Multivariate Data Analysis

Assignment #5

고려대학교 공과대학

산업경영공학부

2017170857 이우준

[Q1] 입력변수 속성 확인

1.1 종속 변수 확인 및 필요 없는 변수 제거

확인주어진 데이터 셋의 40개의 변수와 260601개의 instance로 이루어져 있다. 이 중 train_labels.csv 파일에 포함되어 있는 damagae_grade 변수가 종속변수이며, (1,2,3) 단계로 나누어져있는 categorical 변수이다. 나머지 39개의 변수 중 1번 변수인 building_id는 instance를 구분 하기위한 id로 통계적 의미가 없는 변수로 최종 데이터셋에서는 제거해준다. 데이터 전처리 이전 결측를 확인해 주었다.

1.2 결측치 확인

```
> table(is.na (origindata_input))
   FALSE
10163439
> table(is.na(origindata_target))
 FALSE
521202
```

주어진 데이터를 처리하기전 결측치를 확인해, 결측치가 있는 행을 데이터셋에서 제거해준다. 다행히 주어진 데이터 셋에는 결측치가 존재하지 않았다.

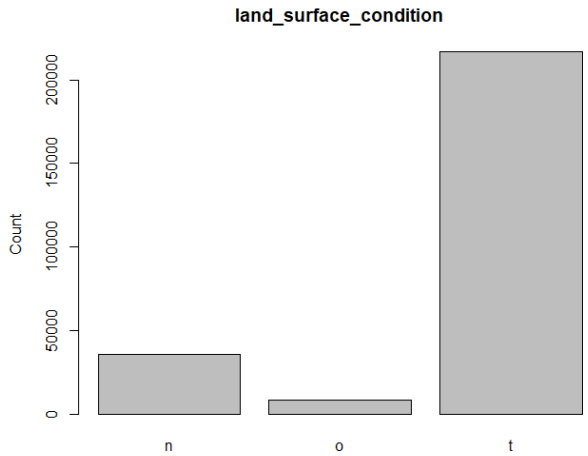
1.3 Numeric이 아닌 변수

Origindata_input 으로 받은 기본 입력변수 39 개 종류들을 확인해보았다. 우선 다른 변수들과 달리 int 형 변수가 아닌 chr 형 변수로 이루어진 (9~15, 27 번)변수는 string 형 categorical 변수이다.

Colum_num	Var	Colum_num	Var
9	land_surface_condition	13	other_floor_type
10	foundation_type	14	position
11	roof_type	15	plan_configuration
12	ground_floor_type	27	legal_ownership_status

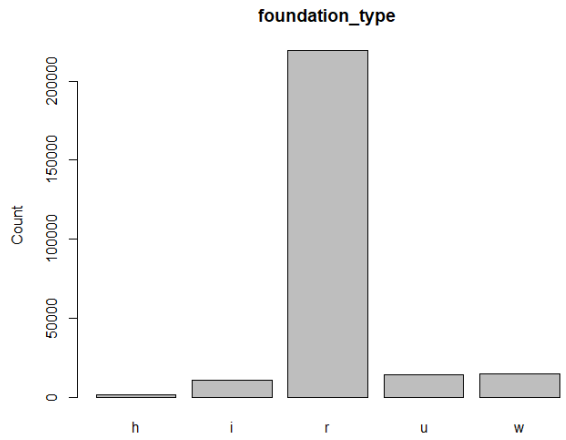
총 8 개의 변수들의 Barchart 는 아래와 같다.

1.3.1 land_surface_condition



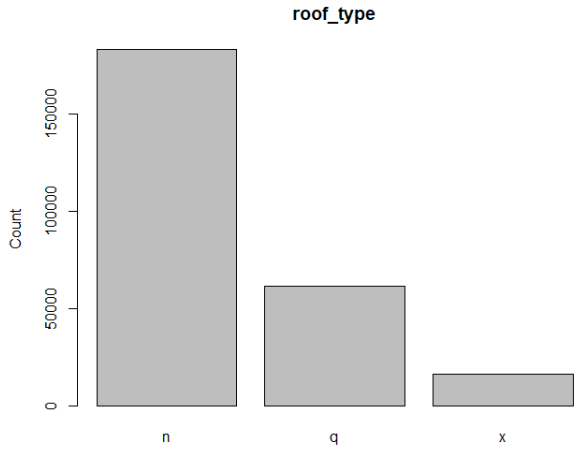
land_surface_condition 의 factor 는 총 3 개 (n,o,t)로 이루어져 있다. 이중 t 의 개수가 20 만개 이상으로 가장 많으며, 그 다음으론 n 과 o 가 뒤를 따른다.

1.3.2 foundation_type



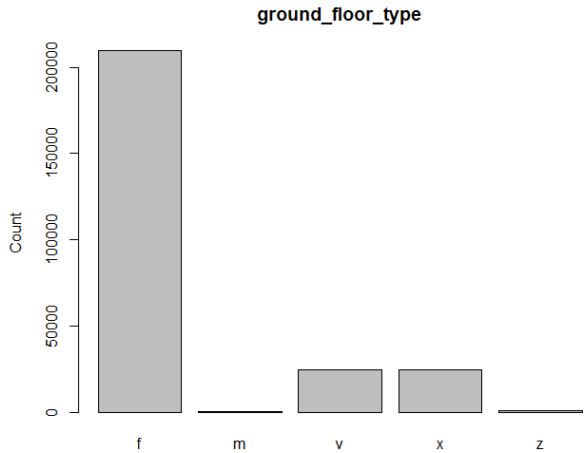
foundation_type 변수는 총 5가지의 factor로 이루어져있다. (h,i,r,u,w) 5가지 factor 중 r factor가 20만개 이상으로 가장 많고 h 가 가장 적은 것을 알 수 있다.

