  3  1  0  1
##   8    0  0  0  0  0  2  0  1  1  1  0
##   9    0  0  0  0  0  1  0  0  0  0  0
##   10   0  0  0  1  0  1  1  0  0  0  3
##     
## pred  0  1  2  3  4  5  6  7  8  9 10
##   0    0  0  0  0  0  0  0  0  0  0  0
##   1    0  0  0  0  0  0  0  0  0  0  0
##   2    0  0  0  2  0  0  0  0  0  0  0
##   3    1  0  1  2  3  3  1  1  0  0  0
##   4    1  0  2  2  0  1  1  0  0  0  0
##   5    4  1  2  3  5 26  6  4  1  0  3
##   6    0  0  0  0  2  1  2  1  0  0  0
```

```
##    7    0   0   0   1   0   2   3   2   2   0   0
##    8    0   0   0   1   0   1   1   2   4   0   1
##    9    0   0   0   0   0   0   0   0   0   0   0
##    10   0   0   0   0   0   0   1   0   0   0   1
##
## pred    0   1   2   3   4   5   6   7   8   9  10
##    0    0   0   0   0   0   0   0   0   0   0   0
##    1    0   0   0   0   0   0   0   0   0   0   0
##    2    0   0   0   0   0   0   0   0   0   0   0
##    3    2   1   3   3   1   3   2   0   0   0   1
##    4    0   0   0   1   2   0   0   0   0   0   0
##    5    4   0   2   4   9  24   5   6   5   1   0
##    6    0   0   0   1   0   0   2   0   1   0   1
##    7    0   0   0   0   0   1   2   4   1   0   0
##    8    0   0   0   0   0   0   0   0   3   1   0
##    9    0   0   0   0   0   0   0   0   1   0   0
##    10   0   0   0   0   0   1   0   1   1   1   3
```

```r
mean(result_acc)
```

```
## [1] 0.3180776
```

```r
mean(result_flex)
```

```
## [1] 0.58474
```

10-fold결과의 평균값 10-fold결과의 좀더 여유있게 정확도를 보면 (예측값의 +-1 정도의 여유치) 좀 더 높게 나오는 것을 알 수 있다.

```r
dat_k13 = dat[,c(1:6,15,length(dat))]
model = randomForest(as.factor(k13)~., data = na.omit(dat_k13))
pred = predict(model, newdata = dat_k13[is.na(dat_k13$k13),], type = 'response')
dat[is.na(dat$k13),]['k13'] = as.integer(as.character(pred))
dat$k13 = as.factor(dat$k13)

dat_k12 = dat[,c(1:6,14,15,length(dat))]
model = randomForest(as.factor(k12)~., data = na.omit(dat_k12))
pred = predict(model, newdata = dat_k12[is.na(dat_k12$k12),], type = 'response')
dat[is.na(dat$k12),]['k12'] = as.integer(as.character(pred))
dat$k12 = as.factor(dat$k12)

dat_k14 = dat[,c(1:6,14:length(dat))]
model = randomForest(as.factor(k14)~., data = na.omit(dat_k14))
pred = predict(model, newdata = dat_k14[is.na(dat_k14$k14),], type = 'response')
dat[is.na(dat$k14),]['k14'] = as.integer(as.character(pred))
dat$k14 = as.factor(dat$k14)

dat_k8 = dat[,c(1:6,12,14:length(dat))]
model = randomForest(as.factor(k8)~., data = na.omit(dat_k8))
pred = predict(model, newdata = dat_k8[is.na(dat_k8$k8),], type = 'response')
dat[is.na(dat$k8),]['k8'] = as.integer(as.character(pred))
dat$k8 = as.factor(dat$k8)

dat_k6 = dat[,c(1:6,10,12,14:length(dat))]
model = randomForest(as.factor(k6)~., data = na.omit(dat_k6))
pred = predict(model, newdata = dat_k6[is.na(dat_k6$k6),], type = 'response')
dat[is.na(dat$k6),]['k6'] = as.integer(as.character(pred))
```

```r
dat$k6 = as.factor(dat$k6)

dat_k3 = dat[,c(1:6,8,10,12,14:length(dat))]
model = randomForest(as.factor(k3)~., data = na.omit(dat_k3))
pred = predict(model, newdata = dat_k3[is.na(dat_k3$k3),], type = 'response')
dat[is.na(dat$k3),]['k3'] = as.integer(as.character(pred))
dat$k3 = as.factor(dat$k3)

dat_k4 = dat[,c(1:6,8:10,12,14:length(dat))]
model = randomForest(as.factor(k4)~., data = na.omit(dat_k4))
pred = predict(model, newdata = dat_k4[is.na(dat_k4$k4),], type = 'response')
dat[is.na(dat$k4),]['k4'] = as.integer(as.character(pred))
dat$k4 = as.factor(dat$k4)

dat_k2 = dat[,c(1:10,12,14:length(dat))]
model = randomForest(as.factor(k2)~., data = na.omit(dat_k2))
pred = predict(model, newdata = dat_k2[is.na(dat_k2$k2),], type = 'response')
dat[is.na(dat$k2),]['k2'] = as.integer(as.character(pred))
dat$k2 = as.factor(dat$k2)
dat_k7 = dat[,c(1:12,14:length(dat))]

model = randomForest(as.factor(k7)~., data = na.omit(dat_k7))
pred = predict(model, newdata = dat_k7[is.na(dat_k7$k7),], type = 'response')
dat[is.na(dat$k7),]['k7'] = as.integer(as.character(pred))
dat$k7 = as.factor(dat$k7)
dat_k10 = dat[,c(1:length(dat))]

model = randomForest(as.factor(k10)~., data = na.omit(dat_k10))
pred = predict(model, newdata = dat_k10[is.na(dat_k10$k10),], type = 'response')
dat[is.na(dat$k10),]['k10'] = as.integer(as.character(pred))
dat$k10 = as.factor(dat$k10)
```

각각의 결측치를 가장적은 k13부터 가장많은 k10을 randomforest를 사용하여 값을 할당한다.

```r
Fold_index <- createFolds(1:nrow(dat), k = 10)

result = c()
result_acc = c()
result_flex = c()
for(k in 1:10){
  Train <- dat[-Fold_index[[k]],]
  Test <- dat[Fold_index[[k]],]

  out <- randomForest(ideo_self~., data = Train)
  pred <- predict(out, Test)

  result[[k]] = table(pred, Test$ideo_self)
  print(result[[k]])

  result_acc[[k]] = sum(diag(result[[k]]))/sum(result[[k]])

  pred_flex <- matrix(result[[k]], 11, 11)

  acc = rep(0, 11)
```

```r
  for(i in 1:ncol(pred_flex))
  {
    if(i == 1)
    {
      acc[i] <- pred_flex[i,i] + pred_flex[i, i+1]
    }
    else if(i == 11)
    {
      acc[i] <- pred_flex[i,i] + pred_flex[i, i-1]
    }
    else
    {
      acc[i] <- pred_flex[i,i] + pred_flex[i, i-1] + pred_flex[i, i+1]
    }
  }
  result_flex[[k]] = sum(acc)/sum(pred_flex)
}
```

```
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0  0  0  0  0  0  0  0  0  0  0  0
##    1  0  0  0  1  0  0  0  0  0  0  0
##    2  0  0  0  0  1  1  0  0  0  0  0
##    3  0  0  1  3  1  3  0  1  0  0  0
##    4  0  0  0  2  1  1  2  0  1  0  0
##    5  2  4  1  5  7 26  6  6  0  1  5
##    6  0  0  0  0  0  0  1  4  1  1  0
##    7  0  0  0  0  1  1  2  3  1  0  0
##    8  0  0  0  0  0  0  1  1  3  0  0
##    9  0  0  0  0  0  0  0  0  0  0  0
##    10 0  0  0  0  0  0  0  0  2  0  2
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0  0  0  0  1  0  0  0  0  0  0  0
##    1  1  0  0  1  1  0  0  0  0  0  0
##    2  0  0  0  2  0  0  0  0  0  0  0
##    3  0  1  2  1  1  4  1  1  1  0  0
##    4  0  0  1  0  3  0  0  2  0  0  0
##    5  5  0  1 10  3 26  4  3  2  2  7
##    6  0  0  0  0  2  1  1  1  1  0  0
##    7  0  0  0  1  0  0  1  3  1  0  1
##    8  0  0  0  1  0  0  0  0  1  0  0
##    9  0  0  0  0  0  2  0  0  0  0  0
##    10 0  0  0  0  1  1  1  0  0  0  0
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0  0  0  0  0  0  0  0  0  0  0  0
##    1  0  0  0  0  0  0  0  0  0  0  0
##    2  1  0  0  1  0  0  0  0  0  0  0
##    3  0  0  1  4  0  1  1  1  1  0  0
##    4  0  0  2  3  0  1  0  1  0  0  0
##    5  3  1  1  4 10 19 13  4  3  1  1
##    6  0  0  0  0  0  1  0  1  0  0  0
```

