Multivariate Data Analysis

Assignment #4

고려대학교 공과대학

산업경영공학부

2017170857 이우준

[Q1] 본인이 생각하기에 "예측 정확도"도 중요하지만 "예측 결과물에 대한 해석"이 매우 중요할 것으로 생각되는 분류 문제를 다루고 있는 데이터 셋을 1개 선정하고 선정 이유를 설명하시오. 데이터셋 탐색은 아래에서 제시된 Data Repository를 포함하여 여러 Repository를 검색해서 결정하시오. 보고서에는 데이터를 다운로드 할 수 있는 링크를 반드시 제공하시오.

의사결정나무는 다른 데이터 분석 기법과 비교해서 설명력이 강한 것이 특징이다. 설명력이 강하다는 것은 즉 예측 결과물에 대한 해석이 용이하다는 것이다. 이번 과제에 사용된 데이터 셋 "League Of Legends High elo Ranked Games(2020)" 을 선정한 이유도 여기에 있다.

많은 경쟁성 스포츠 (축구, 야구, 경마 등)는 승패에 여부 즉 승산에 관해 많은 사람들이 관심을 갖는다. 더 나아가 이러한 승산과 관련하여 돈을 거는 등 이와 관련된 시장은 큰 규모를 이루고 있다. 이는 E-Sports에서도 마찬가지이다. E-Sports 경기는 다른 스포츠경기와 달리 여러 게임 데이터를 실시간으로 쉽게 수집할 수 있고 이를 실시간으로 분석이 가능하다. 관객들에게 이러한 데이터를 바탕으로 승패의 확률을 알려주는 것과, 이를 관객들에게 이해가 가능하게 설명하는 것이 중요하다. 이러한 이유로 경기의 승패의 예측 정확도와, 이에대한 해석이 매우 중요하다고 생각하여 League Of Legends High elo Ranked Games(2020)" 데이터셋을 선정하였다.

Dataset1 : Breast Cancer Wisconsin (Diagnostic) Data Set
https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

Dataset2 : League Of Legends High elo Ranked Games(2020) (중 Challenger_Ranked_Games)
https://www.kaggle.com/gyejr95/league-of-legends-challenger-ranked-games2020

- Libirary 및 사전 함수 정의

```
library(tree)
library(party)
library(ROCR)
# Performance Evaluation Function
perf_eval <- 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(TPR, PRE, TNR, ACC, BCR, F1))
}
# 이상치 기준 분류 만들기 함수
boxplot_outliers <-function(target){
  valuelist <-boxplot(target)$stats
  return(c(valuelist[1,],valuelist[5,]))
}
# Performance table
perf_table_d1 <- matrix(0, nrow = 3, ncol = 7)
rownames(perf_table_d1) <- c("Non-Pruning","Post-Pruning", "Best_Tree_Tr&Val")
colnames(perf_table_d1) <- c("TPR", "Precision", "TNR", "Accuracy", "BCR", "F1-Measure","AUROC")
perf_table_d2 <- matrix(0, nrow = 3, ncol = 7)
rownames(perf_table_d2) <- c("Non-Pruning","Post-Pruning", "Best_Tree_Tr&Val")
colnames(perf_table_d2) <- c("TPR", "Precision", "TNR", "Accuracy", "BCR", "F1-Measure","AUROC")
```

의사결정나무 모델을 만들기 위하여 tree 패키지를 이용하였고, Pre-Pruning 단계에서 쓰일 Party 패키지와,

AUROC 계산을 위한 ROCR 패키지를 이용하였다. 또한 각 기법의 성능을 측정하기 위하여 perf_eval 함수를 정의하였고, 데이터 사전처리 과정중 필요한 이상치 제거를 위하여 box plot을 바탕으로 이상치 기준을 판별하는 함수를 정의 하였다. 또한 각 결과를 정리할 matrix도 정의하였다.

- Data Preprocessing

```
# Load data1 & Preprocessing
data1 <- read.csv("data.csv")
data1_input_idx <- c(3:32)
data1_target_idx <- 2
data1_input <- data1[,data1_input_idx]
data1_target <- as.factor(data1[,data1_target_idx])
data1_origin <- data.frame(data1_input, diagnosis = as.factor(data1_target))
#이상치 제거
data1_outliers <- matrix(0, nrow = 30, ncol = 2)
rownames(data1_outliers) <- names(data1_input)
colnames(data1_outliers) <- c("LCL", "UCL")
for(i in 1:30){
  data1_outliers[i,]<-boxplot_outliers(data1_origin[,i])
}
for (i in 1:30){
  data1_origin[,i]<-ifelse(data1_origin[,i] < data1_outliers[i,1] | data1_origin[,i] >
data1_outliers[i,2], NA,data1_origin[,i])
}
data1_origin<-na.omit(data1_origin)
ndata1 <- nrow(data1_origin)
#Training, Validating, Testing Set 분리
set.seed(123456)
data1_idx <- sample(1:ndata1, round(0.6*ndata1))
data1_trn <- data1_origin[data1_idx,]
data1_temp <-data1_origin[-data1_idx,]
set.seed(123456)
data1_idx <- sample(1:nrow(data1_temp), round(0.375*nrow(data1_temp)))
data1_val<- data1_temp[data1_idx,]
data1_tst<- data1_temp[-data1_idx,]
# Load data2 & Preprocessing
data2 <- read.csv("Challenger_Ranked_Games.csv")
data2_input_idx <- c(2,4:26,33:38,41:50)
data2_target_idx <- c(3)
data2_input <- data2[,data2_input_idx]
data2_input[,c(2:6)]<-data.frame(apply(data2_input[,c(2:6)], 2, as.factor))
data2_target <- as.factor(data2[,data2_target_idx])
data2_origin <- data.frame(data2_input, blueWins = data2_target)
#이상치 제거
data2_outliers <- matrix(0, nrow = 40, ncol = 2)
rownames(data2_outliers) <- names(data2_input)
colnames(data2_outliers) <- c("LCL", "UCL")
for(i in c(1,7:40)){
  data2_outliers[i,]<-boxplot_outliers(data2_origin[,i])
}
for (i in c(1,7:40)){
  data2_origin[,i]<-ifelse(data2_origin[,i] < data2_outliers[i,1] | data2_origin[,i] >
data2_outliers[i,2], NA,data2_origin[,i])
}
data2_origin<-na.omit(data2_origin)
ndata2 <- nrow(data2_origin)
```

Dataset 1 과 DataSet 2 를 불러와 사전처리를 해주었다. 여기서 의사결정나무는 Information based Algorithm 이여서 정규화나 이상치제거가 필요하진 않지만, Q6 에서 같은 데이터를 바탕으로 Logistic Regression 과 Decision Tree 의 성능을 비교하기 위하여 사전에 이상치 제거를 해주었다.

또한 DataSet 2 에서는 사전에 Input Data 를 처리하여 주었다. DataSet 2 에는 50 가지 데이터가 존재한다. 이중 3 번 항목 blueWins 는 종속변수로 분리해주었다. 이를 제외한 9 개의 데이터를 제외해 총 39 가지의 데이터를 input 데이터로 처리하였다. 사전 배제한 데이터 9 개는 ("gameID", "redWins", "redFirstBlood", "redFirstTower", "redFirstBaron", "redFirstDragon", "redFirstInhibitor", "redKills" , "redDeath") 이다. "gameID" 는 ID 값 자체로 어떠한 Value 를 가지고 있기 때문에 제거하였다. ("redWins", "redFirstBlood", "redFirstTower", "redFirstBaron", "redFirstDragon", "redFirstInhibitor", "redKills" , "redDeath")는 ("blueWins", "blueFirstBlood", "blueFirstTower", "blueFirstBaron", "blueFirstDragon", "blueFirstInhibitor", "blueKills", "blueDeath") 와 정확히 반대되는 binary 변수이기 때문에 다중공산성제거를 위해 input data 에 할당하지 않았다.