1.3.3 roof_type



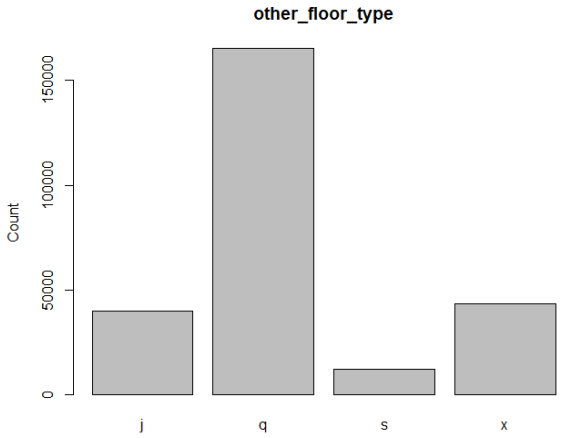
roof_type 변수는 총 3가지의 factor로 이루어져있다. (n,q,x) 중 n factor가 15만개를 넘기며 가장 나타나는 빈도가 높은 factor이고 그 다음으론 q factor가 5만번 이상 나타나는 두번째로 빈도가 높은 factor이다.

1.3.4 ground_floor_type



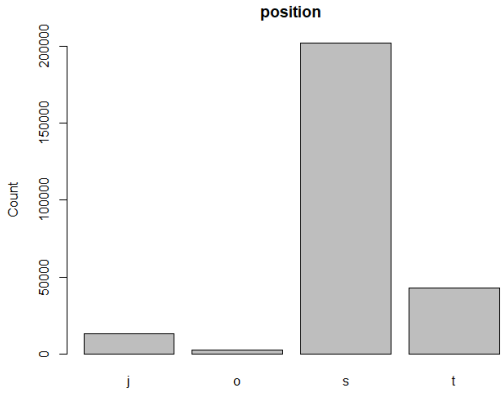
ground_floor_type 변수는 f,m,v,x,z 총 5가지의 factor로 이루어져 있다. 이중 f factor가 20만번 이상 나타나는 가장 출현 빈도가 높은 factor 이다.

1.3.5 other_floor_type



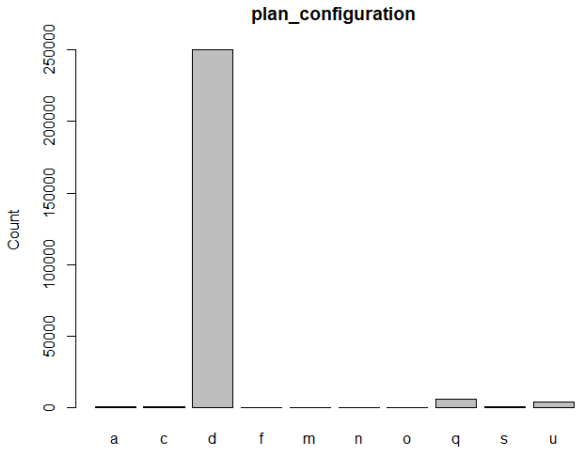
other_floor_type는 j,q,s,x 4가지 factor로 이루어져 있다. 이중 j factor가 가장 출현 빈도가 높은 factor 이다.

1.3.6 position



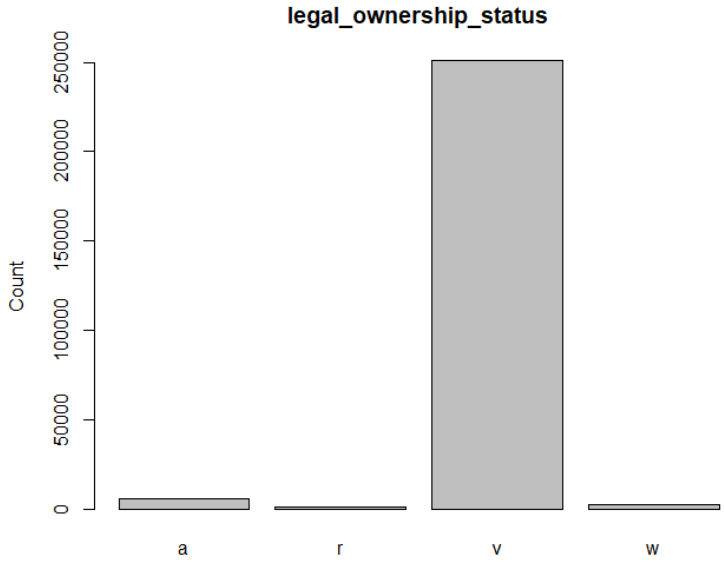
position 변수는 j,o,s,t 4가지의 factor로 이루어져 있다. 이중 s factor가 가장 출현 빈도가 높다.

1.3.7 plan_configuration



plan_configuration 변수는 a,c,d,f,m,n,o,q,s,u 총 10가지의 factor로 이루어져있다. 이중 d factor가 25만번이상 나타난다.

1.3.8 legal_ownership_status



legal_owner_ship 변수는 a,r,v,w 총 4가지의 factor로 이루어져있다. v facor의 변수가 가장 출현 빈도가 높다.

