Multivariate Data Analysis

Assignment #3

고려대학교 공과대학

산업경영공학부

2017170857 이우준

[Part 1] Multivariate Linear Regression

Dataset for Multivariate Linear Regression: Seoul Bike Sharing Demand Data Set

- Data Preprocessing

```
# Load the data & Preprocessing
part1_data <- read.csv("seoulBikeData.csv")
part1_data<-part1_data[!(part1_data$Functioning.Day=="No"),] #자전거가 운행하는 날만 측정
part1_data_target<-part1_data$Rented.Bike.Count
part1_data_input <- part1_data[,-c(1,2,14)] #날짜, 종속변수 제거, Fucntioning.Day 항목 제거
#명목형 변수 변환
nbike<-nrow(part1_data)
Spring <- rep(0,nbike)
Summer <- rep(0,nbike)
Autumn <- rep(0,nbike)
Winter <- rep(0,nbike)
Spring_idx <- which(part1_data$Seasons == "Spring")
Summer_idx <- which(part1_data$Seasons == "Summer")
Autmumn_idx <- which(part1_data$Seasons == "Autumn")
Winter_idx <- which(part1_data$Seasons == "Winter")
Spring[Spring_idx] <-1
Summer[Summer_idx] <-1
Autumn[Autmumn_idx] <-1
Winter[Winter_idx] <-1
seasons <- data.frame(Winter, Spring, Summer, Autumn)
part1_data_input$Holiday<-ifelse(part1_data_input$Holiday=="No Holiday",0,1)
part1_data_input<-cbind(part1_data_input[,-c(10)], seasons)
part1_data_final<-data.frame(part1_data_input,part1_data_target)
```

Seoulbike Data Set으로 Multivariate Linear Regression을 시행하기 전 데이터 셋을 사전처리한다. 자전거가 운행하는 날의 데이터로 분석을 진행하기 위해서 Functioning.Day column의 값이 "No"인 경우의 행을 제거해주었고, 시계열분석을 하지 않을 것이기 때문에 날짜 데이터를 제거해주었다. 남은 Input 데이터에서 명목형 항목인 Seasons항목을 One-Hot Encoding 방식으로, Holiday 항목을 0,1로 변환해주는 방식으로 데이터셋을 정리하였다.

- Splitting the Dataset

```
#Training Set ,Validation Set 분리
set.seed(123456)
part1_trn_idx <- sample(1:nbike, round(0.7*nbike))
part1_trn_data<- part1_data_final[part1_trn_idx,]
part1_val_data<- part1_data_final[-part1_trn_idx,]
```

전처리가 끝난 데이터 셋을 70:30 비율로 Training Set과 Validation Set으로 분리시켜준다.

[Q1] Forward Selection, Backward Elimination, Stepwise Selection 방식을 사용하여 MLR변수 선택 과정을 수행해 보시오. 각 방법론마다 Training dataset에 대한 Adjusted R2 및 소요 시간시간, Validation dataset에 대한 RMSE, MAE, MAPE를 산출하시오.

- Preliminary

```
#상한 & 하한선 설정 for Forward, Backward, Stepwise Selection
part1_tmp_x <- paste(colnames(part1_trn_data)[-15], collapse=" + ")
part1_tmp_x
part1_tmp_xy <- paste("part1_data_target ~ ", part1_tmp_x, collapse = "")
as.formula(part1_tmp_xy)
```

Forward Selection, Backward Elimination, Stepwise Selection 방식의 성능을 비교하기 세가지의 차원 축소법의 분석 전 상,하한선을 part1_tmp_x (변수를 다 선택한 케이스), part1_tmp_xy (변수를 다 선택하지 않은 케이스)로 설정하였다.

- Preliminary 2

```
perf_mat_part1 <- matrix(0, nrow = 4, ncol = 5)
rownames(perf_mat_part1) <- c("Forward Selection", " Backward Elimination", "Stepwise Selection",
"GA")
colnames(perf_mat_part1) <- c("Adjusted Rsquare", "Time", "RMSE", "MAE", "MAPE")


# Performance evaluation function for multiple linear regression
perf_eval_mlr <- function(tgt_y, pre_y){

  # RMSE
  rmse <- sqrt(mean((tgt_y - pre_y)^2))
  # MAE
  mae <- mean(abs(tgt_y - pre_y))
  # MAPE
  mape <- 100*mean(abs((tgt_y - pre_y)/tgt_y))

  return(c(rmse, mae, mape))
}
```

Q1과 Q2에서 측정할 차원축소별 법 성능비교를 위해 perf_mat_part1 matrix를 정의해주고, 각각의 RMSE, MAE, MAPE를 측정하기 위한 함수 perf_eval_mlr를 정의해 주었다.

- Forward Selection

```
#Forward Selection_Part1
start_time <- proc.time()
mlr_part1_forward <- step(lm(part1_data_target~ 1, data = part1_trn_data),
                  scope = list(upper = as.formula(part1_tmp_xy), lower = part1_data_target ~ 1),
                  direction="forward")
end_time<-proc.time()
perf_mat_part1[1,2] <- (end_time-start_time)[3]
perf_mat_part1[1,1]<-summary(mlr_part1_forward)$adj.r.squared
summary(mlr_part1_forward)
#Predict by Forward Selection
mlr_part1_forward_p <- predict(mlr_part1_forward, newdata = part1_val_data)
perf_mat_part1[1,c(3,4,5)]<-perf_eval_mlr(part1_val_data$part1_data_target, mlr_part1_forward_p)
perf_mat_part1
```

Forward Selection 방식으로 구한 model 의 값 (Adjusted Rsqaure, Time, RMSE, MAE, MAPE)을 측정하여 perf_mat_part1 matrix 에 저장해주었다. (Backward Selection, Stepwise Selection 까지 측정이 끝난 후 비교)

- Backward Selection

```
#Backward Selection_Part1
start_time <- proc.time()
mlr_part1_backward <- step(lm(part1_data_target~ ., data = part1_trn_data),
                scope = list(upper = as.formula(part1_tmp_xy), lower = part1_data_target ~ 1),
                direction="backward")
end_time<-proc.time()
perf_mat_part1[2,2] <- (end_time-start_time)[3]
perf_mat_part1[2,1]<-summary(mlr_part1_backward)$adj.r.squared
summary(mlr_part1_backward)
#Predict by Backward Selection
mlr_part1_backward_p <- predict(mlr_part1_backward, newdata = part1_val_data)
perf_mat_part1[2,c(3,4,5)]<-perf_eval_mlr(part1_val_data$part1_data_target, mlr_part1_backward_p)
perf_mat_part1
```

Backward Selection도 Forward Selecion과 동일한 방법으로 성능을 측정하여 기록해주었다.