DataSet 1 과 2 모두 Training, Validation, Test 셋을 60 대 15 대 25 의 비율로 분할해주었다. 모델 구축에 있어서 Training 셋이 가장 중요하다고 생각해 가장 큰 60 의 비중을 주었고, 모델의 노이즈를 감소시키기 위해 나머지 40 의 비율을 20 대 20 으로 나누지 않고 Test 셋을 더 크게 비중을 준 이유는 Test 셋이 크면 클수록 관측되지 않은 데이터로 검증하여 모델이 얼마나 잘 일반화 되어있는지 알 수 있기 때문에 Test 셋의 비중을 25, Validation 셋의 비율을 15 로 정하였다.

[Q2] 실습시간에 사용한 "tree" package 를 사용하여 Classification Tree 를 학습한 결과물을 Plotting 하고 이에 대한 해석을 수행하시오. 또한 해당 Tree 를 pruning 을 수행하지 않은 상태에서 Test dataset 에 대한 분류 성능을 평가하시오.

- Plotting the Tree

```
#dataset1
# Training the tree
data1_tree_q2 <- tree(diagnosis ~ ., data1_trn)
summary(data1_tree_q2)
# Plot the tree
plot(data1_tree_q2)
text(data1_tree_q2, pretty = 1)
# Performance of the tree
data1_tree_q2_prediction<-predict(data1_tree_q2, newdata = data1_tst,type="class")
data1_tree_q2_cm <- table(data1_tst$diagnosis, data1_tree_q2_prediction)
data1_tree_q2_cm
perf_table_d1[1,c(1:6)]<-perf_eval(data1_tree_q2_cm)
data1_q2_pred <- prediction(as.numeric(data1_tree_q2_prediction),as.numeric(data1_tst$diagnosis))
perf_table_d1[1,7]<-as.numeric(performance(data1_q2_pred, measure = "auc")@y.values)
perf_table_d1
#dataset2
# Training the tree
data2_tree_q2 <- tree(blueWins ~ ., data2_trn)
summary(data2_tree_q2)
# Plot the tree
plot(data2_tree_q2)
text(data2_tree_q2, pretty = 1)
# Performance of the tree
data2_tree_q2_prediction<-predict(data2_tree_q2, newdata = data2_tst,type="class")
data2_tree_q2_cm <- table(data2_tst$blueWins, data2_tree_q2_prediction)
data2_tree_q2_cm
perf_table_d2[1,c(1:6)]<-perf_eval(data2_tree_q2_cm)
data2_q2_pred <- prediction(as.numeric(data2_tree_q2_prediction),as.numeric(data2_tst$blueWins))
perf_table_d2[1,7]<-as.numeric(performance(data2_q2_pred, measure = "auc")@y.values)
perf_table_d2
```

Preprocessing 을 끝낸 데이터셋을 이용하여 tree 패키지를 이용하여 Pruning 을 거치지 않은 의사결정나무를 만들었다. 다음 페이지 그림 1 을 보면 알 수 있듯 Data Set 1 로 만들어진 Tree 는 총 9 개의 Leaf node 로 이루어져 있고 perimeter_worst, texture_se, symmetry_se, texture_worst, radius_worst, concave.points_mean 6 개의 변수가 사용되었다.

DataSet 2 로 만들어진 Tree 는 6 개의 Leaf 로 이루어져있으며 blueTowerKills, blueDeath, blueKills, redInhibitorKills 총 4 개의 변수가 사용되었다. 두 Tree 의 성능은 perf_table_d1 과 perf_table_d2 에 기록하였다.

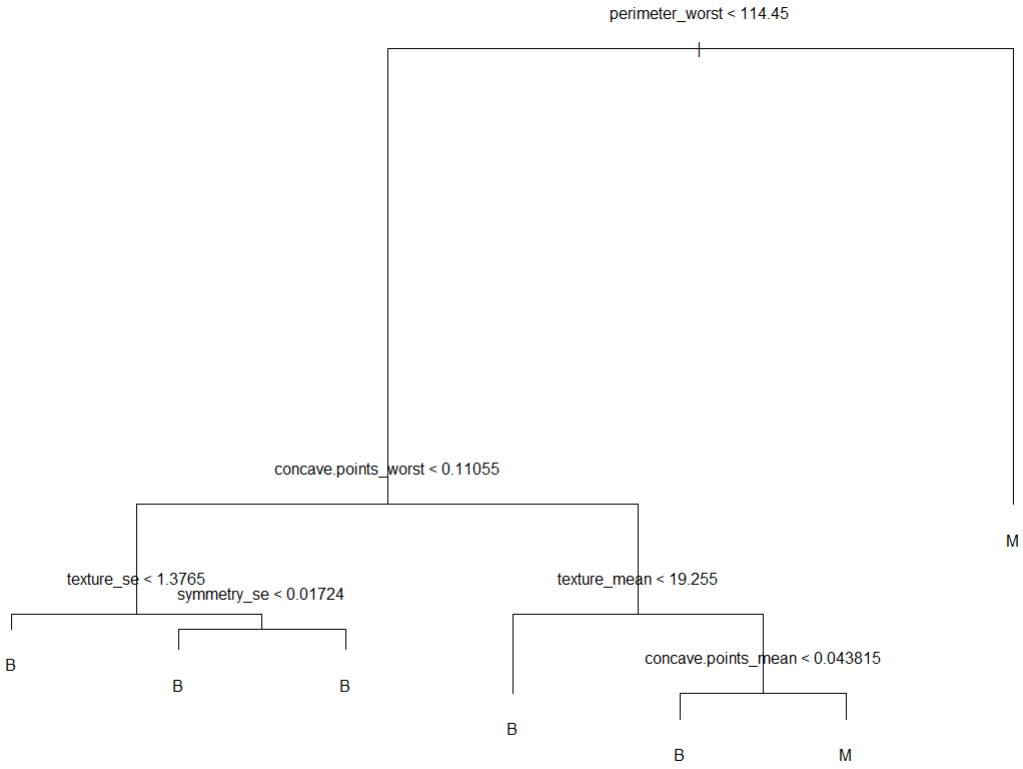

그림 1) DataSet 1로 만든 Pruning을 거치지 않은 Tree

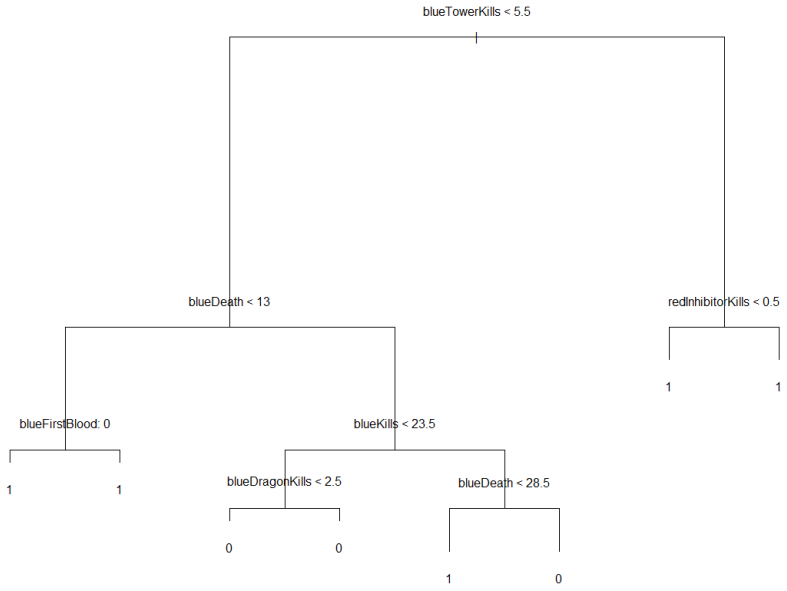

그림 2) DataSet 2로 만든 Pruning을 거치지 않은 Tree

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure	AUROC
Non-Pruning	0.818181818	0.857142857	0.961038961	0.929292929	0.886738183	0.837209302	0.88961039
Post-Pruning	0	0	0	0	0	0	0
Best_Tree_Tr&Val	0	0	0	0	0	0	0