[Q2] Dummy Variable 생성

2.1 One hot encoding이 필요한 변수

Q1에서 string이루어진 categorical 변수 8개를 함수 class.ind 를 이용하여 one hot encoding 을 진행해 주었다.

```
origindata_input_cat_1hot<-matrix(0,nrow=origindata_instance)
for (i in 1:8){
  t<-class.ind(origindata_input_cat[,i])
  colnames(t) <- paste(colnames(origindata_input_cat[i]), colnames(t), sep = "_")
  origindata_input_cat_1hot<-cbind(origindata_input_cat_1hot,t)
}
```

One hot encoding을 통해 만들어진 dummy variable들의 이름은 원래의 variable 명 + _ + factor 명 의 규칙으로 구현하였다. 아래는 8개의 변수 중 대표로 land_surface_condition 변수의 one hot encoding 후 생성된 dummy variable의 예시이다.

land_surface_condition	n	o	t
land_surface_condition_n	1	0	0
land_surface_condition_o	0	1	0
land_surface_condition_t	0	0	1

2.2 최종 데이터셋 제작

```
origindata_target_1hot<-class.ind(origindata_target[,2])
colnames(origindata_target_1hot) <- paste("damagae_grade", colnames(origindata_target_1hot), sep
= "_")
#Scale the numeric data and make Final data
```

```
origindata_input_num<-scale(origindata_input_num,center=TRUE, scale=TRUE)
finaldata<-data.frame(origindata_input_num, origindata_input_bin, origindata_input_cat_1hot[,-1],
origindata_target_1hot)
```

Target 데이터 또한 categorical 변수이므로 class.ind 함수를 이용하여 one hot encoding을 진행한다. 이후 numeric 변수 중 binary 변수가 아닌 변수들을 정규화 시킨후, finaldata라는 이름의 데이터프레임으로 묶어주었다.

2.3 Training Set, Validation Set, Test Set 분할

```
set.seed(123456)
trn_idx <- sample(1:origindata_instance, 150000)
earth_trn<-finaldata[trn_idx,]
earth_temp<-finaldata[-trn_idx,]
set.seed(123456)
val_idx <-sample(1:nrow(earth_temp),50000)
earth_val<-earth_temp[val_idx,]
earth_tst<-earth_temp[-val_idx,]
trn_target <- earth_trn[,c(69,70,71)]
trn_input <- earth_trn[,-c(69,70,71)]
val_target <- earth_val[,c(69,70,71)]
val_input <- earth_val[,-c(69,70,71)]
tst_target <- earth_tst[,c(69,70,71)]
tst_input <- earth_tst[,-c(69,70,71)]
```

총 260601개의 instance 중 150000개는 Training Set, 50000개는 Valdation Set, 나머지 60601개는 Test_Set으로 설정해 주었다. 이후 문제를 위해 각각의 Set을 target과 input으로 나누어 설정해주었다.

[Q3] Grid Search를 통한 최적의 조합 찾기

3.1Hyperparameter 조합을 통해 성능 평가 기록

```
nH <- seq(from=1,to=13,by=2)
max_iter <- seq(from=40, to=100, by=30)
q3_mat <- matrix(0,length(nH)*length(max_iter),4)
colnames(q3_mat) <- c("Hidden Nodes", "Max Iteration", "ACC", "BCR")
start <- proc.time()
n<-1
for (i in 1:length(nH)) {
  cat("Training ANN: the number of hidden nodes:", nH[i], "\n")
  for (j in 1:length(max_iter)) {
    tmp_nnet <- nnet(trn_input,trn_target, size = nH[i], decay = 5e-4, maxit = max_iter[j])
    prey <- predict(tmp_nnet, val_input)
    q3_mat[n,1:2] <- c(nH[i],max_iter[j])
    q3_mat[n,3:4] <- perf_eval_multi2(max.col(val_target), max.col(prey))
    n <- n+1
  }
}
proc.time() - start
```

Grid Search 를 통한 최적의 Hyperparameter의 조합을 찾기 위해 히든노드의 수는 (1,3,5,7,9,11,13)의 수로 설정하였다. 컴퓨팅 능력의 제한이 있어 극단적으로 많은 수의 히든노드의 수를 가정하지 못하였다. Iteration의 수 또한 (40,70,100) 총 3가지의 케이스를 가정하여 최적의 조합을 도출해냈다.

```
 #ACC 기준의 성능 평가
q3_ordered_ACC <- q3_mat[order(q3_mat[,3], decreasing = TRUE),]
colnames(q3_ordered_ACC) <- c("Hidden Nodes", "maxit", "ACC", "BCR")
q3_ordered_ACC
#BCR 기준의 성능 평가
q3_ordered_BCR <- q3_mat[order(q3_mat[,4], decreasing = TRUE),]
colnames(q3_ordered_BCR) <- c("Hidden Nodes", "maxit", "ACC", "BCR")
q3_ordered_BCR
```