- Stepwise Selection

```
#Stepwise Selection_Part1
start_time <- proc.time()
mlr_part1_stepwise <- step(lm(part1_data_target~ 1, data = part1_trn_data),
                        scope = list(upper = as.formula(part1_tmp_xy), lower = part1_data_target
~ 1),
                        direction="both")
end_time<-proc.time()
perf_mat_part1[3,2] <- (end_time-start_time)[3]
perf_mat_part1[3,1]<-summary(mlr_part1_stepwise)$adj.r.squared
summary(mlr_part1_stepwise)
#Predict by Stepwise Selection
mlr_part1_stepwise_p <- predict(mlr_part1_stepwise, newdata = part1_val_data)
perf_mat_part1[3,c(3,4,5)]<-perf_eval_mlr(part1_val_data$part1_data_target, mlr_part1_stepwise_p)
perf_mat_part1
```

Stepwise까지 시행한 후 perf_mat_part1 값이다.

	Adjusted Rsquare	Time	RMSE	MAE	MAPE
Forward Selection	0.538647	1.14	432.9288	322.225	163.2968
Backward Elimination	0.538647	0.36	432.9288	322.225	163.2968
Stepwise Selection	0.538647	1.35	432.9288	322.225	163.2968
GA	0	0	0	0	0

Forward Selection, Backward Selection, Stepwise Selection 모두 같은 모델이 나온 것을 알 수 있다. 총 14개의 Input Variable 중 12개의 Input Varible (Temperature, Hour, Humidity, Rainfall, Solar Radiation, Holiday, Wind Speed, Dew point temperature, Snowfall, Winter, Spring, Autumn) 이 선택었다. (Backward Selecion 기법에서는 Seasons 중 Winter, Spring, Summer 항목이 채택되었는데, 이는 one hot encoding 으로 구한 4개중 하나의 항목은 다중공산성 제거를 위해 빠지는 것이므로 4개중 어느 3개를 택해도 같은 결과를 가지기 때문에 같은 모델이라고 할 수 있다.)

세 기법 모두 같은 모델을 도출해 내어 Adjusted Rsquare, RMSE, MAE, MAPE의 값은 동일하지만 세 기법으로 구한 모델은 Full model에서 2개의 column 만을 제거한 모델이므로 Full model에서 하나씩 제거해 나가는 Backward Selection의 시행 시간이 0.36초로 가장 짧게 걸렸고, Forward Selection이 그 다음을 Forward Selecion보다 계산과정이 더 많은 Stepwise Selection이 가장 오랜 시간이 걸렸다.

- Summary of the model

```
Call:
lm(formula = part1_data_target ~ Temperature.캜. + Hour + Humidity... +
    Winter + Rainfall.mm. + Autumn + Solar.Radiation..MJ.m2. +
    Holiday + Wind.speed..m.s. + Dew.point.temperature.캜. +
    Snowfall..cm. + Spring, data = part1_trn_data)
Residuals:
     Min       1Q   Median       3Q      Max
-1338.19  -275.98   -61.73   215.02  2273.37
Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)              937.3882   110.9910   8.446  < 2e-16 ***
Temperature.캜.           13.3736     4.5833   2.918  0.00354 **
Hour                      27.9643     0.8984  31.127  < 2e-16 ***
Humidity...              -12.6180     1.2561 -10.045  < 2e-16 ***
Winter                  -180.3297    33.4472  -5.391 7.26e-08 ***
Rainfall.mm.             -58.9364     5.4229 -10.868  < 2e-16 ***
Autumn                   184.6350    20.7792   8.886  < 2e-16 ***
```

```
Solar.Radiation..MJ.m2.    -80.9815     9.1974  -8.805  < 2e-16 ***
Holiday                  -153.4877    26.7574  -5.736 1.02e-08 ***
Wind.speed..m.s.           18.4678     6.1947   2.981  0.00288 **
Dew.point.temperature.캜.   15.3730     4.8163   3.192  0.00142 **
Snowfall..cm.              34.1833    13.3084   2.569  0.01024 *
Spring                     35.7087    21.2182   1.683  0.09244 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 436.2 on 5913 degrees of freedom
Multiple R-squared:  0.5396,         Adjusted R-squared:  0.5386
F-statistic: 577.5 on 12 and 5913 DF,  p-value: < 2.2e-16
```

세 기법으로 구한 모델의 Summary를 보면 Spring을 제외한 모든 변수의 p-val가 유의수준 0.05이내에 들어오며, 모델의 p-val값도 0.05보다 자아 통계적으로 유의한 모델이라고 할 수 있다. 또한 모델의 Adjusted R squared값이 0.5386으로 어느정도 통계적 유의성을 가지고 있다고 할 수 있다.

[Q2] Adjusted R2를 Fitness function으로 하는 Genetic Algorithm 기반의 변수 선택 함수를 작성해 보시오. 작성한 함수를 이용하여 GA 기반 변수선택을 수행하고, 선택된 변수를 사용한 MLR의 Validation dataset에 대한 예측성능 (RMSE, MAE, MAPE), 변수 감소율, 수행시간의 세가지 관점에서 Forward Selection, Backward Elimination, Stepwise Selection 방식을 사용한 MLR과 비교해 보시오.

- Fitness function 정의

```r
# Fitness function: Adjusted R square
fit_ars <- function(string){
  sel_var_idx <- which(string == 1)
  # Use variables whose gene value is 1
  sel_x <- x[, sel_var_idx]
  xy <- data.frame(sel_x, y)
  # Training the model
  GA_lr <- lm(y ~ ., data = xy)
  return(summary(GA_lr)$adj.r.squared)
}
```

Adjusted R square 값을 기준으로 하는 Fitness Function을 구현하였다.

- Genetic Algotrithm기반 변수 선택

```r
x <- as.matrix(part1_trn_data[,-15])
y <- part1_trn_data[,15]
# Variable selection by Genetic Algorithm
start_time <- proc.time()
GA_ars <- ga(type = "binary", fitness = fit_ars, nBits = ncol(x),
             names = colnames(x), popSize = 50, pcrossover = 0.5,
             pmutation = 0.01, maxiter = 100, elitism = 2, seed = 123)
end_time <- proc.time()
end_time - start_time
perf_mat_part1[4,2] <- (end_time-start_time)[3]
best_var_idx <- which(GA_ars@solution[which.min(rowSums(GA_ars@solution)),] == 1)
# Model training based on the best variable subset
GA_trn_data_p1 <- part1_trn_data[,c(best_var_idx, 15)]
GA_tst_data_p1 <- part1_val_data[,c(best_var_idx, 15)]
GA_model_p1 <- lm(part1_data_target ~ ., GA_trn_data_p1)
summary(GA_model_p1)
perf_mat_part1[4,1]<-summary(GA_model_p1)$adj.r.squared
# Make prediction
GA_model_prob_p1 <- predict(GA_model_p1, newdata = GA_tst_data_p1)
perf_mat_part1[4,c(3,4,5)]<-perf_eval_mlr(GA_tst_data_p1$part1_data_target, GA_model_prob_p1)
perf_mat_part1
```