```
##     7   0  0  0  0  2  0  3  0  2  0  3
##     8   0  0  0  0  0  3  0  1  2  2  0
##     9   0  0  0  0  0  0  0  0  0  1  0
##     10  0  0  0  0  0  1  2  1  0  0  1
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##     0   0  0  0  0  0  1  0  0  0  0  0
##     1   0  0  0  0  0  0  0  0  0  0  0
##     2   0  0  1  0  0  0  0  0  0  0  0
##     3   1  2  1  2  4  2  0  0  0  0  0
##     4   2  0  1  3  1  2  2  0  0  0  0
##     5   3  3  5  4  5 16  5  3  2  1  1
##     6   0  0  0  1  0  2  3  0  0  0  1
##     7   0  0  0  0  1  1  2  5  0  1  0
##     8   0  0  0  0  0  0  2  2  1  0  1
##     9   0  0  0  0  0  0  0  0  1  0  2
##     10  0  0  0  0  2  1  1  2  1  0  0
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##     0   0  0  0  0  0  0  0  0  0  0  0
##     1   0  0  0  0  0  0  0  0  0  0  0
##     2   0  0  0  1  1  0  0  0  0  0  0
##     3   0  0  0  4  3  3  0  1  1  0  0
##     4   0  1  3  1  2  1  0  0  0  0  0
##     5   2  1  1  7  4 17  4  6  4  1  1
##     6   0  0  0  0  0  7  1  4  2  1  0
##     7   1  0  0  0  1  3  2  1  0  1  1
##     8   0  0  0  0  0  0  0  1  1  1  1
##     9   0  0  0  0  0  0  0  0  0  0  0
##     10  0  0  0  0  0  1  0  2  1  0  2
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##     0   0  0  0  0  0  0  0  0  0  0  0
##     1   0  0  0  0  0  0  0  0  0  0  0
##     2   0  0  0  0  1  0  0  0  0  0  0
##     3   0  0  2  7  2  3  2  1  1  0  1
##     4   0  0  0  2  1  2  0  0  0  0  1
##     5   2  1  5  7  6 20  7  7  2  0  1
##     6   0  0  0  0  1  1  4  0  0  0  0
##     7   0  0  0  2  0  1  3  1  1  0  1
##     8   0  0  0  0  0  1  0  0  2  1  0
##     9   0  0  0  0  0  0  0  0  0  0  0
##     10  0  0  0  0  0  0  0  0  2  0  2
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##     0   0  0  0  0  1  0  0  0  0  0  0
##     1   0  0  0  0  0  0  0  0  0  0  0
##     2   0  0  0  0  0  0  0  0  0  0  0
##     3   0  0  1  0  1  6  2  1  1  0  0
##     4   1  0  0  0  3  3  1  0  0  0  0
##     5   2  3  1  4  6 30  4  8  3  0  3
##     6   0  0  0  0  0  1  3  1  0  0  0
##     7   0  0  0  0  0  1  2  0  2  0  1
##     8   0  0  0  0  0  0  0  0  2  1  1
```

```
##    9   0  0  0  0  1  1  0  0  1  0  0
##    10  0  0  0  0  0  1  0  0  0  0  0
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0   0  0  0  0  1  0  0  0  0  0  0
##    1   0  0  0  0  0  0  0  0  0  0  0
##    2   1  1  1  1  1  0  1  0  0  0  0
##    3   1  1  0  2  3  7  0  1  0  0  0
##    4   0  2  1  2  2  1  0  0  0  0  0
##    5   2  3  1  5  3 21  2  2  2  1  5
##    6   0  0  2  0  1  3  0  1  0  0  0
##    7   0  0  0  0  0  0  1  4  1  0  2
##    8   0  0  0  0  0  1  0  1  4  0  1
##    9   0  0  0  0  0  1  0  0  0  1  0
##    10  0  0  0  0  0  1  1  0  1  1  1
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0   0  0  0  0  0  0  0  0  0  0  0
##    1   0  0  0  0  0  0  0  0  0  0  0
##    2   0  1  0  0  0  0  0  0  0  0  0
##    3   0  0  4  1  2  5  2  0  0  0  0
##    4   1  0  2  0  4  1  0  0  0  0  0
##    5   0  0  1  8  5 27  7  5  2  0  2
##    6   0  0  0  1  1  1  3  2  2  0  0
##    7   0  0  0  0  0  1  0  1  1  0  1
##    8   0  0  0  0  0  1  0  1  2  0  1
##    9   0  0  0  0  0  0  0  0  0  0  0
##    10  0  0  0  0  0  0  1  0  0  0  4
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0   0  0  0  0  0  0  0  0  0  0  0
##    1   0  0  0  0  0  0  0  0  0  0  0
##    2   0  0  0  1  0  0  0  0  0  0  0
##    3   0  0  0  3  1  2  0  0  0  0  0
##    4   0  1  2  2  4  2  1  0  1  0  0
##    5   4  1  1  7  7 25  8  3  1  2  0
##    6   1  0  0  1  0  2  3  2  1  1  2
##    7   0  0  0  0  0  1  1  3  0  0  0
##    8   0  0  0  0  0  0  1  0  1  2  0
##    9   0  0  0  0  0  0  0  0  0  0  0
##    10  1  0  0  0  0  0  0  0  1  1  1
```

```
mean(result_acc)
```

```
## [1] 0.3330494
```

```
mean(result_flex)
```

```
## [1] 0.584761
```

10-fold결과의 평균값 10-fold결과의 좀더 여유있게 정확도를 보면 (예측값의 +-1 정도의 여유치) 좀 더 높게 나오는 것을 알 수 있다.

결과론적으로 mice함수를 사용한 결과와 랜덤포레스트를 사용하여 직접적으로 변수를 채운 결과는 랜덤포레스트를 사용한 것이 더 높은것을 알 수 있다.

```
Fold_index <- createFolds(1:nrow(dat), k = 10)
```

```r
result = c()
result_acc = c()
result_flex = c()
for(k in 1:10){
  Train <- dat[-Fold_index[[k]],]
  Test <- dat[Fold_index[[k]],]

  out <- nnet::multinom(ideo_self~., data = Train[,7:17])
  pred <- predict(out, Test)

  result[[k]] = table(pred, Test$ideo_self)
  print(result[[k]])

  result_acc[[k]] = sum(diag(result[[k]]))/sum(result[[k]])

  pred_flex <- matrix(result[[k]], 11, 11)

  acc = rep(0, 11)

  for(i in 1:ncol(pred_flex))
  {
    if(i == 1)
    {
      acc[i] <- pred_flex[i,i] + pred_flex[i, i+1]
    }
    else if(i == 11)
    {
      acc[i] <- pred_flex[i,i] + pred_flex[i, i-1]
    }
    else
    {
      acc[i] <- pred_flex[i,i] + pred_flex[i, i-1] + pred_flex[i, i+1]
    }
  }
  result_flex[[k]] = sum(acc)/sum(pred_flex)
}
```

```
## # weights:  132 (110 variable)
## initial  value 2275.602614
## iter  10 value 1803.357474
## iter  20 value 1727.198704
## iter  30 value 1717.891559
## iter  40 value 1715.995572
## iter  50 value 1715.651171
## iter  60 value 1715.489704
## iter  70 value 1715.429332
## iter  80 value 1715.425082
## iter  90 value 1715.423924
## final  value 1715.423679
## converged
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0  0  0  0  0  0  0  0  0  0  0  0
##    1  0  0  0  0  0  0  0  0  0  0  0
```

```
##   2   0  0  0  0  0  0  0  0  0  0  0
##   3   0  1  3  4  2  4  1  2  1  0  0
##   4   0  0  0  2  0  0  0  0  0  0  0
##   5   4  3  4  6  6 16  7  7  0  2  3
##   6   0  0  0  1  1  0  0  1  0  0  0
##   7   0  0  0  0  2  1  3  2  0  1  1
##   8   0  0  0  0  1  1  1  3  3  2  1
##   9   0  0  0  0  0  0  1  0  0  0  0
##   10  0  0  0  0  0  0  0  0  1  0  0
## # weights:  132 (110 variable)
## initial  value 2275.602614
## iter  10 value 1842.067318
## iter  20 value 1753.011528
## iter  30 value 1742.901743
## iter  40 value 1741.346332
## iter  50 value 1740.886695
## iter  60 value 1740.766888
## iter  70 value 1740.703426
## iter  80 value 1740.682007
## iter  90 value 1740.679465
## final  value 1740.679075
## converged
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##   0   0  0  0  0  0  0  0  0  0  0  0
##   1   0  0  0  0  0  0  0  0  0  0  0
##   2   0  0  0  0  0  0  0  0  0  0  0
##   3   1  1  3  5  3  2  0  1  0  0  0
##   4   0  0  0  0  0  0  0  0  0  0  0
##   5   4  1  0  7  5 28  9  4  1  2  5
##   6   0  0  0  1  1  3  0  0  0  0  0
##   7   0  0  0  0  1  0  0  2  4  0  0
##   8   0  0  0  0  0  1  0  3  0  2  2
##   9   0  0  0  0  0  0  0  0  0  0  0
##   10  0  0  0  0  0  1  0  0  1  0  1
## # weights:  132 (110 variable)
## initial  value 2273.204719
## iter  10 value 1805.277079
## iter  20 value 1742.122413
## iter  30 value 1734.193040
## iter  40 value 1731.897393
## iter  50 value 1731.148317
## iter  60 value 1730.768859
## iter  70 value 1730.680554
## iter  80 value 1730.670019
## final  value 1730.668897
## converged
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##   0   0  0  0  0  0  0  0  0  0  0  0
##   1   0  0  0  0  0  0  0  0  0  0  0
##   2   0  0  0  0  0  0  0  0  0  0  0
##   3   0  0  0  5  3  1  2  0  1  0  0
##   4   0  0  0  0  0  0  0  0  0  0  0
```