3.2 최적의 Hyperparameter 조합 도출

총 21 가지의 조합의 Accuracy(ACC)와 Balancec Correction Rate(BCR)를 기록한 q3_mat 을 ACC, BCR 기준으로 정렬해주었다.

```
> q3_ordered_ACC
Hidden Nodes maxit     ACC       BCR
 [1,]           13   100 0.64900 0.4758309
 [2,]            9   100 0.64818 0.4883394
 [3,]            7    70 0.64254 0.4729332
 [4,]            9    40 0.63592 0.4618172
 [5,]           13    40 0.63032 0.4710055
 [6,]            5    40 0.61090 0.4348136
 [7,]            7   100 0.60348 0.0000000
 [8,]            3   100 0.60176 0.4769195
 [9,]            3    40 0.59774 0.3989270
[10,]            5   100 0.59746 0.4762234
[11,]           11    40 0.59006 0.0000000
[12,]            3    70 0.58560 0.0000000
[13,]            1    40 0.57034 0.0000000
[14,]            1    70 0.57034 0.0000000
[15,]            1   100 0.57034 0.0000000
[16,]            5    70 0.57034 0.0000000
[17,]            9    70 0.57034 0.0000000
[18,]           11   100 0.57034 0.0000000
[19,]           13    70 0.57034 0.0000000
[20,]            7    40 0.57030 0.0000000
[21,]           11    70 0.52278 0.0000000
```

```
> q3_ordered_BCR
      Hidden Nodes maxit     ACC       BCR
 [1,]            9   100 0.64818 0.4883394
 [2,]            3   100 0.60176 0.4769195
 [3,]            5   100 0.59746 0.4762234
 [4,]           13   100 0.64900 0.4758309
 [5,]            7    70 0.64254 0.4729332
 [6,]           13    40 0.63032 0.4710055
 [7,]            9    40 0.63592 0.4618172
 [8,]            5    40 0.61090 0.4348136
 [9,]            3    40 0.59774 0.3989270
[10,]            1    40 0.57034 0.0000000
[11,]            1    70 0.57034 0.0000000
[12,]            1   100 0.57034 0.0000000
[13,]            3    70 0.58560 0.0000000
[14,]            5    70 0.57034 0.0000000
[15,]            7    40 0.57030 0.0000000
[16,]            7   100 0.60348 0.0000000
[17,]            9    70 0.57034 0.0000000
[18,]           11    40 0.59006 0.0000000
[19,]           11    70 0.52278 0.0000000
[20,]           11   100 0.57034 0.0000000
[21,]           13    70 0.57034 0.0000000
```

이 때 `q3_ordered_ACC` 의 제일 상위 값을 기록한 Hidden Nodes 13개, Max Iteration 100회의 ACC 성능과 `q3_ordered_BCR` 의 제일 상위 값을 기록한 기록한 Hidden Nodes 13개, Max Iteration 100회의 ACC 성능이 0.0082 밖에 차이가 나지 않고, BCR 은 약 0.013 차이나는 것을 알 수 있다. ACC의 성능차이가 BCR 보다 적으므로 후자의 hyper parameter의 조합을 택하는 것이 합리적인 기준이라고 할 수 있다. 그 결과 Hidden Nodes의 수가 9개 Max iteration의 수가 100회일 때가 최적의 hyperparameter라는 것을 알 수 있다.

3.2 가장 우수한 조합과 가장 열등한 조합의 차이

`q3_ordered_ACC`와 `q3_ordered_BCR`를 통해일 통해 가장 열등한 조합은 Hidden Nodes가 11개이고 Max iteration이 70인 경우임을 알 수 있다. 여기서 BCR이 0을 기록한 이유는 다음과 같다. BCR은 True Positive 혹은 False

Negative 둘중 하나라고 0을 갖게 되면 0의 값을 가질 수 있기 때문에 BCR은 0을 기록 할 수 있습니다.

그럼 가장 우수한 조합과 열등한 조합의 Accuracy와 Balanced Correction Rate의 차이는 다음과 같습니다.

ΔACC = 0.64818-0.52778=0.1204, ΔBCR=0.4883394-0=0.4883394

ACC의 차이는 약 0.1204로 대략 12%정도의 성능 차이를 보여주었다. BCR를 통해 측정한 차이는 약 50%의 가까운 차이로 성능 편차가 매우 큼을 알 수 있다.

[Q4] Rang 옵션 조절

4.1 rang 옵션 조절 후 최적의 hyperparameter 조합 찾기

Rang 옵션의 default 값은 0.7 로, rang의 가능한 케이스를 총 (0.1, 0.25, 0.4, 0.55, 0.7, 0.85) 6가지로 나누어 성능을 측정하였다. 나머지 Hyperparameter 인 number of hidden node나 max iteration은 Q3에서 구한 최적값(9,100)으로 설정하여 계산을 진행하였다.

```
rang <- seq(0.1, 0.9, 0.15)
q4_mat <- matrix(0,length(rang),3)
colnames(q4_mat) <- c("rang", "ACC", "BCR")
start <- proc.time()
n <- 1
for (i in 1:length(rang)){
  rang_nnet <- nnet(trn_input,trn_target, size =q3_ordered_BCR[1,1] , rang = rang[i], decay = 5e-
4, maxit =q3_ordered_BCR[1,2] )
  rang_prey <- predict(rang_nnet, val_input)
  q4_mat[n,1] <- rang[i]
  q4_mat[n,2:3] <- perf_eval_multi2(max.col(val_target), max.col(rang_prey))
  n <- n+1

}
proc.time() - start
q4_mat
q4_mat_ACC<-q4_mat[order(q4_mat[,2], decreasing = TRUE),]
q4_mat_BCR<-q4_mat[order(q4_mat[,3], decreasing = TRUE),]
q4_mat_ACC[1,]
q4_mat_BCR[1,]
```

결과를 정리한 q4_mat은 아래와 같다.

```
> q4_mat
     rang     ACC       BCR
[1,] 0.10 0.62006 0.47007966
[2,] 0.25 0.57088 0.01720788
[3,] 0.40 0.64478 0.50548005
[4,] 0.55 0.63982 0.50345576
[5,] 0.70 0.62292 0.43275679
[6,] 0.85 0.63692 0.47755969
```

이를 ACC 기준, BCR 기준으로 정리한 값의 가장 우월한 값을 도출해 보았다.

```
> q4_mat_ACC[1,]
   rang     ACC     BCR
0.40000 0.64478 0.50548
> q4_mat_BCR[1,]
   rang     ACC     BCR
0.40000 0.64478 0.50548
```

q4_mat_ACC나, q4_mat_BCR 의 가장 최적의 rang 값은 0.4로 동일하다는 것을 알 수 있다. 이를 통해 최적의 hyperparameter조합은 다음과 같다는 것을 알 수 있다.