표1) perf_table_d1

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure	AUROC
Non-Pruning	0.963636364	0.929824561	0.9	0.936842105	0.931274786	0.946428571	0.931818182
Post-Pruning	0	0	0	0	0	0	0
Best_Tree_Tr&Val	0	0	0	0	0	0	0

표2) perf_table_d2

perf_table_d1을 보았을 때 AUROC가 0.88로 해당 모델의 구분력이 좋다고 할 수 있다. 또한 TPR, Precision, TNR, Accuracy, F1-Measure 모두다 0.9를 넘기지 못하여 완벽히 구분력이 좋은 모델이라고 할 수는 없지만 주어진 Test 셋의 크기가 99개 밖에 안 된다는 것을 고려하면 구분력이 괜찮다고 할 수 있다.

perf_table_d2를 보면 AUROC 가 0.9318로 구분력이 상당히 좋은 모델이라고 할 수 있다. 나머지 성능수치들 또한 0.90이상인 것을 볼 때 해당 Tree는 상당히 구분력이 좋은 모델이라고 할 수 있다.

[Q3-1] 앞에서 생성한 Tree에 대해서 적절한 Post-Pruning을 수행한 뒤 결과물을 Plotting하고 이에 대한 해석을 수행하시오. Pruning 전과 후에 Split에 사용된 변수는 어떤 변화가 있는가? Test dataset에 대한 분류성능을 평가하고 [Q2]의 결과와 비교해보시오

- Post-Pruning

```
#data1
set.seed(123456)
data1_tree_q3<-cv.tree(data1_tree_q2, FUN=prune.misclass)
plot(data1_tree_q3)
#size 4 is optimal
data1_tree_q3_pp<-prune.misclass(data1_tree_q2, best=4)
plot(data1_tree_q3_pp)
text(data1_tree_q3_pp, pretty = 1)
# Performance of the tree
data1_tree_q3_prediction<-predict(data1_tree_q3_pp, newdata = data1_tst,type="class")
data1_tree_q3_cm <- table(data1_tst$diagnosis, data1_tree_q3_prediction)
data1_tree_q3_cm
perf_table_d1[2,c(1:6)] <-perf_eval(data1_tree_q3_cm)
data1_q3_pred <-prediction(as.numeric(data1_tree_q3_prediction),as.numeric(data1_tst$diagnosis))
perf_table_d1[2,7]<-as.numeric(performance(data1_q3_pred,measure = "auc")@y.values)
perf_table_d1
#data2
set.seed(123456)
data2_tree_q3<-cv.tree(data2_tree_q2, FUN=prune.misclass)
plot(data2_tree_q3)
#size 5 is optimal
data2_tree_q3_pp<-prune.misclass(data2_tree_q2, best=5)
plot(data2_tree_q3_pp)
text(data2_tree_q3_pp, pretty = 1)
# Performance of the tree
data2_tree_q3_prediction<-predict(data2_tree_q3_pp, newdata = data2_tst,type="class")
data2_tree_q3_cm <- table(data2_tst$blueWins, data2_tree_q3_prediction)
data2_tree_q3_cm
perf_table_d2[2,c(1:6)] <-perf_eval(data2_tree_q3_cm)
data2_q3_pred <-prediction(as.numeric(data2_tree_q3_prediction),as.numeric(data2_tst$blueWins))
perf_table_d2[2,7]<-as.numeric(performance(data2_q3_pred,measure = "auc")@y.values)
perf_table_d2
```

cv.tree 함수를 이용하여 tree 오브젝트를 대상으로 교차확인을 실행하였다. Function은 misclassfication 된 변수의 수로 잡았다. 이후 복잡도 계수를 plot해 보았다. 우선 Dataset 1에 관한 plot부터 살펴보겠다.

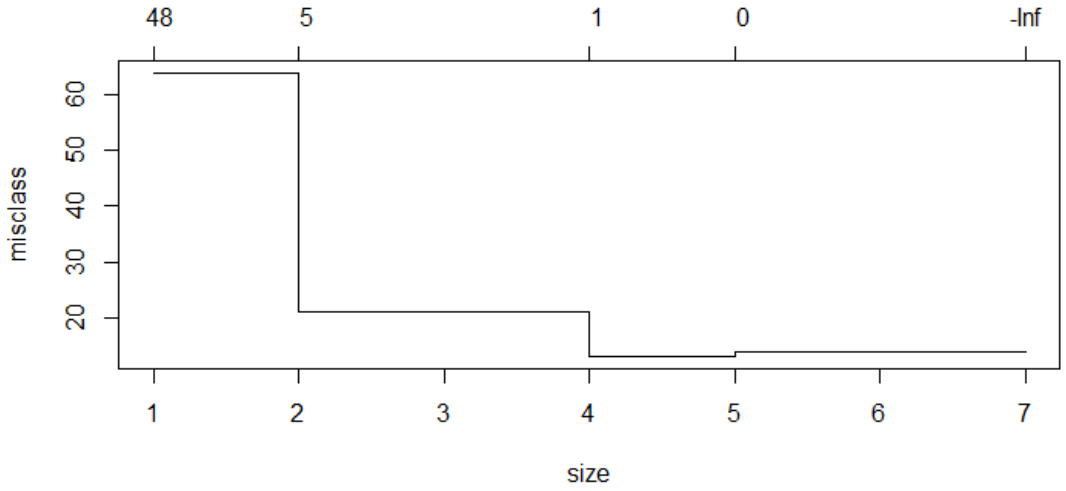

그림3) Dataset1의 Post-Pruning 복잡도계수

복잡도계수를 보면 size가 leaf node의 개수가 4일 때 복잡도계수가 최저임을 알 수 있다. Leaf node의 개수를 4개로 고정 후 의사결정 나무를 plot해 보았다.

6개의 변수를 사용하던 Pruning을 거치지 않았던 Tree와 달리 Post Pruning을 거친 Tree는 총 3개의 변수를 사용한다. 이때의 Tree의 성능을 보면 아래의 perf_table_d1과 같다.

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure	AUROC
Non-Pruning	0.818181818	0.8571429	0.9610390	0.9292929	0.8867382	0.8372093	0.8896104
Post-Pruning	0.818181818	0.75	0.9220779	0.8989899	0.8685778	0.7826087	0.8701299
Best_Tree_Tr&Val	0	0	0	0	0	0	0

표1) perf_table_d1

Post Prune 이후 성능은 조금 하락하였지만 Tree의 구조는 더 단순화 되었음을 알 수있다.