```
##    5   4  1  3  3  6 26 13  7  2  3  4
##    6   0  0  0  0  2  2  0  1  0  0  1
##    7   0  0  0  0  1  1  0  1  0  0  0
##    8   0  0  0  0  0  1  2  3  5  0  1
##    9   0  0  0  0  0  0  0  0  0  0  0
##    10  0  0  0  0  0  0  1  0  0  0  0
## # weights:  132 (110 variable)
## initial  value 2275.602614
## iter  10 value 1821.832094
## iter  20 value 1741.423305
## iter  30 value 1731.627507
## iter  40 value 1729.658922
## iter  50 value 1729.239755
## iter  60 value 1729.068826
## iter  70 value 1729.023280
## iter  80 value 1729.009387
## iter  90 value 1729.007195
## final  value 1729.006779
## converged
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0   0  0  0  0  0  0  0  0  0  0  0
##    1   0  0  0  0  0  0  0  0  0  0  0
##    2   0  0  0  0  0  0  0  0  0  0  0
##    3   2  1  2  4  2  5  0  1  1  0  2
##    4   0  0  0  2  0  3  2  0  0  0  0
##    5   3  1  0  4  4 21  5  6  7  1  3
##    6   0  0  0  0  0  2  1  0  0  0  0
##    7   0  0  0  0  0  1  1  4  4  0  0
##    8   0  0  0  0  0  1  2  0  2  0  1
##    9   0  0  0  0  0  0  0  0  0  0  0
##    10  0  0  0  0  0  2  0  0  0  0  2
## # weights:  132 (110 variable)
## initial  value 2273.204719
## iter  10 value 1822.208433
## iter  20 value 1748.708765
## iter  30 value 1741.139473
## iter  40 value 1739.124134
## iter  50 value 1738.776535
## iter  60 value 1738.581407
## iter  70 value 1738.534963
## iter  80 value 1738.526903
## iter  90 value 1738.525934
## final  value 1738.525715
## converged
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0   0  0  0  0  0  0  0  0  0  0  0
##    1   0  0  0  0  0  0  0  0  0  0  0
##    2   0  1  0  0  0  0  0  0  0  0  0
##    3   0  0  0  3  0  0  0  0  0  0  0
##    4   0  0  2  3  1  1  0  0  1  0  0
##    5   3  2  5  4 10 30  7  6  1  0  4
##    6   0  0  0  1  0  1  0  1  0  0  0
```

```
##    7   0  0  0  0  0  1  0  2  0  0  0
##    8   0  0  0  0  1  1  4  3  3  2  0
##    9   0  0  0  0  0  0  0  0  0  0  0
##    10  0  0  0  0  0  0  1  0  1  0  0
## # weights:  132 (110 variable)
## initial  value 2273.204719
## iter  10 value 1827.938984
## iter  20 value 1741.550036
## iter  30 value 1731.738556
## iter  40 value 1729.973977
## iter  50 value 1729.655194
## iter  60 value 1729.499103
## iter  70 value 1729.451217
## iter  80 value 1729.445486
## iter  90 value 1729.444567
## final  value 1729.444431
## converged
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0   0  0  0  0  0  0  0  0  0  0  0
##    1   0  0  0  0  0  0  0  0  0  0  0
##    2   0  0  0  0  0  0  0  0  0  0  0
##    3   1  0  0  4  1  2  0  0  1  0  0
##    4   0  0  1  2  0  1  0  0  0  0  0
##    5   3  2  1  8 14 24  4  6  2  0  4
##    6   0  0  0  0  0  0  0  2  1  0  0
##    7   0  0  0  0  0  2  0  1  1  0  0
##    8   0  0  0  1  0  1  2  3  3  3  1
##    9   0  0  0  0  0  0  0  0  0  0  0
##    10  0  0  1  0  0  1  0  1  0  0  1
## # weights:  132 (110 variable)
## initial  value 2278.000509
## iter  10 value 1841.647889
## iter  20 value 1766.806727
## iter  30 value 1757.801017
## iter  40 value 1756.374855
## iter  50 value 1756.020180
## iter  60 value 1755.870933
## iter  70 value 1755.814422
## iter  80 value 1755.810664
## iter  90 value 1755.809556
## final  value 1755.809412
## converged
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0   0  0  0  0  0  0  0  0  0  0  0
##    1   0  0  0  0  0  0  0  0  0  0  0
##    2   0  0  0  1  0  0  0  0  0  0  0
##    3   1  0  0  4  5  1  0  0  0  0  0
##    4   0  0  0  0  0  0  0  0  0  0  0
##    5   1  2  1  4 11 29  6  5  4  0  2
##    6   1  0  0  0  0  3  2  0  0  0  0
##    7   0  0  0  0  0  3  0  4  0  0  1
##    8   0  0  0  0  0  4  2  1  1  2  2
```

```
##    9   0  0  0  0  0  0  0  0  0  0  0
##    10  0  0  0  0  1  0  0  0  0  0  0
## # weights:  132 (110 variable)
## initial  value 2273.204719
## iter  10 value 1828.740514
## iter  20 value 1747.904328
## iter  30 value 1738.465804
## iter  40 value 1736.763668
## iter  50 value 1736.410225
## iter  60 value 1736.265750
## iter  70 value 1736.207759
## iter  80 value 1736.197172
## iter  90 value 1736.193598
## iter 100 value 1736.192865
## final  value 1736.192865
## stopped after 100 iterations
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0   0  0  0  0  0  0  0  0  0  0  0
##    1   0  0  0  0  0  0  0  0  0  0  0
##    2   0  0  0  0  0  0  0  0  0  0  0
##    3   2  0  5  5  2  5  1  1  0  0  0
##    4   0  0  0  1  1  0  0  0  0  0  0
##    5   0  2  4  6  6 30  7  3  4  0  2
##    6   1  0  0  0  0  2  0  2  1  2  1
##    7   1  0  0  0  0  0  0  3  0  0  0
##    8   0  0  0  0  0  0  2  0  2  0  0
##    9   0  0  0  0  0  0  0  0  0  0  0
##    10  0  0  0  0  1  0  0  0  0  0  1
## # weights:  132 (110 variable)
## initial  value 2278.000509
## iter  10 value 1828.729419
## iter  20 value 1757.426467
## iter  30 value 1747.900989
## iter  40 value 1746.037702
## iter  50 value 1745.743042
## iter  60 value 1745.598814
## iter  70 value 1745.543713
## iter  80 value 1745.539794
## iter  90 value 1745.538727
## final  value 1745.538645
## converged
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##    0   0  0  0  0  0  0  0  0  0  0  0
##    1   0  0  0  0  0  0  0  0  0  0  0
##    2   0  0  0  0  0  0  0  0  0  0  0
##    3   0  2  2  4  2  0  1  0  0  0  0
##    4   0  0  1  1  1  1  1  0  0  0  0
##    5   1  3  2  8  5 24 10  3  2  0  3
##    6   0  0  0  0  0  1  0  1  1  0  0
##    7   0  0  0  0  0  1  1  0  0  0  0
##    8   0  0  0  0  0  3  1  4  5  1  6
##    9   0  0  0  0  0  0  0  0  0  0  0
```

```
##   10  0  0  0  0  0  0  1  0  0  0  1
## # weights:  132 (110 variable)
## initial  value 2270.806823
## iter  10 value 1797.770532
## iter  20 value 1729.926496
## iter  30 value 1719.489217
## iter  40 value 1718.071344
## iter  50 value 1717.597903
## iter  60 value 1717.426644
## iter  70 value 1717.361400
## iter  80 value 1717.355943
## iter  90 value 1717.354421
## final  value 1717.354347
## converged
##
## pred  0  1  2  3  4  5  6  7  8  9 10
##   0   0  0  0  0  0  0  0  0  0  0  0
##   1   0  0  0  0  0  0  0  0  0  0  0
##   2   0  1  0  0  0  0  0  0  0  0  0
##   3   2  2  1  3  6  4  0  0  1  0  0
##   4   0  0  0  0  1  0  1  0  0  0  0
##   5   1  1  4 11  7 21 10  4  2  0  2
##   6   0  0  0  0  0  0  1  0  0  0  0
##   7   0  0  1  1  0  0  2  0  0  0  1
##   8   1  0  0  0  0  2  2  2  3  2  2
##   9   0  0  0  0  0  0  0  0  0  0  0
##   10  0  0  0  0  0  1  1  0  0  0  0
```