Hidden Nodes	max_it	rang
9	100	0.4

[Q5] ANN 모델 구조 확정과 반복 학습 수행

```
#training set과 validation set 결합
intg_target <- rbind(trn_target,val_target)
intg_input <- rbind(trn_input,val_input)
q5_mat <- matrix(0,10,3)
colnames(q5_mat) <- c("No.", "ACC", "BCR")
start <- proc.time()
n <- 1
for (i in 1:10){
  final_nnet <- nnet(intg_input,intg_target, size = q3_ordered_BCR[1,1] , rang = q4_mat_ACC[1,1],
decay = 5e-4, maxit =q3_ordered_BCR[1,2] )
  final_prey <- predict(final_nnet, tst_input)
  q5_mat[n,1] <- i
  q5_mat[n,2:3] <- perf_eval_multi2(max.col(tst_target), max.col(final_prey))
  n <- n+1
}
proc.time() - start
q5_mat
```

총 10번의 반복으로 Q3,Q4에서 구한 최적의 hyperparameter를 이용하여 ANN 모델의 성능을 평가한다. 이를 위해 trn_data와 val_data를 합쳐 intg_data로 만들어 이를 학습에 이용한다. 총 10번 반복에 대한 성능은 아래와 같다.

```
> q5_mat
      No.       ACC         BCR
 [1,]   1 0.6347915 0.441039192
 [2,]   2 0.6501543 0.479426589
 [3,]   3 0.6283230 0.498049424
 [4,]   4 0.5712117 0.000000000
 [5,]   5 0.6356331 0.495928799
 [6,]   6 0.6457484 0.501056766
 [7,]   7 0.5711622 0.000000000
 [8,]   8 0.6338509 0.494918885
 [9,]   9 0.6355671 0.417728017
[10,]  10 0.5712447 0.009432884
```

10번의 반복 동한 구한 ACC와 BCR의 평균, 분산, 표준편차를 구하기 위하여 q5_total_mat을 만들어 정리하였다.

```
q5_total_mat<-matrix(0,2,3)
colnames(q5_total_mat)<-c("Mean","Varaiance","Standard Deviation")
rownames(q5_total_mat)<-c("ACC","BCR")
q5_total_mat[1,1]<-mean(q5_mat[,2])
q5_total_mat[1,2]<-var(q5_mat[,2])
q5_total_mat[1,3]<-sd(q5_mat[,2])
q5_total_mat[2,1]<-mean(q5_mat[,3])
q5_total_mat[2,2]<-var(q5_mat[,3])
q5_total_mat[2,3]<-sd(q5_mat[,3])
q5_total_mat
```

q5_total_mat은 아래와 같다.

```
> q5_total_mat
        Mean  Varaiance Standard Deviation
ACC 0.6177687 0.00107018         0.0327136
BCR 0.3337581 0.05277865         0.2297361
```

ACC의 평균은 0.6177687이며, 분산은 0.00107018, 표준편차는 0.032727136으로 10번의 반복동안 큰 차이가 없다는 것을 알 수 있다. 이와 다르게 BCR의 평균은 0.3337581, 분산은 0.05277865, 표준편차는 0.2297361로 ACC 보다 10번의 시행간의 차이가 크다는 것을 알 수 있다. 이러한 이유는 BCR에서는 4번째 시행, 7

번째 시행에서의 BCR의 값이 0으로 나와 그 차이가 심하게 났다는 것을 알 수 있다.

[Q6] GA를 이용한 ANN 모델의 변수 찾기

6.1 fitness function 정의하기

GA를 돌리기 앞서 각 세대별 성능의 측정 기준인 fitness function을 정의해주어야 한다. Fitness function은 ACC를 기준으로 삼아 정의하였다. Fitness function 안에서 Training model 안에서 이용되는 ANN 기법은 Q5에서 구했던 최적의 hyperparameter 조합으로 설정해주었다. (maxit는 바꿈: 이유 후술)

```
fit_func <- function(string){
  sel_var_idx <- which(string == 1)
  # Use variables whose gene value is 1
  sel_x <- x[, sel_var_idx]
  # Training the model
  GA_NN <- nnet(sel_x,y, size = q3_ordered_BCR[1,1] , rang = q4_mat_ACC[1,1], decay = 5e-4, maxit
=10 ) #컴퓨팅 능력 제한
  GA_prey <- predict(GA_NN, tst_input[, sel_var_idx])
  return(perf_eval_multi2(max.col(tst_target), max.col(GA_prey))[1])
}
```