Genetic Algorithm을 기반으로 변수를 선택하는 모델을 구현하였다. 총 100번의 iteration을 진행하였으며, crossover 될 확률은 0.5, mutation이 될 확률을 0.01, 한 세대의 크기를 50으로 제한하여 진행하였다. Adjusted Rsquare을 Fitness Function으로 시행한 GA의 결과로 나온 solution은 두가지이다.

```
> GA_ars@solution
     Hour Temperature.캲. Humidity... Wind.speed..m.s. Visibility..10m. Dew.point.temperature.캲.
[1,]    1              1          1               1               0                        1
[2,]    1              1          1               1               0                        1
     Solar.Radiation..MJ.m2. Rainfall.mm. Snowfall..cm. Holiday Winter Spring Summer Autumn
[1,]                       1           1            1       1      1      1      1      1
[2,]                       1           1            1       1      0      1      1      1
```

1번 Solution과 2번 Solution은 Adjusted R Square값은 같지만 One Hot Encoding으로 변환한 Seasons 변수 중 Winter변수의 포함유무가 다르다. Q1의 답변에서와 같이 Seasons 변수가 4개가 다 선택이 되면 다중공산성이 일어나므로 Winter 변수가 포함이 안된 2번 Solution을 택하였다. (변수가 하나 더 적어서 차원 축소의 차원에서 보아도 변수의 개수가 더 적은 2번 Solution 택)

선택한 Solution을 바탕으로 구한 모델의 성능을 측정한 perf_mat_part1을 보면

	Adjusted Rsquare	Time	RMSE	MAE	MAPE
Forward Selection	0.538647	1.14	432.9288	322.225	163.2968
Backward Elimination	0.538647	0.36	432.9288	322.225	163.2968
Stepwise Selection	0.538647	1.35	432.9288	322.225	163.2968
GA	0.538647	24.36	432.9288	322.225	163.2968

Genetic Algorithm으로 구한 모델 또한 앞선 Forward Selection, Backward Elmination, Stepwise Selection으로 구한 모델과 동일하다는 것을 알 수 있다. RMSE ,MAE, MAPE 값 또한 동일하고, 선택된 변수 또한 동일하다. (One Hot Encoding으로 분류한 Seasons 의 변수 4개 중 다중공산성 제거를 위해 탈락된 변수 제외). 다만 Genetic Algorithm은 100번의 iteration 동안 50개의 모델들을 생성 후 Fitness를 측정해야하여 앞 선 세개의 모델과 달리 소요시간이 상당히 오래 걸렸다.

정리하자면 네 모델 모두 다 같은 모델을 도출해 내었지만 도출까지 걸린 시간이 차이가 났다. 이 Seoul bike data로 multivariate linear regression 모델을 구하는 것에 한정해서 Backward Elimination, Forward Selection, Stepwise Selection, GA 순으로 시간이 적게 걸린다는 것을 알 수 있다.

[Q3] Genetic Algorithm에서 변경 가능한 하이퍼파라미터들 (population size, Cross-over rate, Mutation rate 등)에 대해 각각 최소한 세가지 이상의 후보 값들을 선정선정(최소 27가지 이상의 조합)하고 각 조합에 대한 변수 선택 결과를 비교해 보시오. 최종 결과에 가장 큰 영향을 미치는 하이퍼파라미터는 무엇으로 나타났는가? 왜 그런 결과가 나타났다고 생각하는지 자신의 생각을 서술해 보시오.

Genetic Algorithm에서 변경 가능한 하이퍼 파라미터 중 population size, Cross-over-rate, Mutation rate 세가지를 골라 각각 세가지의 후보 값 (25,50,75), (0.3, 0.5, 0.7), (0.001, 0.01, 0.1)을 설정하여 변수 선택 결과를 비교해보았다.

■ Hyper Parameter 조합으로 구한 Genetic Algorithm의 결과

```
popsizelist<-list(25,50,75)
crossoverlist<-list(0.3, 0.5, 0.7)
mutationlist<-list(0.001,0.01,0.1)
q3perf_mat<- matrix(0, nrow = 27, ncol = 5)
colnames(q3perf_mat) <- c("Adjusted Rsquare", "Time", "RMSE", "MAE", "MAPE")
rowname_q3perf<-list()
repeat_time_q3<-1
repeat_time_q3
for (i in popsizelist){
  for (j in crossoverlist){
    for (k in mutationlist){
      rowname_q3perf<-
append(rowname_q3perf,(paste(as.character(i),as.character(j),as.character(k),sep=", ")))

      start_time <- proc.time()
      GA_ars_q3 <- ga(type = "binary", fitness = fit_ars, nBits = ncol(x),
                      names = colnames(x), popSize = i, pcrossover = j,
                      pmutation = k, maxiter = 100, elitism = 2, seed = 123)
      end_time <- proc.time()
      q3perf_mat[repeat_time_q3,2]<-(end_time-start_time)[3]

      best_var_idx <- which(GA_ars@solution[which.min(rowSums(GA_ars@solution)),] == 1)

      # Model training based on the best variable subset
      GA_trn_data_q3 <- part1_trn_data[,c(best_var_idx, 15)]
      GA_tst_data_q3 <- part1_val_data[,c(best_var_idx, 15)]

      GA_model_q3 <- lm(part1_data_target ~ ., GA_trn_data_q3)
      summary(GA_model_q3)
      q3perf_mat[repeat_time_q3,1]<-summary(GA_model_q3)$adj.r.squared


      # Make prediction
      GA_model_prob_q3 <- predict(GA_model_q3, newdata = GA_tst_data_q3)
      q3perf_mat[repeat_time_q3,c(3,4,5)]<-perf_eval_mlr(GA_tst_data_q3$part1_data_target,
GA_model_prob_q3)


      repeat_time_q3<-repeat_time_q3+1



    }
  }
}
rownames(q3perf_mat)<-rowname_q3perf
q3perf_mat
```

해당 코드를 이용해 구한 27가지 경우의 모델의 성능 (Adjusted Rsquare, Time, RMSE, MAE, MAPE)를 q3perf_mat에 기록하였다.

	Adjusted Rsquare	Time	RMSE	MAE	MAPE
25, 0.3, 0.001	0.538647	8.19	432.9288	322.225	163.2968
25, 0.3, 0.01	0.538647	9.29	432.9288	322.225	163.2968
25, 0.3, 0.1	0.538647	11.42	432.9288	322.225	163.2968
25, 0.5, 0.001	0.538647	11.02	432.9288	322.225	163.2968
25, 0.5, 0.01	0.538647	14.07	432.9288	322.225	163.2968
25, 0.5, 0.1	0.538647	16.31	432.9288	322.225	163.2968
25, 0.7, 0.001	0.538647	16.42	432.9288	322.225	163.2968