```r
mean(result_acc)
```

```
## [1] 0.3321472
```

```r
mean(result_flex)
```

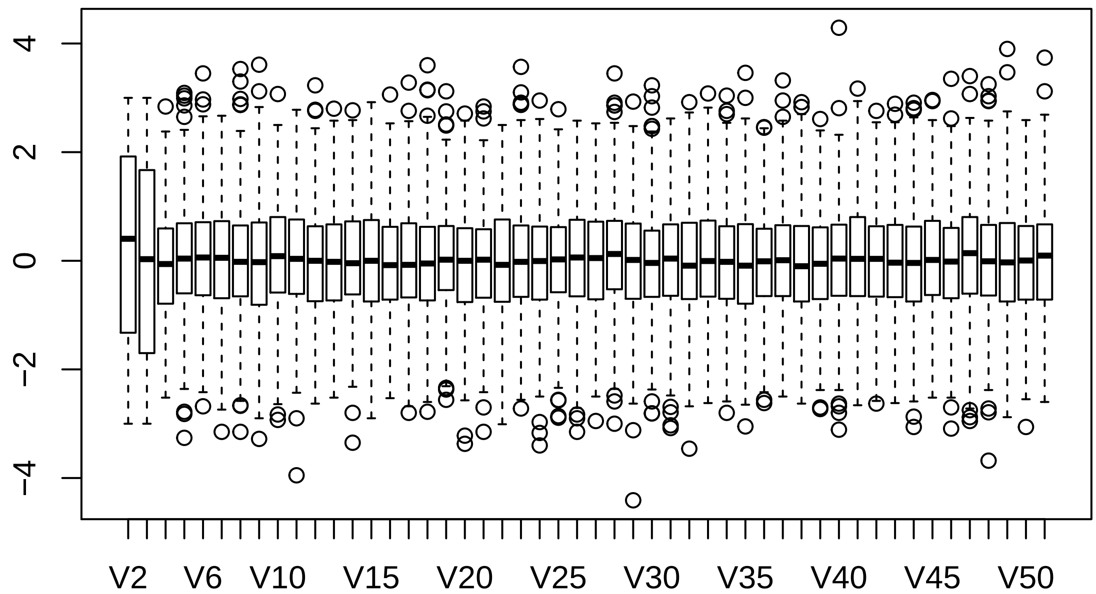
```
## [1] 0.6053969
```

다중클래스로지틱함수와 변수선택을 한 후 모델링을 하였다. 60%가 넘는 (유연한) 결과가 나왔다.

Question 4

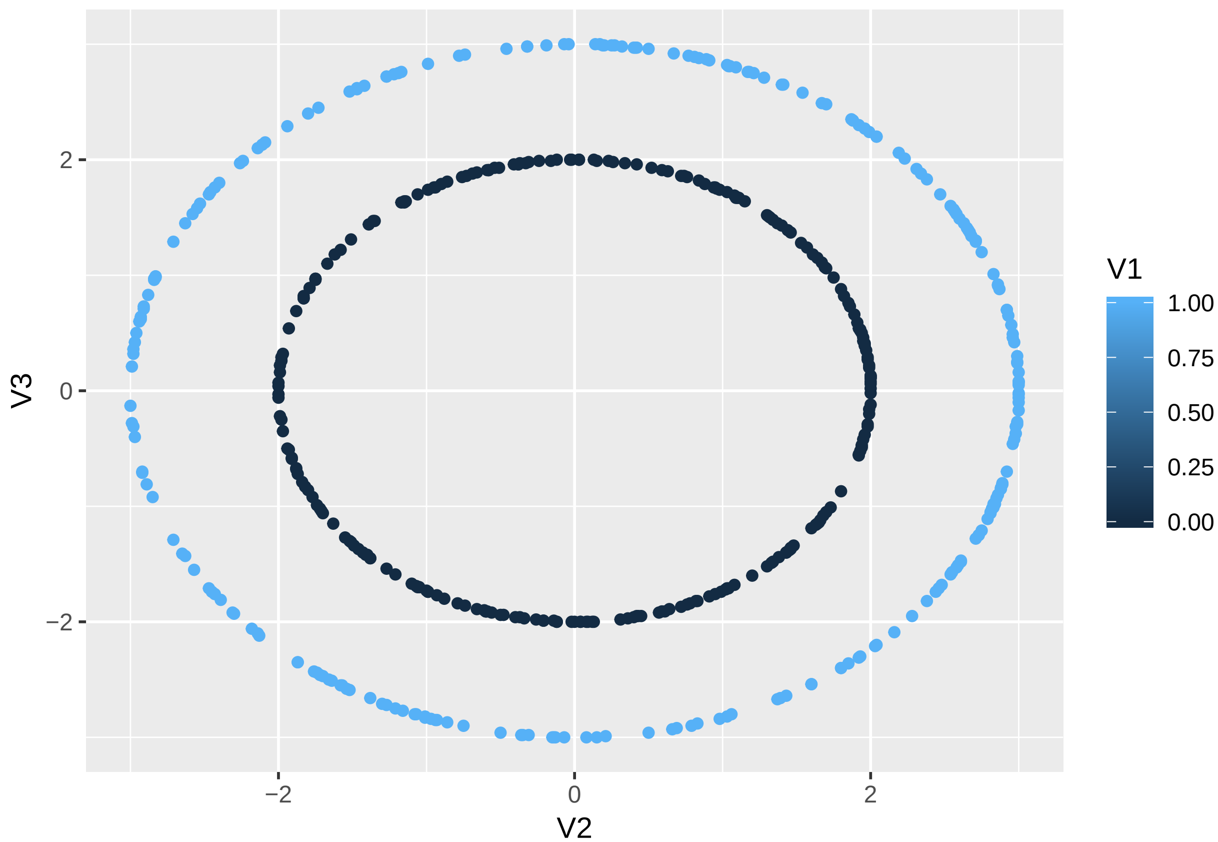
수행한 군집분석을 설명하고 결과를 적어라.

```r
boxplot(dat[,2:51])
```



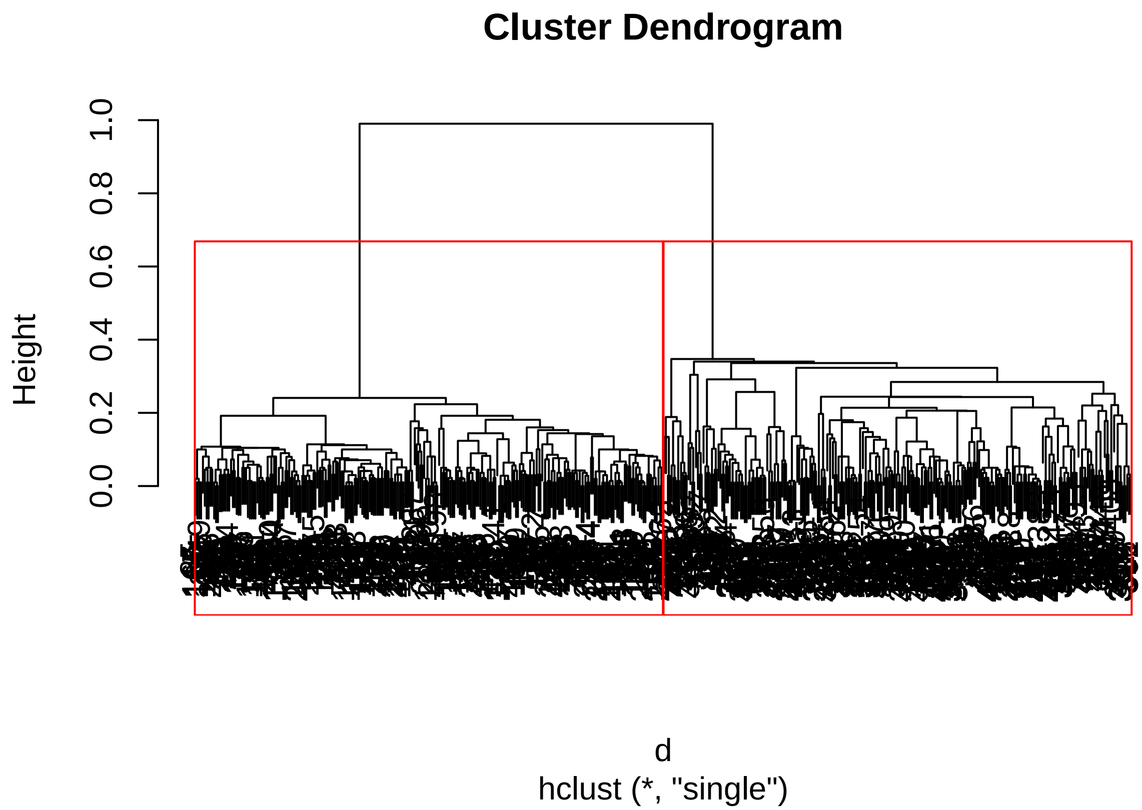
이상치가 없는 V2와 V3를 가지고 군집분석을 수행할 것이다.