GA에 사용될 데이터는 Q5와 동일하게 training set과 validation set이 합쳐진 intergreted set을 이용한다. 지난 Assignment 3에서 찾았던 GA의 최적의 hyperparameter 조건인 popsize=25. pcrossover=0.7, pmutation=0.01로 설정하였다. Seed는 123*i로 설정하여 1회마다 바뀌게 하여 초기 chromosome 값을 랜덤하게 지정해주었다. 매 반복마다 결정되는 변수는 q6_mat에 기록하도록 설정하였다.

```
x <- intg_input
y <- intg_target
q6_mat <- matrix(0,3,2)
colnames(q6_mat) <- c("No.", "variable selected num")
for (i in 1:3){
  GA_q3 <- ga(type = "binary", fitness = fit_func, nBits = ncol(x),
              names = colnames(x), popSize = 25, pcrossover = 0.7,
              pmutation = 0.01, maxiter = 10, elitism = 2, seed = 123*i)

  q6_mat[i,2] <- paste(which(GA_q3@solution[which.min(rowSums(GA_q3@solution)),] == 1),
collapse=",")
  q6_mat[i,1]<-i
  q6_mat
  }
q6_mat
```

앞서 fitness function과 GA 구현 부분에서 maxiter의 사이즈를 임의로 10으로 줄였다. 그 이유는 maxit를 둘다 이전의 이용했던 값인 100으로 설정하면 컴퓨팅 능력의 한계로 총 계산시간이 너무 길어져 현실적으로 구하기 힘들어 각 각 10으로 줄여 계산량을 1/100로 줄여 보다 빠른 결과를 도출해내었다. 3번의 반복으로 구한 선택된 variable들의 index 값들을 q6_mat에다 저장하였고 이는 아래와 같다.

```
> q6_mat
     No. variable selected num
[1,] "1"
"1,3,4,5,6,14,15,16,17,18,19,20,21,22,23,25,26,27,31,33,35,36,38,40,41,42,46,48,51,56,60,61,62,64
,65,67"
[2,] "2"
"1,3,11,13,14,16,17,19,20,21,23,24,27,30,31,32,33,34,35,37,38,39,40,41,42,44,46,47,49,53,54,59,60
,63,64,65,66,67,68"
[3,] "3"
"1,2,4,6,8,10,12,13,16,17,18,19,20,25,26,29,33,34,36,37,39,40,41,43,44,46,52,54,56,57,58,59,61,62
,63,64,65"
```

첫번째 iteration에서는 68개의 입력변수 중 36개의 변수가 선택되었다. 두번째 iteration에서는 39개의 변수가 선택되었고, 마지막 iteration에서는 37개의 변수가 선택되었다. 매 iteration 마다 선택되는 변수는 조금의 차이

를 보였다. 이중 세번의 iteration 중에서 두 번이상 선택된 변수들을 따로 구해주었다.

```
temp_q6<-matrix(0,1,68)
temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[1,2],split=","))))]<-
temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[1,2],split=","))))]+1
temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[2,2],split=","))))]<-
temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[2,2],split=","))))]+1
temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[3,2],split=","))))]<-
temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[3,2],split=","))))]+1
ga_chosen_idx<-which(temp_q6>=2)
ga_chosen_idx
```

총 68개의 column을 가진 빈 matrix를 생성한 후 각 iteration에서 찾은 선택된 입력변수의 index에 해당하는 column에 1씩 더해주었다. 그후 값이 2이상인 부분의 index만을 추출하여 최종 선택 입력변수들을 정리하였다. 최종 선택된 index는 다음과 같다.

```
> temp_q6<-matrix(0,1,68)
> temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[1,2],split=","))))]<-
temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[1,2],split=","))))]+1
> temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[2,2],split=","))))]<-
temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[2,2],split=","))))]+1
> temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[3,2],split=","))))]<-
temp_q6[,c(as.numeric(unlist(strsplit(q6_mat[3,2],split=","))))]+1
> ga_chosen_idx<-which(temp_q6>=2)
> ga_chosen_idx
 [1]  1  3  4  6 13 14 16 17 18 19 20 21 23 25 26 27 31 33 34 35 36 37 38 39 40 41 42 44 46 54 56
59 60 61 62 63 64 65 67
```

Colum_num	Var	Colum_num	Var
1	geo_level_1_id	36	foundation_type_r
3	geo_level_3_id	37	foundation_type_u
4	count_floors_pre_eq	38	foundation_type_w
6	area_percentage	39	roof_type_n
13	has_superstructure_cement_mortar_brick	40	roof_type_q
14	has_superstructure_timber	41	roof_type_x
16	has_superstructure_rc_non_engineered	42	ground_floor_type_f
17	has_superstructure_rc_engineered	44	ground_floor_type_v
18	has_superstructure_other	46	ground_floor_type_z
19	count_families	54	position_t
20	has_secondary_use	56	plan_configuration_c
21	has_secondary_use_agriculture	59	plan_configuration_m
23	has_secondary_use_rental	60	plan_configuration_n
25	has_secondary_use_school	61	plan_configuration_o
26	has_secondary_use_industry	62	plan_configuration_q
27	has_secondary_use_health_post	63	plan_configuration_s
31	land_surface_condition_n	64	plan_configuration_u
33	land_surface_condition_t	65	legal_ownership_status_a
34	foundation_type_h	67	legal_ownership_status_v
35	foundation_type_i		