25, 0.7, 0.01	0.538647	15.74	432.9288	322.225	163.2968
25, 0.7, 0.1	0.538647	19.71	432.9288	322.225	163.2968
50, 0.3, 0.001	0.538647	15.08	432.9288	322.225	163.2968
50, 0.3, 0.01	0.538647	16.93	432.9288	322.225	163.2968
50, 0.3, 0.1	0.538647	19.49	432.9288	322.225	163.2968
50, 0.5, 0.001	0.538647	26.8	432.9288	322.225	163.2968
50, 0.5, 0.01	0.538647	26.26	432.9288	322.225	163.2968
50, 0.5, 0.1	0.538647	29.44	432.9288	322.225	163.2968
50, 0.7, 0.001	0.538647	39.15	432.9288	322.225	163.2968
50, 0.7, 0.01	0.538647	34.97	432.9288	322.225	163.2968
50, 0.7, 0.1	0.538647	44.72	432.9288	322.225	163.2968
75, 0.3, 0.001	0.538647	22.33	432.9288	322.225	163.2968
75, 0.3, 0.01	0.538647	22.14	432.9288	322.225	163.2968
75, 0.3, 0.1	0.538647	32.46	432.9288	322.225	163.2968
75, 0.5, 0.001	0.538647	42.74	432.9288	322.225	163.2968
75, 0.5, 0.01	0.538647	43.09	432.9288	322.225	163.2968
75, 0.5, 0.1	0.538647	42.76	432.9288	322.225	163.2968
75, 0.7, 0.001	0.538647	57.94	432.9288	322.225	163.2968
75, 0.7, 0.01	0.538647	48.83	432.9288	322.225	163.2968
75, 0.7, 0.1	0.538647	51.87	432.9288	322.225	163.2968

해당 표의 Row의 이름은 순서대로 population size, Crossover rate, Mutation rate이다. 결과를 보면 알 수 있듯 27가지의 조합 모두 앞선 Q1과 Q2에서 구한 최적의 모델과 같은 모델이 나온 것을 알 수 있다. 모든 27가지 모델에서 14개의 입력변수 중 12개의 변수가 선택되었으며, Adjusted R Square, RMSE, MAE, MAPE 값 모두 0.538647, 432.9288, 322.225, 163.2968로 같다는 것을 알 수 있다.

각 모델의 차이점은 모델을 구하기까지의 걸린 시간이다. 해당 표를 보면 세가지의 Hyper Parameter 중 Mutation rate에 의한 시간 변화율이 가장 작고, Crossover rate이 그 다음으로 작다, 마지막으로 Population Size에 의한 소요시간 변화율이 가장 크다는 것을 알 수 있다. 이를 통해 Genetic Algorithm에서 가장 큰 영향을 끼치는 Hyperparmeter는 **population size**라고 할 수 있다. 물론 세가지의 Hyper parameter들이 동일한 비교값, 단위를 가진 것이 아니기 때문에 절대적으로 population size가 가장 중요하다고 할 수 는 없다.

[Part 2] Logistic Regression

Dataset for Logistic Regression: Breast Cancer Wisconsin (Diagnostic) Data Set

- Data Preprocessing

```
# 이상치 기준 분류 만들기 함수
boxplot_outliers <-function(target){
  valuelist <-boxplot(target)$stats
  return(c(valuelist[1,],valuelist[5,]))
}

part2_data  <- read.csv("data.csv")
part2_data_input <- part2_data[,-c(1,2,33)] #1 은 ID, 2 는 목적변수, 33 은 잉여 변수 (all NA 값)
part2_data_input_scaled <- scale(part2_data_input, center = TRUE, scale = TRUE)
part2_data_target <- part2_data$diagnosis
part2_data_target <- ifelse(part2_data_target=="B",0,1)
part2_data_scaled <- data.frame(part2_data_input_scaled, part2_data_target)
#이상치 제거
outliers <- matrix(0, nrow = 30, ncol = 2)
rownames(outliers) <- names(part2_data_input)
colnames(outliers) <- c("LCL", "UCL")
for(i in 1:30){
  outliers[i,]<-boxplot_outliers(part2_data_input_scaled[,i])
}
for (i in 1:30){
  part2_data_scaled[,i]<-ifelse(part2_data_scaled[,i] < outliers[i,1] | part2_data_scaled[,i] >
outliers[i,2], NA, part2_data_scaled[,i])
}
sum(is.na(part2_data_scaled))
part2_data_scaled<-na.omit(part2_data_scaled)
ndata2<-nrow(part2_data_scaled)
```

Breast Cancer Wisconsin (Diagnostic) Data Set으로 Logistic Regression을 시행하기 전 데이터 셋을 사전처리한다. 잉여 column인 33번 컬럼과 ID coulmn을 제거해 주었고, 전부다 표준화해주었다. +- 1.5 IQR을 벗어나는 이상치들을 찾아내주어 NA값으로 변환 시킨후 데이터 셋에서 제거해주었다.

- Splitting the Dataset

```
#Training Set ,Validation Set 분리
set.seed(123456)
part2_trn_idx <- sample(1:ndata2, round(0.7*ndata2))
part2_trn_data<- part2_data_scaled[part2_trn_idx,]
part2_val_data<- part2_data_scaled[-part2_trn_idx,]
```

전처리가 끝난 데이터 셋을 70:30 비율로 Training Set과 Validation Set으로 분리시켜주었다.

[Q4] Forward Selection, Backward Elimination, Stepwise Selection 방식을 사용하여 Logistic Regression변수 선택 과정을 수행해보시오. 각 방법론마다 Training dataset에 대한 AUROC 및 소요시간, Validation dataset에 대한 AUROC, Accuracy, BCR, F1-Measure를 산출하시오.

- Preliminary

```
#상한 & 하한선 설정 for Forward, Backward, Stepwise Selection
part2_tmp_x <- paste(colnames(part2_trn_data)[-31], collapse=" + ")
part2_tmp_x
part2_tmp_xy <- paste("part2_data_target ~ ", part2_tmp_x, collapse = "")
as.formula(part2_tmp_xy)
```

Forward Selection, Backward Elimination, Stepwise Selection 방식의 성능을 비교하기 세가지의 차원 축소법의 분석 전 상,하한선을 part2_tmp_x (변수를 다 선택한 케이스), part2_tmp_xy (변수를 다 선택하지 않은 케이스)로 설정하였다.

- Preliminary 2

```
perf_mat_part2 <- matrix(0, nrow = 4, ncol = 6)
rownames(perf_mat_part2) <- c("Forward", "Backward","Stepwise","GA")
colnames(perf_mat_part2) <- c("T_AUROC", "Time", "V_AUROC", "Accuracy", "BCR", "F1-Measure")


# Performance evaluation function for logistic regression
perf_eval_logi <- function(cm){

  # True positive rate: TPR (Recall)
  TPR <- cm[2,2]/sum(cm[2,])
  # Precision
  PRE <- cm[2,2]/sum(cm[,2])
  # True negative rate: TNR
  TNR <- cm[1,1]/sum(cm[1,])
  # Simple Accuracy
  ACC <- (cm[1,1]+cm[2,2])/sum(cm)
  # Balanced Correction Rate
  BCR <- sqrt(TPR*TNR)
  # F1-Measure
  F1 <- 2*TPR*PRE/(TPR+PRE)

  return(c(ACC, BCR, F1))
}
```

Q4 과 Q5 에서 측정할 차원축소별 법 성능비교를 위해 perf_mat_part2 matrix 를 정의해주고, 각각의 Accuracy, BCR, F1-Measure 를 측정하기 위한 함수 perf_eval_logi 를 정의해 주었다.