```
ggplot(dat, aes(V2, V3, color = V1)) + geom_point()
```



V2와 V3가 완벽하게 V1을 구분하는 것을 알 수 있다.

```
dat1 = dat[,c(2:3)]
d <- dist(dat1, method="euclidean")
fit <- hclust(d, method="single")
plot(fit)
rect.hclust(fit, k=2, border = 'red')
```



계층적 군집을 사용하였고, 거리행렬은 euclidean거리를 사용하였으며 군집간의 거리계산은 최단연결법을 사용하였다. 그 결과 위의 군집이 형성되었다.

```
confusionMatrix(as.factor(cutree(fit, k=2) - 1), as.factor(dat[,1]))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 250   0
##          1   0 250
##
##                Accuracy : 1
##                  95% CI : (0.9926, 1)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##              Prevalence : 0.5
##          Detection Rate : 0.5
##    Detection Prevalence : 0.5
```

```
##         Balanced Accuracy : 1.0
##
##          'Positive' Class : 0
##
```

정확도의 경우 100%가 되었다.

binary classification문제를 고려하여 최적의 classifier를 찾고 10-fold cv을 이용한 시험오차에 대한 혼동행렬을 계산하여라

```
result = list()
result_acc = 0
Fold_index <- createFolds(1:nrow(dat), k = 10)

for(k in 1:10){
  Train <- dat[-Fold_index[[k]],]
  Test <- dat[Fold_index[[k]],]

  out <- randomForest(V1~V2+V3, data = Train)
  pred <- round(predict(out, Test))

  result[[k]] = table(Test$V1, pred)
  result_acc[k] = sum(diag(result[[k]]))/sum(result[[k]])
}
```

위의 비지도 학습에서 V2, V3로 100%의 정확도가 나왔기 때문에 두 변수를 사용하였다. 10-fold cv를 실행하였고, 분류기는 randomForest를 사용하였다.

```
for ( i in 1:10)
{
  print(result[[i]])
}
```

```
##    pred
##      0  1
##   0 26  0
##   1  0 24
##    pred
##      0  1
##   0 24  0
##   1  0 25
##    pred
##      0  1
##   0 24  0
##   1  0 26
##    pred
##      0  1
##   0 26  0
##   1  0 25
##    pred
##      0  1
##   0 26  0
##   1  0 25
##    pred
```

```
##       0  1
##    0 24  0
##    1  0 25
##     pred
##       0  1
##    0 24  0
##    1  0 25
##     pred
##       0  1
##    0 26  0
##    1  0 25
##     pred
##       0  1
##    0 24  0
##    1  0 25
##     pred
##       0  1
##    0 26  0
##    1  0 25
```

```
mean(result_acc)
```

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

혼동행렬과 정확도가 모두 100%가 나왔다.

Question 5

```
dat = read.csv('data5.csv')
rownames(dat) = dat[,1]
dat1 = dat[,-c(1,length(dat))]
dat1 = 1455 - dat1
dat1 = scale(dat1)
```

위의 과정은 전처리 과정인데, rownames를 설정해주고, 불필요한 컬럼은 제거하였으며, 거리가 가까움과 멀음이 반대로 되어있으므로 max값에서 빼줌으로써 반대로 바꿔주었다. 그 후 분산을 줄여주기 위하여 scaling을 하였다.

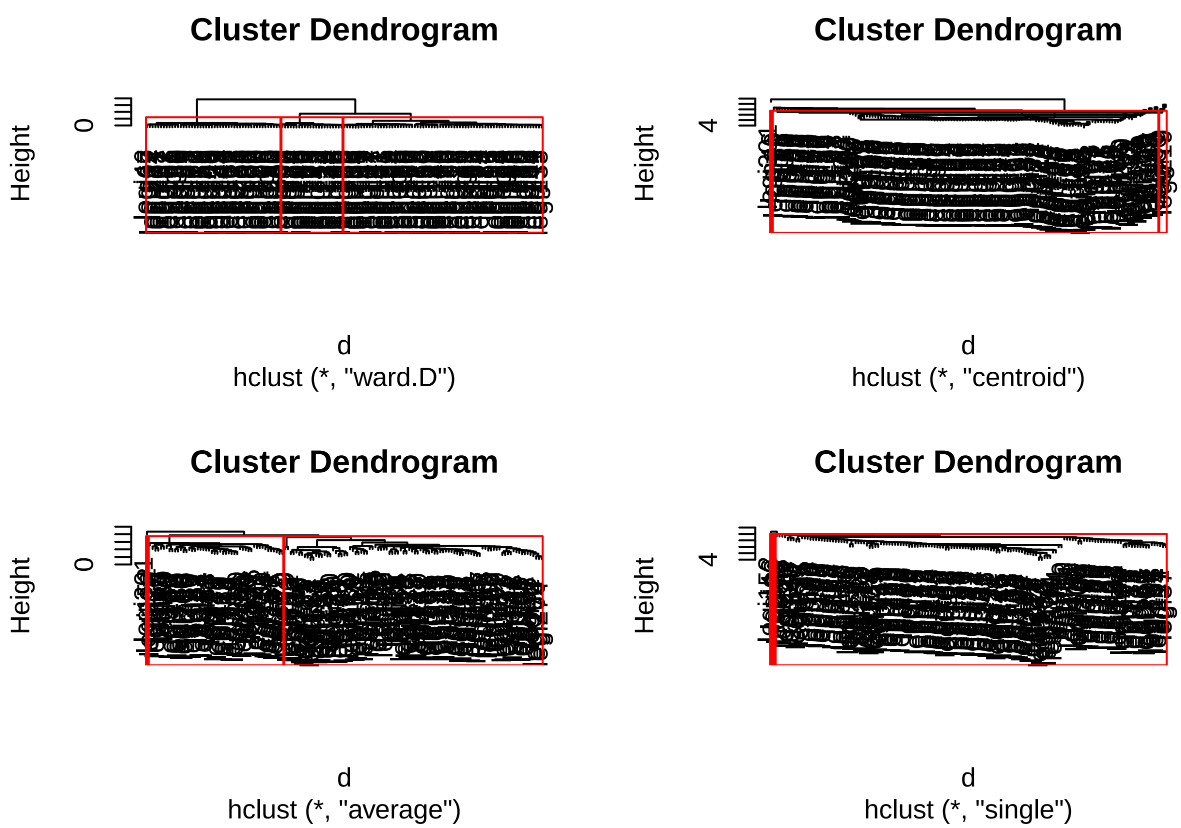
```
knitr::kable(table(dat$party), caption = '정당 별 국회의원의 수')
```

Table 1: 정당 별 국회의원의 수

Var1	Freq
국민의당	24
더불어민주당	67
무소속	1
바른정당	7
자유한국당	41
정의당	1

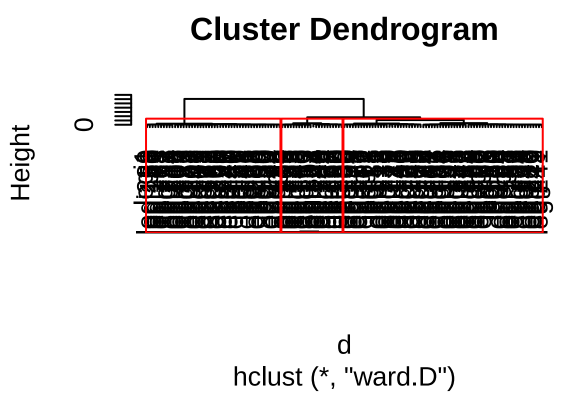
데이터가 주로 3개의 정당이 주를 이루는 걸 알 수 있다.

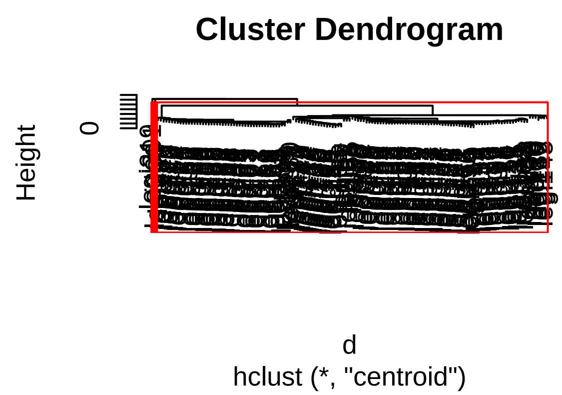
```r
par(mfrow = c(2,2))
d <- dist(dat1, method="euclidean")
fit <- hclust(d, method="ward.D")
plot(fit)
rect.hclust(fit, k = 3, border = 'red')