[Q7] GA를 이용해 선택한 변수로 ANN 시행

Q6에서 구한 변수들과 Q4,Q5에서 구한 최적의 hyperparameter 조건을 바탕으로 최종 ANN 모델의 Accuracy와 Balanced Correction Rate을 구해주었다.

```
intg_input_q7 <- intg_input[,ga_chosen_idx]
q7_mat <- matrix(0,10,3)
colnames(q7_mat) <- c("No.", "ACC", "BCR")
start <- proc.time()
n <- 1
for (i in 1:10){
  final_nnet <- nnet(intg_input_q7,intg_target, size = q3_ordered_BCR[1,1] , rang =
q4_mat_ACC[1,1], decay = 5e-4, maxit =q3_ordered_BCR[1,2] )
  final_prey <- predict(final_nnet, tst_input[,ga_chosen_idx])
  q7_mat[n,1] <- i
  q7_mat[n,2:3] <- perf_eval_multi2(max.col(tst_target), max.col(final_prey))
  n <- n+1
}
proc.time() - start
q7_mat
```

이에 대한 결과 q7_mat을 BCR이 큰 순서로 정리하여 나열하면 아래와 같은 표를 얻을 수 있다.

```
q7_mat <- q7_mat[order(q7_mat[,3], decreasing = T),]
> q7_mat
      No.       ACC       BCR
 [1,]   5 0.6194452 0.4538785
 [2,]   6 0.6129272 0.4396119
 [3,]   7 0.6301546 0.4368752
 [4,]   2 0.6457649 0.4326359
 [5,]   8 0.6257983 0.4228058
 [6,]   1 0.6143628 0.4163108
 [7,]   9 0.6429267 0.4118560
 [8,]   3 0.6217554 0.3932187
 [9,]   4 0.5712117 0.0000000
[10,]  10 0.5712117 0.0000000
```

q7_mat 속 ACC와 BCR의 값들의 평균과 분산 표준편차를 구하여 정리해보았다.

```
q7_total_mat<-matrix(0,2,3)
colnames(q7_total_mat)<-c("Mean","Varaiance","Standard Deviation")
rownames(q7_total_mat)<-c("ACC","BCR")
q7_total_mat[1,1]<-mean(q7_mat[,2])
q7_total_mat[1,2]<-var(q7_mat[,2])
q7_total_mat[1,3]<-sd(q7_mat[,2])
q7_total_mat[2,1]<-mean(q7_mat[,3])
q7_total_mat[2,2]<-var(q7_mat[,3])
q7_total_mat[2,3]<-sd(q7_mat[,3])
q7_total_mat
```

이에 대한 결과는 아래와 같다.

```
> q7_total_mat
         Mean    Varaiance Standard Deviation
ACC 0.6155558 0.0006638316         0.02576493
BCR 0.3407193 0.0325252826         0.18034767
```

ACC의 평균은 0.61555580이며, 분산은 0.0006638316, 표준편차는 0.002574693으로 10번의 반복동안 큰 차이가 없다는 것을 알 수 있다. 이와 다르게 BCR의 평균은 0.3407193, 분산은 0.0325252826, 표준편차는 0.18034767로 ACC 보다 10번의 시행간의 차이가 크다는 것을 알 수 있다. 이러한 이유는 BCR에서는 9번째 시행, 10번째 시행에서의 BCR의 값이 0으로 나와 그 차이가 심하게 났다는 것을 알 수 있다.

Q5의 결과와 평균을 비교해 보면 아래와 같다.

	ACC	BCR
Q5	0.6177687	0.3337581
Q7	0.6155558	0.3407193

위의 표에서 알 수 있듯 Q5와 Q7의 ACC 와 BCR 값은 큰 차이가 없다는 것을 알 수 있다. GA를 통해 68개의 변수에서 39개의 변수를 줄인 이후에 성능차이가 큰 차이가 없는 것을 통해 Q6와 Q7에서 진행한 차원축소법이 의미가 있었다고 할 수 있다.

[Q8] Decision Tree의 ACC, BCR

8.1 최적의 hyperparameter 찾기

Hyperarameter를 조정 전 앞 Q1~7까지 썼던 데이터들은 종속변수까지 one hot encoding이 되어 있어 이를 다시 하나의 변수로 묶어주는 과정을 진행하여 q8_data,q8_tst_data를 제작한다.

```
min_criterion = c(0.875,0.9, 0.95, 0.975, 0.99)
min_split = c(3000,5000,10000)
max_depth = c(0,1,3,5)
intg_target_q8<-as.numeric(toupper(substr(names(intg_target), nchar(names(intg_target))-0,
nchar(names(intg_target)))[max.col(intg_target)]))
tst_target_q8<-as.numeric(toupper(substr(names(tst_target), nchar(names(tst_target))-0,
nchar(names(tst_target)))[max.col(tst_target)]))
q8_data<-cbind(intg_input,damagae_grade=intg_target_q8)
q8_tst_data<-cbind(tst_input,damagae_grade=tst_target_q8)
q8_mat <- matrix(0,length(min_criterion)*length(min_split)*length(max_depth),5)
colnames(q8_mat) <- c("min_criterion","min_split","max_depth","ACC", "BCR")
iter_cnt = 1
l<-(1:3)
for (i in 1:length(min_criterion)){
  for ( j in 1:length(min_split)){
    for ( k in 1:length(max_depth)){

      tmp_control = ctree_control(mincriterion = min_criterion[i], minsplit = min_split[j],
maxdepth = max_depth[k])
      tmp_tree <- ctree(damagae_grade ~ ., data = q8_data, controls = tmp_control)
      tmp_tree_val_prediction <- predict(tmp_tree, newdata = q8_tst_data)
      tmp_tree_val_response <- treeresponse(tmp_tree, newdata = q8_tst_data)
      round(tmp_tree_val_prediction)
      # Confusion matrix
      cfm <-  table(factor(round(tmp_tree_val_prediction),l), factor(q8_tst_data$damagae_grade,l))
      # parameters
      q8_mat[iter_cnt,1] = min_criterion[i]
      q8_mat[iter_cnt,2] = min_split[j]
      q8_mat[iter_cnt,3] = max_depth[k]
      # Performances from the confusion matrix
      q8_mat[iter_cnt,4:5] = perf_eval_multi(cfm)
      iter_cnt <-iter_cnt+1
    }
  }
}
```