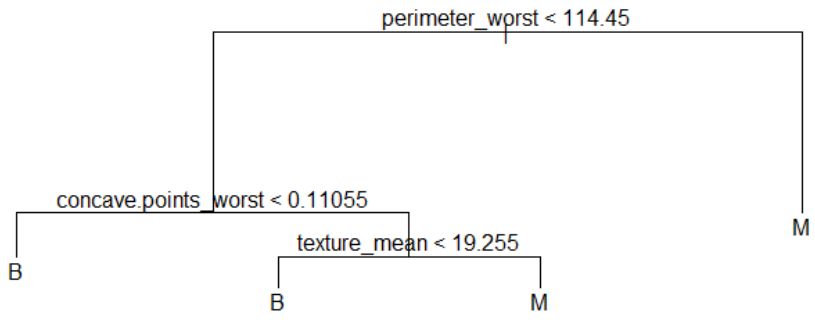

그림4) Dataset1의 Post-Pruning을 진행한 Tree

DataSet2의 leaf node 별 복잡도 계수를 plot 해보면 아래와 같다.

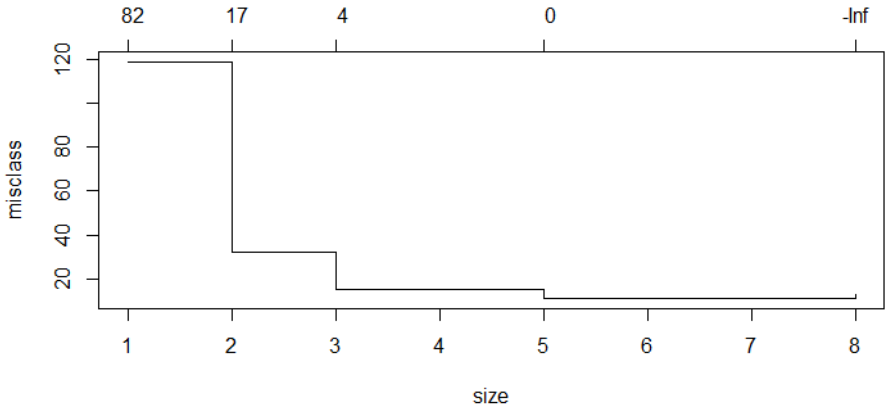

그림5) Dataset2의 Post-Pruning 복잡도계수

복잡도계수를 보면 size가 leaf node의 개수가 5일 때 복잡도계수가 최저임을 알 수 있다. Leaf node의 개수를 5개로 고정 후 의사결정 나무를 plot해 보았다.

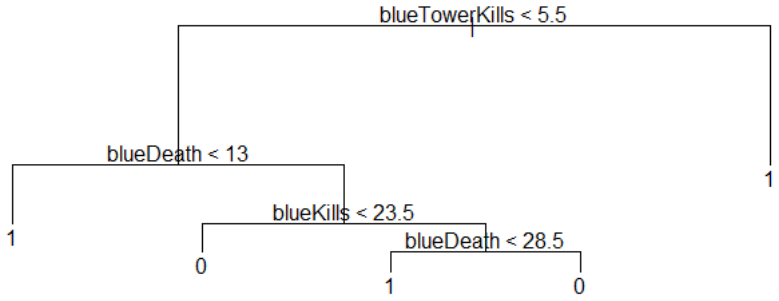

그림6) Dataset2의 Post-Pruning을 진행한 Tree

4개의 변수를 사용하던 Pruning을 거치지 않았던 Tree와 달리 Post Pruning을 거친 Tree는 총 3개의 변수를 사용한다. 이때의 Tree의 성능을 보면 아래의 perf_table_d2과 같다.

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure	AUROC
Non-Pruning	0.963636364	0.929824561	0.9	0.936842105	0.931274786	0.946428571	0.931818182
Post-Pruning	0.963636364	0.929824561	0.9	0.936842105	0.931274786	0.946428571	0.931818182
Best_Tree_Tr&Val	0	0	0	0	0	0	0

표2) perf_table_d2

perf_table_d2를 보면 모든 성능 지표에서 모두 pruning전과 동일한 것을 알 수 있다. 성능은 동일하지만 Tree의 복잡도도 내려간 Post Pruning을 한 Tree모델이 더 낫다고 할 수 있다.

[Q3-2] "party" package를 사용하여 다음 조건에 맞는 Pre-pruning을 수행하여 가장 최적의min_criterion, min_split, max_depth 값을 찾아보시오.
(조건1) 각 하이퍼파라미터들은 최소 5가지 이상의 후보 값을 가질 것 (총125가지 이상)
(조건2) 평가지표는 Validation Dataset에 대한 AUROC 사용

- Pre-Pruning (DataSet 1)

```
# tree parameter settings
d1_min_criterion = c(0.875,0.9, 0.95, 0.975, 0.99)
d1_min_split = c(10,30,50,100,150)
d1_max_depth = c(0,1,3,5,7)
data1_q3_2 = matrix(0,length(d1_min_criterion)*length(d1_min_split)*length(d1_max_depth),11)
colnames(data1_q3_2) <- c("min_criterion", "min_split", "max_depth",
                                    "TPR", "Precision", "TNR", "ACC", "BCR", "F1", "AUROC",
"N_leaves")
iter_cnt = 1
for (i in 1:length(d1_min_criterion)){
  for ( j in 1:length(d1_min_split)){
    for ( k in 1:length(d1_max_depth)){

      cat("Min criterion:", d1_min_criterion[i], ", Min split:", d1_min_split[j], ", Max depth:",
d1_max_depth[k], "\n")
      tmp_control = ctree_control(mincriterion = d1_min_criterion[i], minsplit = d1_min_split[j],
maxdepth = d1_max_depth[k])
      tmp_tree <- ctree(diagnosis ~ ., data = data1_trn, controls = tmp_control)
      tmp_tree_val_prediction <- predict(tmp_tree, newdata = data1_val)
      tmp_tree_val_response <- treeresponse(tmp_tree, newdata = data1_val)
      tmp_tree_val_prob <- 1-unlist(tmp_tree_val_response,
use.names=F)[seq(1,nrow(data1_val)*2,2)]
      tmp_tree_val_rocr <- prediction(tmp_tree_val_prob, data1_val$diagnosis)
      # Confusion matrix for the validation dataset
      tmp_tree_val_cm <- table(data1_val$diagnosis, tmp_tree_val_prediction)

      # parameters
      data1_q3_2[iter_cnt,1] = d1_min_criterion[i]
      data1_q3_2[iter_cnt,2] = d1_min_split[j]
      data1_q3_2[iter_cnt,3] = d1_max_depth[k]
      # Performances from the confusion matrix
      data1_q3_2[iter_cnt,4:9] = perf_eval(tmp_tree_val_cm)
      # AUROC
      data1_q3_2[iter_cnt,10] = unlist(performance(tmp_tree_val_rocr, "auc")@y.values)
      # Number of leaf nodes
      data1_q3_2[iter_cnt,11] = length(nodes(tmp_tree, unique(where(tmp_tree))))
      iter_cnt = iter_cnt + 1
    }
  }
}
# Find the best set of parameters
data1_q3_2 <- data1_q3_2[order(data1_q3_2[,10], decreasing = T),]
data1_q3_2
d1_best_criterion <- data1_q3_2[1,1]
d1_best_split <- data1_q3_2[1,2]
d1_best_depth <- data1_q3_2[1,3]
```

party 패키지를 이용한 pre-pruning을 위해 min_criterion, min_split, max_depth 이 세가지의 hyperparmeter를 조정해주어 최적의 hyperparameter 조합을 찾아보았다. min_criterion 즉 불순도의 감소의 통계적 유의성 검증을 위한 기준은 0.875부터 가장 예민한 0.99까지 설정해주었다. 최소한 X개의 수 이상이여야 split의 여부를 결정하는 min_split의 기준은 training 데이터 셋의 데이터 수가 239개의 한정된 수를 감안하여 최소 10부터 150까지 5단계로 나누어 설정하였다. max_depth는 Tree가 분개될때마다 1씩 늘어나는 기준으로 0,1,3,5,7로 간격을 두어 관찰하였다.