d <- dist(dat1, method="euclidean")
fit <- hclust(d, method="centroid")
plot(fit)
rect.hclust(fit, k = 3, border = 'red')

d <- dist(dat1, method="euclidean")
fit <- hclust(d, method="av")
plot(fit)
rect.hclust(fit, k = 3, border = 'red')

d <- dist(dat1, method="euclidean")
fit <- hclust(d, method="single")
plot(fit)
rect.hclust(fit, k = 3, border = 'red')
```

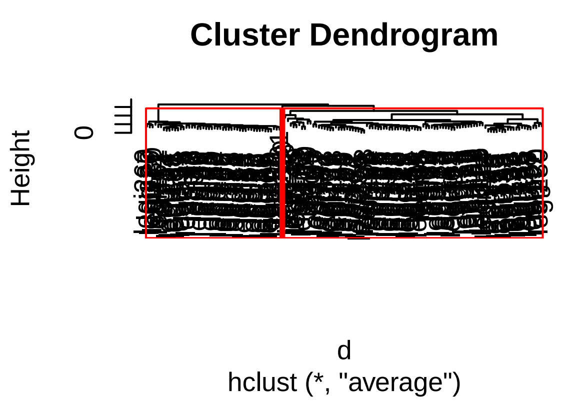


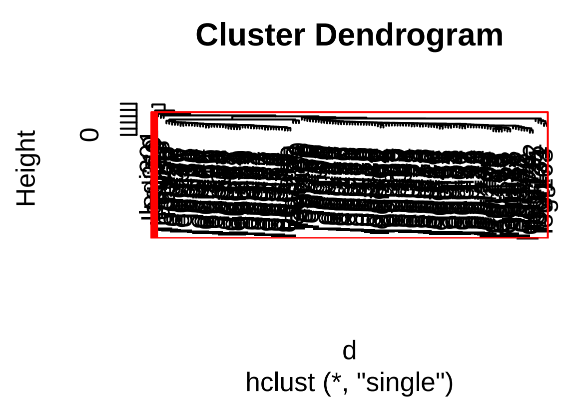

```r
par(mfrow = c(2,2))
d <- dist(dat1, method="manhattan")
fit <- hclust(d, method="ward.D")
plot(fit)
rect.hclust(fit, k = 3, border = 'red')
```

```r
d <- dist(dat1, method="manhattan")
fit <- hclust(d, method="centroid")
plot(fit)
rect.hclust(fit, k = 3, border = 'red')

d <- dist(dat1, method="manhattan")
fit <- hclust(d, method="av")
plot(fit)
rect.hclust(fit, k = 3, border = 'red')

d <- dist(dat1, method="manhattan")
fit <- hclust(d, method="single")
plot(fit)
rect.hclust(fit, k = 3, border = 'red')
```



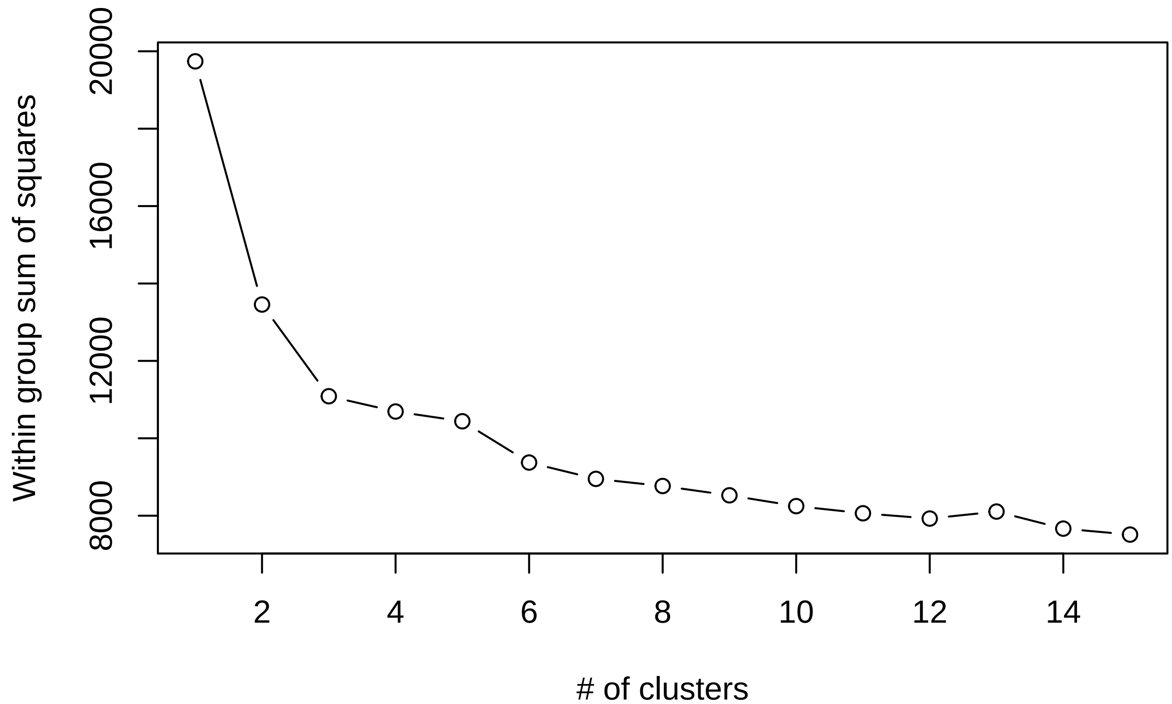






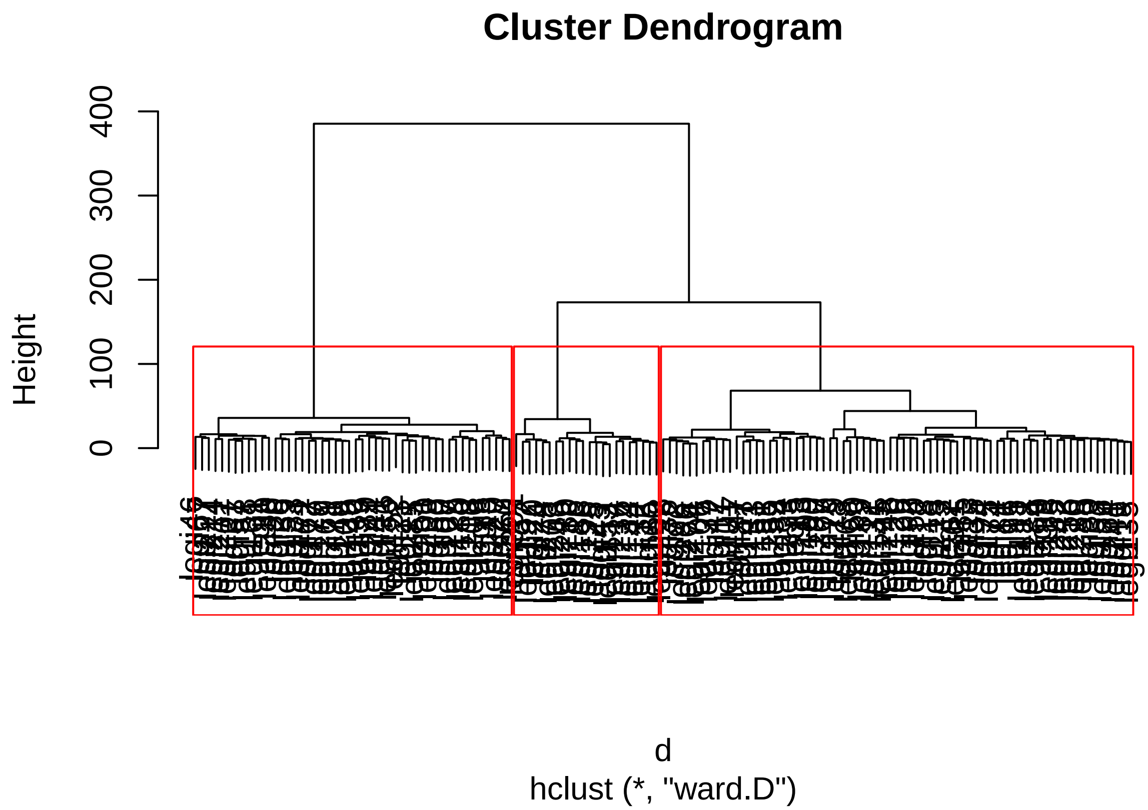
먼저 계층적 군집을 사용하였고, 거리행렬은 euclidean거리와 manhattan거리를 사용하였으며 군집간의 거리계산은 와드연결법, 중심연결법, 평균연결법, 최단연결법을 사용하였다. 그 결과 위의 8가지 군집이 형성되었다. 위의 table에서의 정답 개수와 위의 계층적군집을 보았을 때 3개의 군집이 가장 최적이라고 판단이 되었고, 3개의 군집으로 나눠보았다. 거리행렬의 거리와는 관계없이 와드연결법이 가장 좋게 나타난 것으로 보인다.