지난 Assignment 4에서 설정한 hyper parameter의 조합중 min_criterion 과 max_depth 는 동일하게 설정해주었고, 데이터가 20만개로 학습해 min_split은 3000,5000,10000으로 설정하였다. 각 조합에 대한 성능(ACC, BCR)을 q8_mat 저장하였고 그결과는 아래와 같다.

```
> q8_mat <- q8_mat[order(q8_mat[,5], decreasing = T),]
> q8_mat
     min_criterion min_split max_depth       ACC       BCR
 [1,]         0.875      3000         5 0.6015412 0.6686452
 [2,]         0.875      5000         5 0.6015412 0.6686452
 [3,]         0.900      3000         5 0.6015412 0.6686452
 [4,]         0.900      5000         5 0.6015412 0.6686452
```

```
 [5,]        0.950      3000         5 0.6015412 0.6686452
 [6,]        0.950      5000         5 0.6015412 0.6686452
 [7,]        0.975      3000         5 0.6015412 0.6686452
 [8,]        0.975      5000         5 0.6015412 0.6686452
 [9,]        0.990      3000         5 0.6015412 0.6686452
[10,]        0.990      5000         5 0.6015412 0.6686452
[11,]        0.875     10000         5 0.6005841 0.6609718
[12,]        0.900     10000         5 0.6005841 0.6609718
[13,]        0.950     10000         5 0.6005841 0.6609718
[14,]        0.975     10000         5 0.6005841 0.6609718
[15,]        0.990     10000         5 0.6005841 0.6609718
[16,]        0.875      3000         0 0.6352040 0.6430338
[17,]        0.900      3000         0 0.6352040 0.6430338
[18,]        0.950      3000         0 0.6352040 0.6430338
[19,]        0.975      3000         0 0.6347585 0.6385694
[20,]        0.990      3000         0 0.6347585 0.6385694
```

-총 60개의 데이터 중 상위 20개만 추출-

Balanced Correction Rate를 기준으로 보았을 때 min_criterion, min_split, max_depth는 (0.875,3000,5) 가 최적의 Hyperparameter의 조합인 것을 알 수 있다. 이때의 Test data의 대한 decision tree의 ACC 와 BCR은 0.601542,0.6686452 이다.

[Q9] MLR 진행, 최종 비교테이블 작성

9.1 Multinomial logistic regression 진행

앞서 Q8에서 제작한 Data를 바탕으로 Multinomial logistic regression을 시행한다.

```
# Multinominal logistic  regression
mlr_model <- lm(damagae_grade ~ ., data = q8_data)
mlr_prey <- predict(mlr_model, newdata = q8_tst_data)
q9_cfm <-  table(factor(round(mlr_prey),l), factor(q8_tst_data$damagae_grade,l))
perf_eval_multi(q9_cfm)
```

```
> perf_eval_multi(q9_cfm)
[1] 0.5796604 0.5595802
```

Multinomial logistic regression 의 결과 ACC 는 0.5796604, BCR 은 05595802 라는 것을 알 수있다. 이젠 지금까지 진행한 Q9, Q8, Q7, Q6 의 결과를 바탕으로 최종 비교 matrix 를 작성하였다. 이에 앞서 여러 개의 시험 데이터가 나온 Q7 과 Q5 에 대해선 BCR 을 기준으로 정렬 후 가장 높은 성능을 선택하였다. 그 이유는 Q1 에서 봤듯 주어진 데이터는 Imbalanced data 이기 때문에 Accuracy 보단 BCR 을 기준으로 선택하는 것이 통계적 유의성을 갖는다.

```
final_table<-matrix(0,2,4)
colnames(final_table) <- c("ANN","ANN_GA","Decision Tree","Multinomial Logistic Regression")
rownames(final_table) <-c("Accruacy", "BCR")
q7_mat <- q7_mat[order(q7_mat[,3], decreasing = TRUE),]
q5_mat <- q5_mat[order(q5_mat[,3], decreasing = TRUE),]
final_table[,4]<-perf_eval_multi(q9_cfm)
final_table[,3]<-q8_mat[1,c(4,5)]
final_table[,2]<-q7_mat[1,c(2,3)]
final_table[,1]<-q5_mat[1,c(2,3)]
final_table
```

최종 테이블은 아래와 같다.

	ANN	ANN_GA	Decision Tree	Multinomial Logistic Regression
Accuracy	0.6457484	0.6194452	0.6015412	0.5796604
BCR	0.5010568	0.4538785	0.6686452	0.5595802

해당 테이블을 통해 각 모델별 종합적인 성능을 비교해보면 ANN, Decision Tree, ANN_GA, Multinomal Logistic Regression 순으로 성능이 좋다. 물론 GA를 시행할 때 컴퓨팅능력의 한계로 Iteraton을 제한하여 정확한 결과를 알 수없었고, Balanced Correction Rate의 정의의 태생적 한계로 0값을 갖거나 NA 값을 가질 수 있어, 평균을 토대로한 성능비교가 어려워 해당 우선순위가 정확하다고 할 수는 없다.