5 X 5 X 5 즉 125가지 모델의 성능을 data1_q3_2에 저장해주었고 이를 AUROC기준으로 정렬후 최적의 hyperparmeter 조합을 찾았다.

	min_criterion	min_split	max_depth	TPR	Precision	TNR	ACC	BCR	F1	AUROC	N_leaves
1	0.875	50	0	0.833333	0.833333	0.958333	0.933333	0.89365	0.833333	0.921875	3
2	0.875	50	3	0.833333	0.833333	0.958333	0.933333	0.89365	0.833333	0.921875	3
3	0.875	50	5	0.833333	0.833333	0.958333	0.933333	0.89365	0.833333	0.921875	3
4	0.875	50	7	0.833333	0.833333	0.958333	0.933333	0.89365	0.833333	0.921875	3
5	0.875	100	0	0.833333	0.833333	0.958333	0.933333	0.89365	0.833333	0.921875	3

표3) data1_q3_2

최적의 hyperparametr 조합은 min_criterion이 0.875, min_split이 50, max_depth 주어진 조합에서는 상관이 없게 나왔지만, Leaf node의 개수가 3개인 것을 보아 2보다 커야함을 알 수 있다. 이 때의 Leaf node의 수는 3개로 pruning을 거치지 않은 Tree의 leaf node의 수 (8개)보다 줄어든 것을 확인 할 수 있다.

■ Pre-Pruning (DataSet 2)

```
#data2
# tree parameter settings
d2_min_criterion = c(0.875,0.9, 0.95, 0.975, 0.99)
d2_min_split = c(10,30,50,70,100)
d2_max_depth = c(0,1,3,5,7)
data2_q3_2 = matrix(0,length(d2_min_criterion)*length(d2_min_split)*length(d2_max_depth),11)
colnames(data2_q3_2) <- c("min_criterion", "min_split", "max_depth",
                          "TPR", "Precision", "TNR", "ACC", "BCR", "F1", "AUROC", "N_leaves")
iter_cnt = 1
for (i in 1:length(d2_min_criterion)){
  for ( j in 1:length(d2_min_split)){
    for ( k in 1:length(d2_max_depth)){

      cat("Min criterion:", d2_min_criterion[i], ", Min split:", d2_min_split[j], ", Max depth:",
d2_max_depth[k], "\n")
      tmp_control = ctree_control(mincriterion = d2_min_criterion[i], minsplit = d2_min_split[j],
maxdepth = d2_max_depth[k])
      tmp_tree <- ctree(blueWins ~ ., data = data2_trn, controls = tmp_control)
      tmp_tree_val_prediction <- predict(tmp_tree, newdata = data2_val)
      tmp_tree_val_response <- treeresponse(tmp_tree, newdata = data2_val)
      tmp_tree_val_prob <- 1-unlist(tmp_tree_val_response,
use.names=F)[seq(1,nrow(data2_val)*2,2)]
      tmp_tree_val_rocr <- prediction(tmp_tree_val_prob, data2_val$blueWins)
      # Confusion matrix for the validation dataset
      tmp_tree_val_cm <- table(data2_val$blueWins, tmp_tree_val_prediction)

      # parameters
      data2_q3_2[iter_cnt,1] = d2_min_criterion[i]
      data2_q3_2[iter_cnt,2] = d2_min_split[j]
      data2_q3_2[iter_cnt,3] = d2_max_depth[k]
      # Performances from the confusion matrix
      data2_q3_2[iter_cnt,4:9] = perf_eval(tmp_tree_val_cm)
      # AUROC
      data2_q3_2[iter_cnt,10] = unlist(performance(tmp_tree_val_rocr, "auc")@y.values)
      # Number of leaf nodes
      data2_q3_2[iter_cnt,11] = length(nodes(tmp_tree, unique(where(tmp_tree))))
      iter_cnt = iter_cnt + 1
    }
  }
}
# Find the best set of parameters
data2_q3_2 <- data2_q3_2[order(data2_q3_2[,10], decreasing = T),]
data2_q3_2
d2_best_criterion <- data2_q3_2[1,1]
d2_best_split <- data2_q3_2[1,2]
d2_best_depth <- data2_q3_2[1,3]
```

DataSet 2 또한 DataSet 1과 비슷하게 Training Set의 수가 229개로 Hyperparameter의 조합을 똑같이 설정하였다. min_criterion은 0.875부터 0.99까지, split의 여부를 결정하는 min_split은 최소 10부터 150까지 5단

계로 나누어 설정하였다. max_depth는 Tree가 분개될때마다 1씩 늘어나는 기준으로 0,1,3,5,7로 간격을 두어 관찰하였다.

5 X 5 X 5 즉 125가지 모델의 성능을 data2_q3_2에 저장해주었고 이를 AUROC기준으로 정렬후 최적의 hyperparmeter 조합을 찾았다.

	min_criterion	min_split	max_depth	TPR	Precision	TNR	ACC	BCR	F1	AUROC	N_leaves
1	0.875	10	3	0.869565	0.952381	0.970588	0.929825	0.918689	0.909091	0.950128	6
2	0.875	30	3	0.869565	0.952381	0.970588	0.929825	0.918689	0.909091	0.950128	6
3	0.875	50	3	0.869565	0.952381	0.970588	0.929825	0.918689	0.909091	0.950128	6
4	0.875	70	3	0.869565	0.952381	0.970588	0.929825	0.918689	0.909091	0.950128	6
5	0.9	10	3	0.869565	0.952381	0.970588	0.929825	0.918689	0.909091	0.950128	6

표4) data2_q3_2

최적의 hyperparametr 조합은 min_criterion이 0.875, max_depth는 3, min_split의 조건은 크게 상관이 없는 것을 알 수 있다. Leaf node의 수는 6개로 pruning을 거치지 않은 Tree의 leaf node의 수 (8개)보다 줄어든 것을 확인 할 수 있다.

[Q4] 최적의 결정나무의 Plot을 그리고, 대표적인 세가지 규칙에 대해서 설명해보시오.

- Plotting the Decision Tree

```
#data1
#Find the best model
d1_tree_control = ctree_control(mincriterion = d1_best_criterion, minsplit = d1_best_split,
maxdepth = d1_best_depth)
data1_tree_q4 <- ctree(diagnosis ~ ., data = data1_trn, controls = d1_tree_control)
# Plot the best tree
plot(data1_tree_q4, type="simple")
#data2
#Find the best model
d2_tree_control = ctree_control(mincriterion = d2_best_criterion, minsplit = d2_best_split,
maxdepth = d2_best_depth)
data2_tree_q4 <- ctree(blueWins ~ ., data = data2_trn, controls = d2_tree_control)
# Plot the best tree
plot(data2_tree_q4, type="simple")
```

[Q3-2]에서 구한 최적의 결정나무는 아래의 그림과 같다.