```r
wss <- 0; set.seed(1)
for(i in 1:15) wss[i] <- kmeans(dat1, centers=i)$tot.withinss
plot(1:15, wss, type="b", xlab="# of clusters", ylab="Within group sum of squares")
```



군집이 3개가 가장 좋다는 근거를 뒷받침 하기 위해 kmeans를 사용하여 최적의 군집수를 도식화한 결과 분산의 감소량이 군집이 3개일 때부터 크게 감소하는 것을 볼 수 있다.

```
d <- dist(dat1, method="euclidean")
fit <- hclust(d, method="ward.D")
plot(fit)
rect.hclust(fit, k = 3, border = 'red')
```




```r
dat2 = data.frame(cbind(rownames(dat1),cutree(fit, k=3)))
dat2 = cbind(dat2, party = dat$party)
dat2 %>% group_by(X2, party) %>% summarise(n = n())
```

```
## # A tibble: 7 x 3
## # Groups:   X2 [?]
##   X2    party            n
##   <fct> <fct>        <int>
## 1 1     바른정당         7
## 2 1     자유한국당      41
## 3 2     국민의당         2
## 4 2     더불어민주당    67
## 5 2     무소속           1
## 6 2     정의당           1
## 7 3     국민의당        22
```

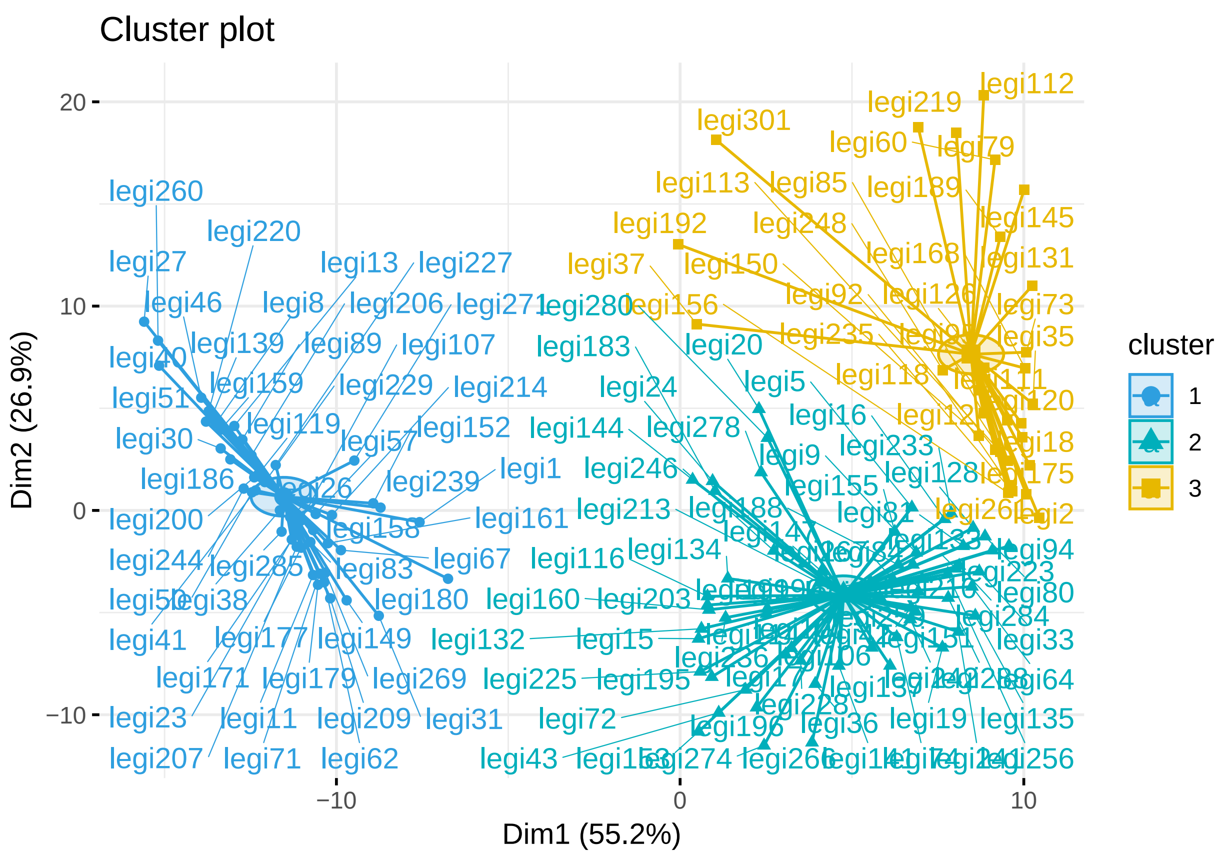
위의 맨해튼과 유클리디안 거리행렬이 따로 차이가 없으므로 유클리디안 거리행렬을 사용하였다. 계층적군집을 3개의 군집을 생성하였고 군집별로 정당이 어떻게 분포되어있는지 볼수 있는 표이다. 보수당인 바른정당과 자유한국당이 한 군집, 진보당인 국민의당 소수, 더불어민주당, 정의당이 한 군집, 국민의당이 한 군집으로 구성되어 있다. 각 정당의 특징별로 잘 군집되었다고 판단된다.

```r
km = kmeans(d, 3, nstart = 10)
```

군집의 개수를 3개로 정했으니, 군집이 3개인 kmeans를 실행한다.

```r
fviz_cluster(km, data = d,
             palette = c("#2E9FDF", "#00AFBB", "#E7B800", "#FC4E07"),
             ellipse.type = "euclid", # Concentration ellipse
```

```
              star.plot = TRUE, # Add segments from centroids to items
              repel = TRUE, # Avoid label overplotting (slow)
              ggtheme = theme_minimal()
)
```



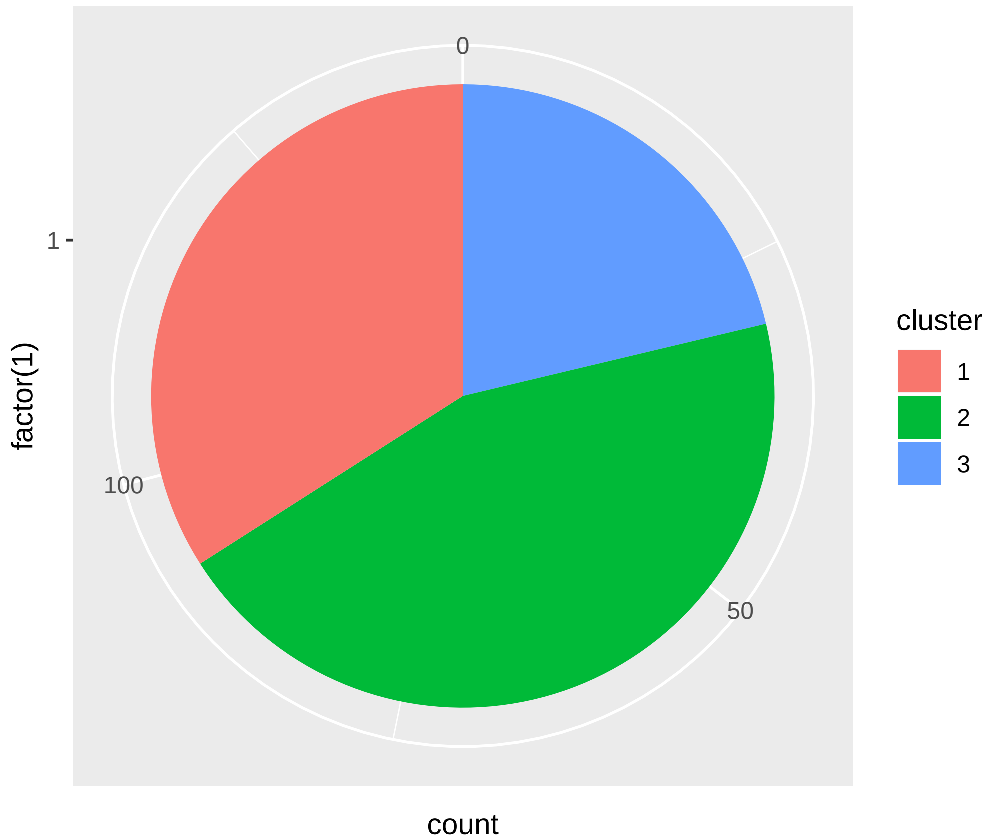
주성분 분석이 실행되어, 2개의 변수로 전체 변수의 82%를 설명하였다. 3개지 군집이 이쁘게 잘 형성된 것을 볼 수 있다. 그러나 kmeans의 단점인 가중치와 거리에 대해서 이야기를 하기 어렵다.