- Forward Selection

```
#Forward Selection_part2
start_time <- proc.time()
part2_forward <- step(glm(part2_data_target~ 1, data = part2_trn_data),
                      scope = list(upper = as.formula(part2_tmp_xy), lower = part2_data_target ~
1),
                      direction="forward")
end_time <- proc.time()
perf_mat_part2[1,2]<-(end_time-start_time)[3]
summary(part2_forward)
#Training AUROC
prob <- predict(part2_forward, type = "response")
pred <- prediction(prob,part2_trn_data$part2_data_target)
perf_mat_part2[1,1]<-as.numeric(performance(pred, measure = "auc")@y.values)
# Make prediction
part2_forward_prob <- predict(part2_forward, type = "response", newdata = part2_val_data)
part2_forward_prey <- rep(0, nrow(part2_val_data))
part2_forward_prey[which(part2_forward_prob >= 0.5)] <- 1
part2_forward_cm <- table(part2_val_data$part2_data_target, part2_forward_prey)
part2_forward_cm
# Peformance evaluation
perf_mat_part2[1,c(4,5,6)]<- perf_eval_logi(part2_forward_cm)
perf_mat_part2
#Validation Auroc
pred <- prediction(part2_forward_prob,part2_val_data$part2_data_target)
perf_mat_part2[1,3]<-as.numeric(performance(pred, measure = "auc")@y.values)
perf_mat_part2
```

Forward Selection 기법으로 구한 모델의 summary는 아래와 같다.

```
> summary(part2_forward)
Call:
glm(formula = part2_data_target ~ area_worst + concavity_mean +
    area_mean + texture_mean + concavity_se + smoothness_se +
    concave.points_worst + compactness_mean + fractal_dimension_worst +
    area_se + fractal_dimension_se + perimeter_worst + radius_se +
    symmetry_mean, data = part2_trn_data)
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.53584  -0.14263  -0.02811   0.09684   0.89258
Coefficients:
                       Estimate Std. Error t value Pr(>|t|)
(Intercept)             0.48044    0.02581  18.615  < 2e-16 ***
area_worst             -0.04895    0.23718  -0.206 0.836641
concavity_mean          0.30364    0.07757   3.914 0.000115 ***
area_mean              -0.18602    0.13898  -1.338 0.181892
texture_mean            0.07690    0.01782   4.316 2.25e-05 ***
concavity_se           -0.13032    0.06197  -2.103 0.036409 *
smoothness_se           0.06853    0.02867   2.390 0.017555 *
concave.points_worst    0.09366    0.04923   1.903 0.058185 .
compactness_mean       -0.18745    0.05399  -3.472 0.000603 ***
fractal_dimension_worst  0.13273    0.05227   2.539 0.011677 *
area_se                 0.75633    0.30586   2.473 0.014037 *
fractal_dimension_se   -0.09923    0.06527  -1.520 0.129618
perimeter_worst         0.33088    0.18202   1.818 0.070226 .
radius_se              -0.28423    0.15837  -1.795 0.073848 .
symmetry_mean           0.03395    0.02281   1.489 0.137751
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for gaussian family taken to be 0.05407674)
    Null deviance: 53.900  on 278  degrees of freedom
Residual deviance: 14.276  on 264  degrees of freedom
AIC: -5.5915
Number of Fisher Scoring iterations: 2
```

총 30 개의 Input Variable 중 14 개의 Variable 만이 선택되었다. 이 중 8 개의 variable 의 p-val 값이 유의수준 0.05 보다 작아 통계적 유의성을 가진다. 해당 모델의 성능이 나타난 perf_mat_part2 는 backward elimination 과 stepwise selection 의 성능 측정이 끝난 후 한꺼번에 비교하겠다.

- Backward Elimination

```
#Backward Selection_part2
start_time <- proc.time()
part2_backward <- step(glm(part2_data_target~ .,family=binomial, data = part2_trn_data),
                       scope = list(upper = as.formula(part2_tmp_xy), lower = part2_data_target ~
1),
                       direction="backward",trace=1)
summary(part2_backward)
end_time <- proc.time()
perf_mat_part2[2,2]<-(end_time-start_time)[3]
summary(part2_backward)
#Training AUROC
prob <- predict(part2_backward, type = "response")
pred <- prediction(prob,part2_trn_data$part2_data_target)
perf_mat_part2[2,1]<-as.numeric(performance(pred, measure = "auc")@y.values)
# Make prediction
part2_backward_prob <- predict(part2_backward, type = "response", newdata = part2_val_data)
part2_backward_prey <- rep(0, nrow(part2_val_data))
part2_backward_prey[which(part2_backward_prob >= 0.5)] <- 1
part2_backward_cm <- table(part2_val_data$part2_data_target, part2_backward_prey)
part2_backward_cm
# Peformance evaluation
perf_mat_part2[2,c(4,5,6)]<- perf_eval_logi(part2_backward_cm)
#Validation Auroc
pred <- prediction(part2_backward_prob,part2_val_data$part2_data_target)
perf_mat_part2[2,3]<-as.numeric(performance(pred, measure = "auc")@y.values)
```

Backward Elimination 기법으로 구한 모델의 summary 는 아래와 같다.

```
> summary(part2_backward)
Call:
glm(formula = part2_data_target ~ radius_mean + texture_mean +
    perimeter_mean + compactness_mean + concave.points_mean +
    perimeter_se + fractal_dimension_se + radius_worst + fractal_dimension_worst,
    family = binomial, data = part2_trn_data)
Deviance Residuals:
       Min          1Q      Median          3Q         Max
-3.531e-04  -2.000e-08  -2.000e-08   2.000e-08   4.171e-04
Coefficients:
                       Estimate Std. Error z value Pr(>|z|)
(Intercept)               37.87    4947.71   0.008    0.994
radius_mean            -9121.61  563888.13  -0.016    0.987
texture_mean             222.04   13643.99   0.016    0.987
perimeter_mean          9114.96  569441.06   0.016    0.987
compactness_mean       -1208.77   73737.96  -0.016    0.987
concave.points_mean     1037.45   65640.36   0.016    0.987
perimeter_se             429.18   27188.06   0.016    0.987
fractal_dimension_se    -768.10   48803.53  -0.016    0.987
radius_worst             769.09   49922.34   0.015    0.988
fractal_dimension_worst  514.94   32228.22   0.016    0.987
(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 3.2072e+02  on 278  degrees of freedom
Residual deviance: 7.3186e-07  on 269  degrees of freedom
AIC: 20
Number of Fisher Scoring iterations: 25
```

Backward Elimination 으로는 총 30 개의 Input Varibale 중 21 개의 variable 들이 탈락되고 9 개의 variable 만이 선택되었다. 선택된 variable 들이 모두 다 p-val 값이 0.98 이상이여 통계적 유의성을 띄지는 않는다.