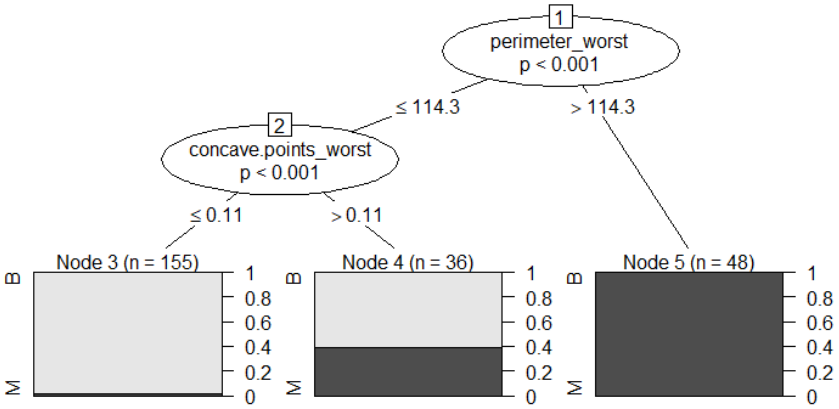

그림7) Dataset 1로 구한 Pre-Pruning을 거친 Decision Tree

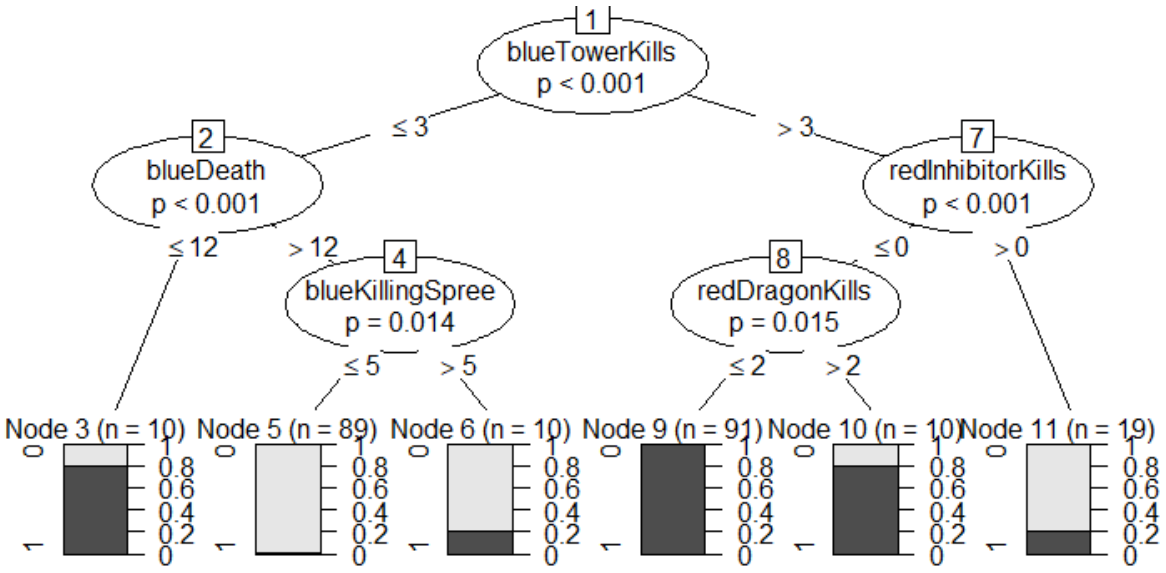

그림8) Dataset 2로 구한 Pre-Pruning을 거친 Decision Tree

Dataset 1 으로 구한 Pre Pruning 을 거친 Decision Tree 의 대표적인 규칙 3 가지는 아래와 같다.

첫번째 규칙) 주어진 데이터 중 둘레의 극값, 즉 perimeter_worst 의 값이 114.3 보다 크면 100%의 확률로 Maligant, 즉 악성 종양, 유방암으로 분류한다.

두번째 규칙) 주어진 데이터 중 둘레의 극값, 즉 perimeter_worst 의 값이 114.3 보다 작고 concave.points _worst 가 0.11 보다 작거나 같으면 98.7%의 확률로 Benign, 즉 양성종양, 유방암이 아닌것으로 분류한다.

세번째 규칙 주어진 데이터 중 둘레의 극값, 즉 perimeter_worst 의 값이 114.3 보다 작고 concave.points _worst 가 0.11 보다 크다면 38.9%의 확률로 악성종양, 유방암으로 분류한다.

Dataset 2 로 구한 Pre Pruning 을 거친 Decision Tree 의 대표적인 규칙 3 가지는 아래와 같다.

첫번째 규칙) 블루팀이 부순 타워의 수가 3 개 보다 많고, 상대팀의 억제기를 전혀 부수지 않고 상대팀이 용을 두마리 이하로 죽였다면 100%의 확률로 블루팀이 이겼다고 할 수 있다.

두번째 규칙) 블루팀이 부순 타워의 수가 3 개 이하이며, 블루팀의 팀 데스수가 12 이하이면 80%의 확률로 블루팀이 이겼다고 할 수 있다.

세번째 규칙) 블루팀이 부순 타워의 수가 3 개 이하이며, 블루팀의 데스수가 12 보다 크고, 블루팀의 연속킬이 5 회이하이면 98.9%의 확률로 블루팀이 졌다고 할 수 있다.

[Q5] [Q4]에서 선택한 하이퍼 파라미터 조합을 이용하여 Training Dataset 과 Validation Dataset 을 결합한 데이터셋을 학습한 뒤, Test Datase 에 적용해보고 분류 성능을 평가하시오

- Performance Evaluation

```
#data1
# Use the training and validation dataset to train the best tree
data1_trn_q5 <- rbind(data1_trn, data1_val)
data1_tree_q5 <- ctree(diagnosis ~ ., data = data1_trn_q5, controls = d1_tree_control)
data1_q5_pre_prediction <- predict(data1_tree_q5, newdata = data1_tst)
data1_q5_pre_response <- treeresponse(data1_tree_q5, newdata = data1_tst)
# Performance of the best tree
data1_q5_cm <- table(data1_tst$diagnosis, data1_q5_pre_prediction)
```

```
data1_q5_cm
perf_table_d1[3,c(1:6)] <- perf_eval(data1_q5_cm)
data1_q5_pred <-prediction(as.numeric(data1_q5_pre_prediction),as.numeric(data1_tst$diagnosis))
perf_table_d1[3,7]<-as.numeric(performance(data1_q5_pred,measure = "auc")@y.values)
perf_table_d1
#data2
# Use the training and validation dataset to train the best tree
data2_trn_q5 <- rbind(data2_trn, data2_val)
data2_tree_q5 <- ctree(blueWins ~ ., data = data2_trn_q5, controls = d2_tree_control)
data2_q5_pre_prediction <- predict(data2_tree_q5, newdata = data2_tst)
data2_q5_pre_response <- treeresponse(data2_tree_q5, newdata = data2_tst)
# Performance of the best tree
data2_q5_cm <- table(data2_tst$blueWins, data2_q5_pre_prediction)
data2_q5_cm
perf_table_d2[3,c(1:6)] <- perf_eval(data2_q5_cm)
data2_q5_pred <-prediction(as.numeric(data2_q5_pre_prediction),as.numeric(data2_tst$blueWins))
perf_table_d2[3,7]<-as.numeric(performance(data2_q5_pred,measure = "auc")@y.values)
perf_table_d2
```

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure	AUROC
Non-Pruning	0.818181818	0.8571429	0.9610390	0.9292929	0.8867382	0.8372093	0.8896104
Post-Pruning	0.818181818	0.75	0.9220779	0.8989899	0.8685778	0.7826087	0.8701299
Best_Tree_Tr&Val	0.6818182	0.8823529	0.9740260	0.9090909	0.8149286	0.7692308	0.8279221

표1) perf_table_d1

	TPR	Precision	TNR	Accuracy	BCR	F1-Measure	AUROC
Non-Pruning	0.963636364	0.929824561	0.9	0.936842105	0.931274786	0.946428571	0.931818182
Post-Pruning	0.963636364	0.929824561	0.9	0.936842105	0.931274786	0.946428571	0.931818182
Best_Tree_Tr&Val	0.8727273	0.9230769	0.9	0.8842105	0.8862587	0.8971963	0.8863636

표2) perf_table_d2

우선 Dataset 1 의 Training Set 과 Validation Set 을 합친 데이터를 바탕으로 최적의 하이퍼파라미터의 조합으로 학습시킨 Tree 의 결과물은 위의 표와 같다. 위에 표에서 보고 알 수 있듯 True Positive Rate (TPR)과 BCR, AUROC 의 값은 그 전보다 못하다는 것을 알 수 있다. 하지만 Precision 이나 True Negative Rate, Accuracy 는 더 나아진 것을 확인할 수 있다. 이를 통해 세가지의 모델 중 어느 모델이 더 우월하다고 확정지어 말하긴 힘들지만, 마지막으로 구성한 Tree 의 복잡도가 가장 낮은거에 (Leaf node 의 수 3 개) 불구하고 성능이 비슷한 것을 보아 해당 Tree 가 가장 나은 모델이라고 할 수 있다.