```
dat2 = data.frame(cbind(cluster = km$cluster, party = unfactor(dat$party)))
dat2 %>% group_by(cluster, party) %>% summarise(n = n())
```

```
## # A tibble: 8 x 3
## # Groups:   cluster [?]
##   cluster party            n
##   <fct>   <fct>        <int>
## 1 1       바른정당         7
## 2 1       자유한국당      41
## 3 2       국민의당        20
## 4 2       더불어민주당    42
## 5 2       정의당           1
## 6 3       국민의당         4
## 7 3       더불어민주당    25
## 8 3       무소속           1
```

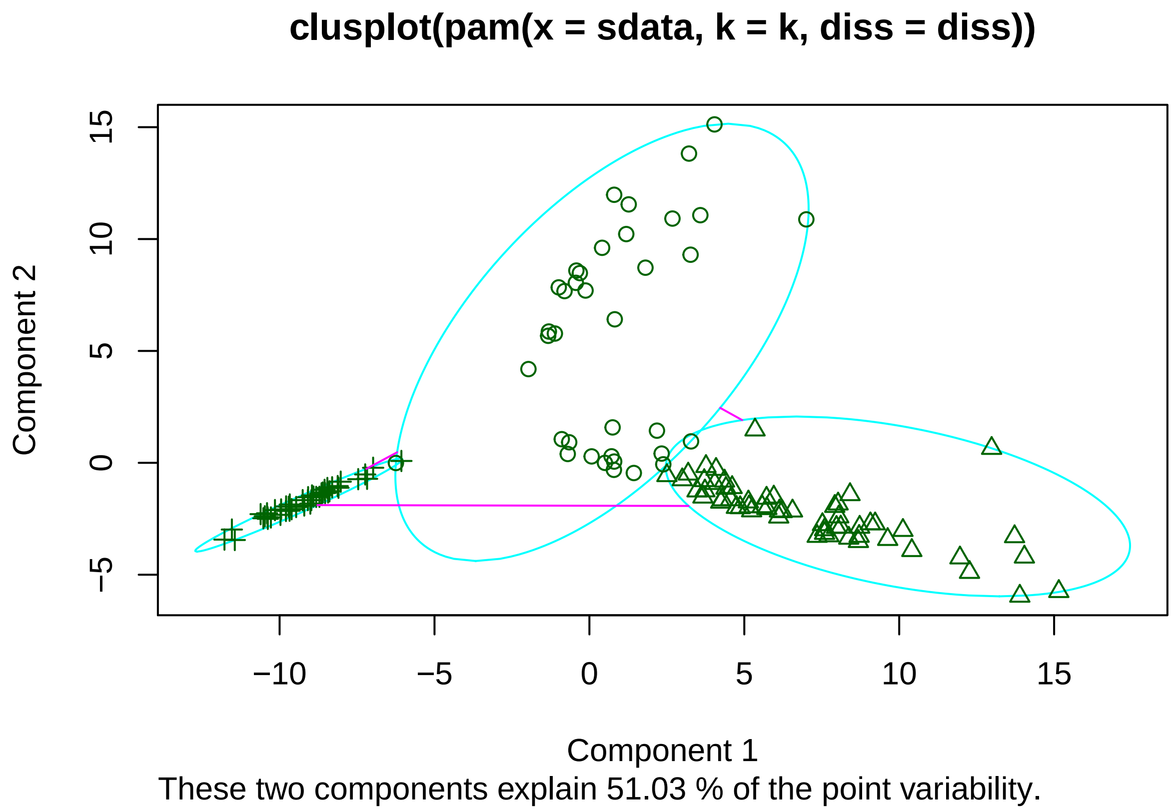
k means의 결과이다. 더불어민주당이 2군집으로 나누어 진 것을 볼 수 있다. 정당이 같아도 내부에서의 파벌이라던지, 다른 네트워크가 있다는 사실을 유추할 수 있다.

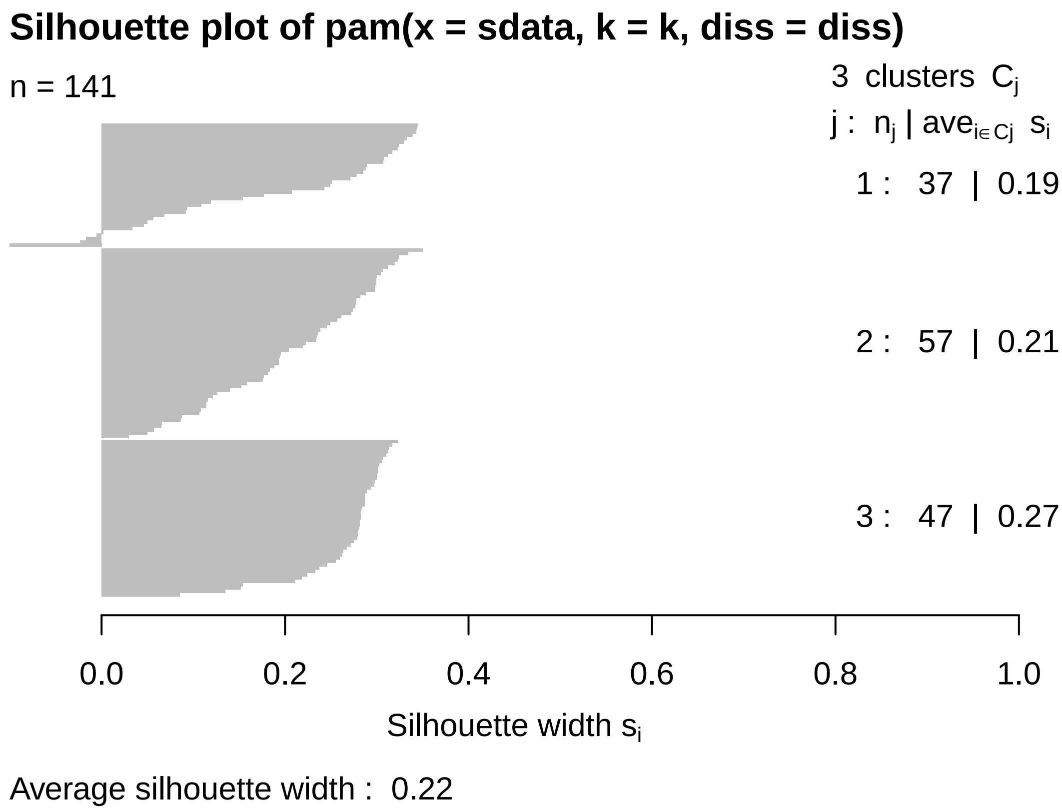
```
x <- ggplot(dat2, aes(x=factor(1), fill=cluster))
x + geom_bar(width=1) + coord_polar(theta="y")
```



k means 의 군집비율을 도식화하였다.

```
pamk.result <- pamk(dat1, 3)

plot(pamk.result$pamobject)
```


clusplot(pam(x = sdata, k = k, diss = diss))
Component 2
15
10
5
0
−5
−10
−5
0
5
10
15
Component 1
These two components explain 51.03 % of the point variability.



추가적으로 k medoid군집을 실행하였다. PAM은 K-medoid clustering이 가장 일반적으로 실현화된 것이다. 알고리즘은 k-means 알고리즘과 관련된 군집 알고리즘이고 medoidshift 알고리즘이다. k-means 와 k-medoids 알고리즘 둘다 나누는 일을 하고(즉 데이터셋을 군집들로 쪼갠다는 의미) 또한 한 군집 안에 있는 포인트들과 그 군집의 중심점 사이의 거리를 최소화 한다. k-means 알고리즘과는 다르게(한 군집의 평균을 중심점으로 잡음), k-medoids는 데이터 포인트들을 중심점(medoids 또는 exemplars라고 함)으로 선택한다.

```
dat_ta = cbind(cluster = data.frame(pamk.result$pamobject$clustering), party = dat$party)
colnames(dat_ta)[1] = 'cluster'
dat_ta %>% group_by(cluster, party) %>% summarise(n = n())
```

```
## # A tibble: 9 x 3
## # Groups:   cluster [?]
##   cluster party          n
##     <int> <fct>      <int>
## 1       1 국민의당      23
## 2       1 더불어민주당  12
## 3       1 바른정당       1
## 4       1 정의당         1
## 5       2 국민의당       1
## 6       2 더불어민주당  55
## 7       2 무소속         1
## 8       3 바른정당       6
## 9       3 자유한국당    41
```

이 방법을 사용했을 때에 신기하게 바른정당인원이 진보정당군집으로 보이는 1군집에 속한 것을 볼 수 있다. 그 인원은 legi1로 알 수 있는데, 국민의당과 나름 밀접한 관계가 있다고 추정된다.

여러가지 군집분석 방법론을 사용하면서 자유한국당과 더불어민주당은 한 군집으로 좀처럼 섞이지 않는다. 이것은 두 정당의

발의안을 좀처럼 공동발의 하지 않는다는 점을 알 수 있다. 흥미로운 점은 계층적 군집인 경우 더불어 민주당은 군집이 나누어지지 않았지만 비계층적 군집을 시행하였을 경우에 2가지의 군집으로 나누어지는 것을 볼 수 있었다. 이 점으로 보아 더불어민주당은 민주당내에서에도 2가지의 그룹으로 나뉜다는 것을 추정 할 수 있다.