- Stepwise Selection

```
#Stepwise Selection_part2
start_time <- proc.time()
part2_stepwise <- step(glm(part2_data_target~ 1, data = part2_trn_data),
                       scope = list(upper = as.formula(part2_tmp_xy), lower = part2_data_target ~
1),
                       direction="both")
end_time <- proc.time()
perf_mat_part2[3,2]<-(end_time-start_time)[3]
summary(part2_stepwise)
#Training AUROC
prob <- predict(part2_stepwise, type = "response")
pred <- prediction(prob,part2_trn_data$part2_data_target)
perf_mat_part2[3,1]<-as.numeric(performance(pred, measure = "auc")@y.values)
# Make prediction
part2_stepwise_prob <- predict(part2_stepwise, type = "response", newdata = part2_val_data)
part2_stepwise_prey <- rep(0, nrow(part2_val_data))
part2_stepwise_prey[which(part2_stepwise_prob >= 0.5)] <- 1
part2_stepwise_cm <- table(part2_val_data$part2_data_target, part2_forward_prey)
part2_stepwise_cm
# Peformance evaluation
perf_mat_part2[3,c(4,5,6)]<- perf_eval_logi(part2_stepwise_cm)
#Validation Auroc
pred <- prediction(part2_stepwise_prob,part2_val_data$part2_data_target)
perf_mat_part2[3,3]<-as.numeric(performance(pred, measure = "auc")@y.values)
```

Stepwise Selection 기법으로 구한 모델의 summary는 아래와 같다.

```
> summary(part2_stepwise)
Call:
glm(formula = part2_data_target ~ area_worst + concavity_mean +
    texture_mean + concavity_se + smoothness_se + concave.points_worst +
    compactness_mean + fractal_dimension_worst + area_se + fractal_dimension_se,
    data = part2_trn_data)
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.53470  -0.13870  -0.03063   0.10859   0.79012
Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)              0.46308    0.02106  21.985  < 2e-16 ***
area_worst               0.15766    0.07282   2.165 0.031268 *
concavity_mean           0.29124    0.07529   3.868 0.000138 ***
texture_mean             0.08110    0.01770   4.583 7.03e-06 ***
concavity_se            -0.10254    0.05941  -1.726 0.085496 .
smoothness_se            0.04997    0.02613   1.912 0.056911 .
concave.points_worst     0.11310    0.04700   2.406 0.016800 *
compactness_mean        -0.16246    0.04860  -3.343 0.000948 ***
fractal_dimension_worst  0.16693    0.04119   4.053 6.64e-05 ***
area_se                  0.31703    0.09404   3.371 0.000859 ***
fractal_dimension_se    -0.13990    0.05893  -2.374 0.018301 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for gaussian family taken to be 0.05477081)
    Null deviance: 53.900  on 278  degrees of freedom
Residual deviance: 14.679  on 268  degrees of freedom
AIC: -5.8378
Number of Fisher Scoring iterations: 2
```

Stepwise Selection 기법으로 선택된 variable은 총 10개의 variable이다 이중 8개의 variable의 p-val이 0.05보다 작아 통계적으로 유의미한 variable이라고 할 수 있다.

이제 Forward Selection, Backward Elimination, Stepwise Selection 기법으로 구한 모델들의 성능비교를 해보았다.

	T_AUROC	Time	V_AUROC	Accuracy	BCR	F1-Measure
Forward	0.991555	0.58	0.997447	0.966387	0.916515	0.913043
Backward	1	5	0.943617	0.94958	0.938537	0.884615
Stepwise	0.994813	0.65	0.998298	0.966387	0.916515	0.913043
GA	0	0	0	0	0	0

Stepwise Selection 기법으로 구한 모델의 변수의 수는 10개이고, Forward Selection 기법으로 구한 모델의 변수의 수는 14개이다. 소요시간은 Stepwise 기법이 0.07초 더 걸리긴 했지만 이때의 둘의 Accuracy, BCR, F1-Measure 값이 같은 것을 보면 성능상으론 둘이 동등하다고 할 수 있다. 고로 Stepwise Selection 기법이 Forward Selection 보다 더 나은 모델을 찾았다고 할 수 있다.

Backward Elimnation으로 구한 모델은 소요시간 도 5초로 제일 길고 BCR을 제외한 F1-Measure 값과 Accuracy 모두 Forward Selection이나, Stepwise Selection 보다 열등하다. 고로 Stepwise Selection 기법이나, Forward Selection 기법이 Backward Elimination 기법보다 더 나은 모델을 찾았다고 할 수 있다.

정리하자면 Stepwise Selection, Forward Selection, Backward Elimination 순으로 더 좋은 모델을 생성해 내었다.

하지만 조금 특이한 점이 있다. 특이한 점은 세 모델 다 Training셋에 대한 AUROC가 1이거나 1에 매우 근사한 값을 갖는 이상할 정도로 이상적인 모델이라는 점이다. 이에 대한 나름의 해석으론 세가지 기법으로 모두 다 변

수의 갯수 유의미하게 줄이지 못하여 curse of dimensionality에 걸린 상태라고 생각하였다. 이로 인한 Overfitting이 유발되었다. Overfit 되었다는 것은 즉 테스트 셋에 대해 높은 정확도를 띄는 것이니, Training 셋에 대한 AUROC 값이 1또는 1과 매우 근사한 수가 나왔다고 생각하였다.

그렇다면 Validate Set에 대한 AUROC또한 1에 가깝게 나왔는데 이는 Training Set과 Validation Set의 데이터의 차이가 거의 없거나, Model이 완벽하게 생성되어 Validation Set에 대해서도 완벽하게 분류해내는 수준까지 model이 학습되었다고 생각하였다.

[Q5] AUROC를 Fitness function으로 하는 Genetic Algorithm 기반의 변수 선택 함수를 작성해 보시오.작성한 함수를 이용하여 GA 기반 변수 선택을 수행하고, 선택된 변수를 사용한 Logistic Regression의 Validation dataset에 대한 분류 성능(AUROC, Accuracy, BCR, F1-Measure), 변수 감소율, 수행 시간의 세 가지 관점에서 Forward Selection, Backward Elimination, Stepwise Selection 방식을 사용한 Logistic Regression과 비교해보보시오.

AUROC의 값을 바탕으로 하는 Fitness Function의 구현은 아래와 같다.