두번째 Dataset 2 의 Training Set 과 Validation Set 을 합친 데이터를 바탕으로 최적의 하이퍼파라미터의 조합으로 학습시킨 Tree 의 결과물은 위의 표와 같다. Precison, 과 TNR 을 제외한 나머지 모든 지표에서 다른 Tree 들 보다 성능이 열등하게 나왔다. 마지막 Pre Pruning 을 거쳐 만든 모델은 Leafnode 의 수가 6 개이지만, 이전 Post-Pruning 을 통해 만든 Leaf node 의 수는 5 개로 더 Tree 의 복잡도가 낮은 모델이다. 고로 Post-Pruning 을 거친 Tree 모형이 주어진 정보를 바탕으로 보았을 때 세가지 Tree 모델 중 제일 나은 모델이라고 할 수 있다.

[Q6] 과제 2 를 수행하기 위해 사용된 데이터셋(Dataset1)과 이번 과제 수행을 위해 선택된 데이터셋(Dataset2)에 대해서 각각 로지스틱 회귀분석을 수행하여 TestDataset 에 대한 다음 ConfusionMatrix 를 채우고 이에 대한 결과를 해석해보시오.

- Logistic Regression

```
#로지스틱 회귀분석을 진행하기 전 주어진 데이터 셋을 표준화 시켜준다
#data1
data1_trn_q6 <- data.frame(scale(data1_trn_q5[,c(1:30)], center = TRUE, scale = TRUE),diagnosis =
as.numeric(data1_trn_q5$diagnosis))
data1_tst_q6 <- data.frame(scale(data1_tst[,c(1:30)], center = TRUE, scale = TRUE),diagnosis =
as.numeric(data1_tst$diagnosis))
data1_trn_q6$diagnosis<-ifelse(data1_trn_q6$diagnosis==2,1,0)
data1_tst_q6$diagnosis<-ifelse(data1_tst_q6$diagnosis==2,1,0)
```

```
perf_table_d1_q6 <- matrix(0, nrow = 2, ncol = 5)
rownames(perf_table_d1_q6) <- c("Logistic Regression", "Decison Tree")
colnames(perf_table_d1_q6) <- c("TPR", "TNR", "Accuracy", "BCR", "F1-Measure")
perf_table_d1_q6[2,]<-perf_table_d1[3,c(1,3,4,5,6)]
perf_table_d1_q6
#data1 로지스틱 회귀
#상한 & 하한선 설정 for ForwardSelection
data1_tmp_x <- paste(colnames(data1_trn_q6)[-31], collapse=" + ")
data1_tmp_x
data1_tmp_xy <- paste("diagnosis ~ ", data1_tmp_x, collapse = "")
as.formula(data1_tmp_xy)
#Forward Selection
data1_logi_forward <- step(glm(diagnosis~ 1, data = data1_trn_q6),
                    scope = list(upper = as.formula(data1_tmp_xy), lower = diagnosis~ 1),
                    direction="forward")
summary(data1_logi_forward)
# Make prediction
data1_forward_prob <- predict(data1_logi_forward, type = "response", newdata = data1_tst_q6)
data1_forward_prey <- rep(0, nrow(data1_tst_q6))
data1_forward_prey[which(data1_forward_prob >= 0.5)] <- 1
data1_forward_cm <- table(data1_tst_q6$diagnosis, data1_forward_prey)
data1_forward_cm
# Performance evaluation
perf_table_d1_q6[1,]<-perf_eval(data1_forward_cm)[c(1,3,4,5,6)]
perf_table_d1_q6
#data2
data2_trn_q5[,c(2:6)]<-data.frame(apply(data2_trn_q5[,c(2:6)], 2, as.numeric))
data2_tst[,c(2:6)]<-data.frame(apply(data2_tst[,c(2:6)], 2, as.numeric))
data2_trn_q6 <- data.frame(scale(data2_trn_q5[,c(1:39)], center = TRUE, scale = TRUE),blueWins =
as.numeric(data2_trn_q5$blueWins))
data2_tst_q6 <- data.frame(scale(data2_tst[,c(1:39)], center = TRUE, scale = TRUE),blueWins =
as.numeric(data2_tst$blueWins))
data2_trn_q6$blueWins<-ifelse(data2_trn_q6$blueWins==2,1,0)
data2_tst_q6$blueWins<-ifelse(data2_tst_q6$blueWins==2,1,0)
perf_table_d2_q6 <- matrix(0, nrow = 2, ncol = 5)
rownames(perf_table_d2_q6) <- c("Logistic Regression", "Decison Tree")
colnames(perf_table_d2_q6) <- c("TPR", "TNR", "Accuracy", "BCR", "F1-Measure")
perf_table_d2_q6[2,]<-perf_table_d2[3,c(1,3,4,5,6)]
perf_table_d2_q6
#data2 로지스틱 회귀
#상한 & 하한선 설정 for ForwardSelection
data2_tmp_x <- paste(colnames(data2_trn_q6)[-40], collapse=" + ")
data2_tmp_x
data2_tmp_xy <- paste("blueWins ~ ", data2_tmp_x, collapse = "")
as.formula(data2_tmp_xy)
#Forward Selection_part2
data2_logi_forward <- step(glm(blueWins~ 1, data = data2_trn_q6),
                        scope = list(upper = as.formula(data2_tmp_xy), lower = blueWins~ 1),
                        direction="forward")
summary(data2_logi_forward)
# Make prediction
data2_forward_prob <- predict(data2_logi_forward, type = "response", newdata = data2_tst_q6)
data2_forward_prey <- rep(0, nrow(data2_tst_q6))
data2_forward_prey[which(data2_forward_prob >= 0.5)] <- 1
data2_forward_cm <- table(data2_tst_q6$blueWins, data2_forward_prey)
data2_forward_cm
# Performance evaluation
perf_table_d2_q6[1,]<-perf_eval(data2_forward_cm)[c(1,3,4,5,6)]
perf_table_d2_q6
```

Logistic Regression 을 해주기 전 각 데이터셋을 표준화 시켜준다. DataSet 2 에는 factor 형으로 되어있는 변수를 numerical 변수로 바꿔준후 표준화를 진행한다. Logistic Regression 과 Decision Tree 의 성능을 비교하기 위해선 같은 DataSet 으로 학습된 모델을 비교해야하기 때문에 Q5 에서 썼던 Training Set 과 Validation Set 을 합쳐준 데이터를 이용한다. 물론 비교대상이 될 Decision Tree 도 이를 바탕으로 생성한 최적의 hyper parameter 의 조합으로 만들어진 Tree 를 사용한다.

Decision Tree 또한 Pruning을 거친 모델로 성능을 비교하기 때문에 Logistic Regression을 진행할 때 Forward Selection 기법으로 차원을 축소한 결과를 측정하여 비교하기로 하였다.