- Fitness Function 정의

```
fit_auroc <- function(string){
  sel_var_idx <- which(string == 1)
  # Use variables whose gene value is 1
  sel_x <- x[, sel_var_idx]
  xy <- data.frame(sel_x, y)
  # Training the model
  GA_lr <- glm(y ~ ., family = binomial, data = xy)
  GA_lr_prob <- predict(GA_lr, type = "response", newdata = xy)
  pred <- prediction(GA_lr_prob,y)
  return(as.numeric(performance(pred, measure = "auc")@y.values))
}
x <- as.matrix(part2_trn_data[,-31])
y <- part2_trn_data[,31]
```

- Genetic Algorithm

```
# Variable selection by Genetic Algorithm
start_time <- proc.time()
GA_auroc <- ga(type = "binary", fitness = fit_auroc, nBits = ncol(x),
           names = colnames(x), popSize = 50, pcrossover = 0.5,
           pmutation = 0.01, maxiter = 100, elitism = 2, seed = 123)
end_time <- proc.time()
perf_mat_part2[4,2] <- (end_time-start_time)[3]
best_var_idx <- which(GA_auroc@solution[which.min(rowSums(GA_auroc@solution)),] == 1)
# Model training based on the best variable subset
GA_trn_data_q5 <- part2_trn_data[,c(best_var_idx, 31)]
GA_tst_data_q5 <- part2_val_data[,c(best_var_idx, 31)]
GA_model_q5 <- glm(part2_data_target ~ ., family=binomial, GA_trn_data_q5)
summary(GA_model_q5)
#Training AUROC
prob_q5 <- predict(GA_model_q5, type = "response")
pred_q5 <- prediction(prob_q5,GA_trn_data_q5$part2_data_target)
perf_mat_part2[4,1]<-as.numeric(performance(pred_q5, measure = "auc")@y.values)
# Make prediction
GA_model_prob_q5 <- predict(GA_model_q5, type = "response", newdata = GA_tst_data_q5)
GA_model_prey_q5 <- rep(0, nrow(GA_tst_data_q5))
GA_model_prey_q5[which(GA_model_prob_q5 >= 0.5)] <- 1
GA_model_cm_q5 <- table(GA_tst_data_q5$part2_data_target, GA_model_prey_q5)
GA_model_cm_q5
# Peformance evaluation
perf_mat_part2[4,c(3,4,5)]<- perf_eval_logi(GA_model_cm_q5)
```

```
#Validation Auroc
pred <- prediction(GA_model_prob_q5,part2_val_data$part2_data_target)
perf_mat_part2[4,3]<-as.numeric(performance(pred, measure = "auc")@y.values)
```

Genetic Algorithm을 기반으로 변수를 선택하는 모델을 구현하였다. 총 100번의 iteration을 진행하였으며, crossover 될 확률은 0.5, mutation이 될 확률을 0.01, 한 세대의 크기를 50으로 제한하여 진행하였다. Adjusted Rsquare을 Fitness Function으로 시행한 GA의 결과로 나온 solution은 8가지이다. 이 중 변수의 개수가 가장 적은 것을 선택하였다.

선택된 모델은 아래와 같다

```
> summary(GA_model_q5)
Call:
glm(formula = part2_data_target ~ ., family = binomial, data = GA_trn_data_q5)
Deviance Residuals:
       Min          1Q      Median          3Q         Max
-1.133e-04  -2.100e-08  -2.100e-08   2.100e-08   1.428e-04
Coefficients:
                         Estimate Std. Error z value Pr(>|z|)
(Intercept)             1.047e+02  5.250e+04   0.002    0.998
radius_mean            -3.357e+02  8.574e+05   0.000    1.000
texture_mean            7.920e+01  2.038e+04   0.004    0.997
area_mean              -3.624e+01  1.056e+06   0.000    1.000
compactness_mean       -1.775e+02  6.770e+04  -0.003    0.998
concave.points_mean     2.992e+02  1.094e+05   0.003    0.998
symmetry_mean           1.388e+01  1.496e+04   0.001    0.999
texture_se             -5.230e-01  1.706e+04   0.000    1.000
perimeter_se            4.069e+01  3.955e+04   0.001    0.999
smoothness_se          -3.676e+01  9.094e+04   0.000    1.000
compactness_se         -7.650e+01  8.157e+04  -0.001    0.999
concave.points_se       7.334e+01  1.128e+05   0.001    0.999
symmetry_se             2.833e+01  2.925e+04   0.001    0.999
fractal_dimension_se   -1.492e+02  5.676e+04  -0.003    0.998
radius_worst           -5.543e+02  8.419e+05  -0.001    0.999
perimeter_worst         5.011e+02  4.601e+05   0.001    0.999
area_worst              7.567e+02  8.102e+05   0.001    0.999
smoothness_worst        1.991e+01  7.376e+04   0.000    1.000
compactness_worst      -2.708e+01  6.130e+04   0.000    1.000
concavity_worst         7.849e+01  3.857e+04   0.002    0.998
concave.points_worst   -1.060e+02  1.203e+05  -0.001    0.999
fractal_dimension_worst  1.232e+02  3.595e+04   0.003    0.997
(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 3.2072e+02  on 278  degrees of freedom
Residual deviance: 1.0635e-07  on 257  degrees of freedom
AIC: 44
Number of Fisher Scoring iterations: 25
```

Genetic Algorithm으로 구한 모델은 총 30개의 Input Variable 중에서 21개의 Input Variable으로 모델이 이루어져 있다. 각 Variable의 p-val값은 1.0에 매우 가깝거나, 1이므로 통계적 유의성이 전혀 없다고 할 수 있다. 즉 차원 축소가 정상적으로 이루어지지 않았다고 할 수 있다.

모델의 성능을 보기위해 perf_mat_part2를 보면

	T_AUROC	Time	V_AUROC	Accuracy	BCR	F1-Measure
Forward	0.991555	0.58	0.997447	0.966387	0.916515	0.913043
Backward	1	5	0.943617	0.94958	0.938537	0.884615
Stepwise	0.994813	0.65	0.998298	0.966387	0.916515	0.913043
GA	1	42.65	0.966596	0.949580	0.938537	0.884615

GA기법으로 구한 모델의 Accuracy, BCR, F1-Measure이 Backward Elimination기법으로 구한 모델과 같은 값을 갖는다는 것을 알 수 있다. Q4에서 구했듯 Backward Elimination기법으로 구한 모델은 Stepwise기법과 Forward Selection기법으로 구한 모델보다 상대적으로 성능이 안 좋은 모델이라고 할 수 있다. 이러한 모델과 같은 성능을 가지고 있고 또한 변수의 수는 Backward Elimination기법으로 구한 모델의 경우 9개의 변수로 구성되어있지만, GA기법으로 구한 모델의 변수는 21개의 변수로 이루어져 있어, Backward Elimination 기법보다 모델의 완성도가 떨어진다고 할 수있다. 소요시간 또한 4개의 기법중 가장 오래 걸려 가장 최악의 모델이라고 말할 수 있다.

Genetic Algorithm으로 구한 모델의 성능이 안 좋은 이유는 바로 Fitness Function에 있다고 생각한다. Q5에 적용한 Fitness Function은 AUROC 값을 기준으로 삼았는데, Q4의 답안에서도 적어듯 변수가 유의미하게 줄어들지 않은 경우 curse of dimensionality에 빠져 모델이 overfit하게 된다. Overfitting 된 상황은 테스트 셋을 필요 이상으로 완벽하게 학습하게 되어, 모든 트레이닝 데이터를 완벽하게 분류해내며 AUROC값이 1이 되는 특이한 상황이 발생하게 된다. 고로 AUROC 값을 Fitness Function의 기준을 잡게 되면 변수의 감소를 이끌어 내지 않고, 예외적인 상황에 수렴하게 되므로 차원축소를 위한 Fitness function으로는 적합하지 않다고 할 수 있다.

[Q6] Genetic Algorithm에서 변경 가능한 하이퍼파라미터들 (population size, Cross-over rate, Mutation rate 등등)에 대해 각각 최소한 세가지 이상의 후보 값들을 선정(최소 27가지 이상의 조합)하고 각 조합에 대한 변수 선택 결과를 비교해보시오. 최종 결과에 가장 큰 영향을 미치는 하이퍼파라미터는 무엇으로 나타났는가? 왜 그런 결과가 나타났다고 생각하는지 자신의 생각을 서술해보시오.

- Hyper Parameter 조합으로 구한 Genetic Algorithm의 결과

```
popsizelist<-list(25,50,75)
crossoverlist<-list(0.3, 0.5, 0.7)
mutationlist<-list(0.001,0.01,0.1)
q6perf_mat<- matrix(0, nrow = 27, ncol = 6)
colnames(q6perf_mat) <- c("AUROC", "Time", "Length", "Accuracy", "BCR", "F1-Measure")
rowname_q6perf<-list()
repeat_time_q6<-1
for (i in popsizelist){
  for (j in crossoverlist){
    for (k in mutationlist){

      rowname_q6perf<-
append(rowname_q6perf,(paste(as.character(i),as.character(j),as.character(k),sep=", ")))

      start_time <- proc.time()
      GA_auroc_q6 <- ga(type = "binary", fitness = fit_auroc, nBits = ncol(x),
                        names = colnames(x), popSize = i, pcrossover = j,
                        pmutation = k, maxiter = 100, elitism = 2, seed = 123)
      end_time <- proc.time()
      q6perf_mat[repeat_time_q6,2]<-(end_time-start_time)[3]

      best_var_idx_q6 <- which(GA_auroc_q6@solution[which.min(rowSums(GA_auroc_q6@solution)),] ==
1)
      q6perf_mat[repeat_time_q6,3]<-min(rowSums(GA_auroc_q6@solution))


      # Model training based on the best variable subset
      GA_trn_data_q6 <- part2_trn_data[,c(best_var_idx_q6, 31)]
      GA_tst_data_q6 <- part2_val_data[,c(best_var_idx_q6, 31)]

      GA_model_q6 <- glm(part2_data_target ~ ., family=binomial, GA_trn_data_q6)

      #Training AUROC
      prob_q6 <- predict(GA_model_q6, type = "response")
      pred_q6 <- prediction(prob_q6,GA_trn_data_q6$part2_data_target)
      q6perf_mat[repeat_time_q6,1]<-as.numeric(performance(pred_q6, measure = "auc")@y.values)
```

```
      # Make prediction
      GA_model_prob_q6 <- predict(GA_model_q6, type = "response", newdata = GA_tst_data_q6)
      GA_model_prey_q6 <- rep(0, nrow(GA_tst_data_q6))
      GA_model_prey_q6[which(GA_model_prob_q6 >= 0.5)] <- 1
      GA_model_cm_q6 <- table(GA_tst_data_q6$part2_data_target, GA_model_prey_q6)
      # Peformance evaluation
      q6perf_mat[repeat_time_q6,c(4,5,6)]<-perf_eval_logi(GA_model_cm_q6)

      repeat_time_q6<-repeat_time_q6+1


    }
  }
}
rownames(q6perf_mat)<-rowname_q6perf
q6perf_mat
```

해당 코드를 이용해 구한 27가지 경우의 모델의 성능 (AUROC, Time, Length, Accuracy ,BCR, F1-Measure)를 q6perf_mat에 기록하였다.

	AUROC	Time	Length	Accuracy	BCR	F1-Measure
25, 0.3, 0.001	1	12.74	20	0.97479	0.969316	0.941176
25, 0.3, 0.01	1	11.77	17	0.966387	0.948908	0.92
25, 0.3, 0.1	1	14.37	16	0.966387	0.948908	0.92
25, 0.5, 0.001	1	18.14	20	0.966387	0.964034	0.923077
25, 0.5, 0.01	1	23.31	23	0.94958	0.953381	0.888889
25, 0.5, 0.1	1	31.86	15	0.966387	0.964034	0.923077
25, 0.7, 0.001	1	34.82	22	0.983193	0.97457	0.96
25, 0.7, 0.01	1	26.19	20	0.991597	0.994667	0.980392
25, 0.7, 0.1	1	26.67	17	0.966387	0.964034	0.923077
50, 0.3, 0.001	1	21.77	18	0.957983	0.943736	0.901961
50, 0.3, 0.01	1	24.46	20	0.932773	0.92805	0.851852
50, 0.3, 0.1	1	28.84	17	0.92437	0.891807	0.823529
50, 0.5, 0.001	1	33.47	18	0.957983	0.943736	0.901961
50, 0.5, 0.01	1	39.22	21	0.94958	0.938537	0.884615
50, 0.5, 0.1	1	44.36	19	0.941176	0.933308	0.867925
50, 0.7, 0.001	1	51.89	19	0.941176	0.933308	0.867925
50, 0.7, 0.01	1	50.84	17	0.94958	0.938537	0.884615
50, 0.7, 0.1	1	52.8	14	0.966387	0.964034	0.923077
75, 0.3, 0.001	1	28.03	16	0.966387	0.964034	0.923077
75, 0.3, 0.01	1	29.72	17	0.957983	0.943736	0.901961
75, 0.3, 0.1	1	41.71	17	0.957983	0.943736	0.901961
75, 0.5, 0.001	1	56.5	21	0.94958	0.953381	0.888889
75, 0.5, 0.01	1	51.95	18	0.966387	0.964034	0.923077
75, 0.5, 0.1	1	59.42	14	0.941176	0.933308	0.867925
75, 0.7, 0.001	1	73.78	17	0.941176	0.933308	0.867925
75, 0.7, 0.01	1	89.44	21	0.941176	0.933308	0.867925
75, 0.7, 0.1	1	92.86	14	0.966387	0.964034	0.923077

해당 표의 Row의 이름은 순서대로 population size, Crossover rate, Mutation rate이다. 결과를 보면 알 수 있듯 27가지의 조합 모두 앞선 Q4과 Q5에서 구한 최적의 모델과 같이 AUROC 값이 1인 여러가지 모델이 선정되었다. AUROC가 기준이 되었기 때문에 선택된 변수의 수는 적은건 14개에서 많은건 23개까지 변수들이 선택되었다. Hyperparmeter별 비교를 해보았을땐 Mutation rate이 높아질수록 선택된 변수의 수가 적어지는 것을 알 수 있다. 그에 반해 Crossover rate의 변화는 선택된 변수의 수나, Accuracy, BCR, F1-Measure의 유의미한 변화를 보여주지 않았다. 마지막으로 population size를 고려해보면 populationsize가 커질수록 소요시간이 오래걸리지만, 모델의 성능측면에서는 큰 차이를 보이지않았다. 이를 통해 Genetic Algorithm의 Hyperparmeter중 가장 큰 영향을 미치는 항목은 Mutation rate이라고 할 수 있다.