다음은 DataSet 1로 decision tree와 Logistic Regression을 진행한 결과의 성능 비교 차트이다.

	TPR	TNR	Accuracy	BCR	F1-Measure
Logistic Regression	0.818182	1	0.959596	0.904534	0.9
Decison Tree	0.681818	0.974026	0.909091	0.814929	0.769231

표3) perf_table_d1_q6

해당 표를 보면 Logistic Regression의 모든 지표상에서 더 우월한 결과를 나왔으므로 DataSet 1에대한 구분능력은 Logistici Regression이 Decision Tree보다 성능이 더 좋다는 것을 알 수 있다. 특히나 True Negative Rate가 1인 것을 통해 Logistic Regression으로 구한 모델로는 True Negative 즉 진단 결과 양성 종양이라고 판단하는 구분력은 매우 높다는 것을 알 수 있다.

다음은 DataSet 2로 decision tree와 Logistic Regression을 진행한 결과의 성능 비교 차트이다.

	TPR	TNR	Accuracy	BCR	F1-Measure
Logistic Regression	0.981818	1	0.989474	0.990867	0.990826
Decison Tree	0.872727	0.9	0.884211	0.886259	0.897196

표4) perf_table_d2_q6

해당 표를 보면 알 수 있듯 DataSet1과 같이 DataSet 2로 만든 Logistic Regression의 모델이 Decision Tree보다 모든 지표상에서 더 우월한 결과를 얻어낸 것을 알 수 있다. TPR, TNR, Accuracy, BCR, F1-Measure 모두 1에 근접한 결과를 보여주었고, 이를 통해 Logistic Regression으로 만든 모델은 주어진 Test 셋을 거의 완벽하게 분류해냈다는 것을 알 수 있다. 물론 Decision Tree를 통해 구한 모델도 구분력이 있는 좋은 모델이지만, Logistic Regression을 통해 구한 모델이 더 구분력이 좋은 모델이라고 할 수 있다.

문제에서 주어진 confusion matrix을 채워보면 다음과 같다.

Dataset	Model	TPR	TNR	Accuracy	BCR	F1-Measure
Dataset 1	Logistic Regression	0.818182	1	0.959596	0.904534	0.9
	Decison Tree	0.681818	0.974026	0.909091	0.814929	0.769231
Dataset 2	Logistic Regression	0.981818	1	0.989474	0.990867	0.990826
	Decison Tree	0.872727	0.9	0.884211	0.886259	0.897196

[Q7] 각 데이터셋마다 Logistic Regression에 의해 중요하다고 판별된 변수들과 의사결정 나무에 의해 중요하다고 판별된 변수들을 확인해보고 차이가 있는지의 여부와, 차이가 존재할 경우 그 이유에 대한 본인의 생각을 서술해보시오.

DataSet1 과 DataSet 2 로 만든 Logistic Regression 모델에서 쓰인 변수들을 파악하기 위해 `summary(data1_logi_forward), summary(data2_logi_forward)` 를 확인해 보았다.

- Summary of data1_logi_forward

```
> summary(data1_logi_forward)
Call:
glm(formula = diagnosis ~ area_worst + concavity_mean + texture_worst +
    perimeter_mean + concave.points_mean + compactness_mean +
    perimeter_worst + texture_mean + fractal_dimension_worst +
    radius_mean + symmetry_mean, data = data1_trn_q6)
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.57578  -0.12947  -0.02287   0.09460   0.81685
Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)              0.25418    0.01327  19.161  < 2e-16 ***
area_worst               0.16792    0.09221   1.821   0.0696 .
concavity_mean           0.04502    0.04475   1.006   0.3152
texture_worst           -0.01138    0.03507  -0.324   0.7459
perimeter_mean           1.10789    0.69437   1.596   0.1117
concave.points_mean      0.20152    0.04638   4.345 1.94e-05 ***
compactness_mean        -0.22694    0.04986  -4.551 7.89e-06 ***
perimeter_worst          0.27938    0.11877   2.352   0.0193 *
texture_mean             0.08769    0.03441   2.548   0.0113 *
fractal_dimension_worst  0.05362    0.02243   2.391   0.0174 *
radius_mean             -1.33878    0.66189  -2.023   0.0440 *
symmetry_mean            0.02329    0.01533   1.519   0.1298
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for gaussian family taken to be 0.05261821)
    Null deviance: 56.682  on 298  degrees of freedom
Residual deviance: 15.101  on 287  degrees of freedom
AIC: -18.185
Number of Fisher Scoring iterations: 2
```

summary(data1_logi_forward) 를 통해 DataSet 1으로 forward selection 기법으로 변수를 선택한 Logistic Regression 모델은 총 11개의 변수를 사용한 것을 알 수 있다. Decision Tree를 통해 만든 모델에서는 2개의 변수를 이용한 거와는 상반된 결과이다.두 모델에서 공통적으로 사용된 변수는 perimeter_worst 변수 하나이다.

- Summary of data2_logi_forward

```
> summary(data2_logi_forward)
Call:
glm(formula = blueWins ~ blueTowerKills + blueDeath + blueKills +
    redTowerKills + blueAssist + redTotalLevel + blueTotalLevel +
    blueKillingSpree + blueFirstBaron + blueJungleMinionKills +
    blueDragonKills + redKillingSpree + blueTotalMinionKills +
    blueTotalGold + blueWardkills + gameDuraton + blueWardPlaced +
    blueFirstBlood + blueFirstInhibitor + redBaronKills + redTotalGold,
    data = data2_trn_q6)
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.38598  -0.09367  -0.00276   0.10288   0.45096
Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)         0.475524   0.008675  54.816  < 2e-16 ***
blueTowerKills      0.030537   0.031127   0.981 0.327465
blueDeath           0.064673   0.032430   1.994 0.047155 *
blueKills          -0.296556   0.046834  -6.332 1.04e-09 ***
redTowerKills      -0.048936   0.020292  -2.412 0.016567 *
blueAssist          0.031520   0.024913   1.265 0.206905
```

```
redTotalLevel         -0.468371   0.060287  -7.769 1.76e-13 ***
blueTotalLevel         0.672175   0.061150  10.992  < 2e-16 ***
blueKillingSpree       0.089129   0.021180   4.208 3.53e-05 ***
blueFirstBaron        -0.045801   0.014232  -3.218 0.001451 **
blueJungleMinionKills -0.121358   0.023827  -5.093 6.70e-07 ***
blueDragonKills        0.052564   0.014200   3.702 0.000261 ***
redKillingSpree       -0.053054   0.017550  -3.023 0.002750 **
blueTotalMinionKills  -0.176614   0.040957  -4.312 2.29e-05 ***
blueTotalGold          0.354983   0.107439   3.304 0.001085 **
blueWardkills          0.023514   0.018649   1.261 0.208481
gameDuraton           -0.091182   0.066317  -1.375 0.170315
blueWardPlaced         0.044523   0.020207   2.203 0.028434 *
blueFirstBlood         0.017677   0.010305   1.715 0.087447 .
blueFirstInhibitor     0.031832   0.017344   1.835 0.067586 .
redBaronKills          0.028708   0.015208   1.888 0.060160 .
redTotalGold          -0.136621   0.084698  -1.613 0.107931
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for gaussian family taken to be 0.02152249)
    Null deviance: 71.3287  on 285  degrees of freedom
Residual deviance:  5.6819  on 264  degrees of freedom
AIC: -263.12
Number of Fisher Scoring iterations: 2
```

summary(data2_logi_forward) 를 통해 DataSet 2으로 만든 forward selection 기법으로 변수를 선택한 Logistic Regression 모델은 총 21개의 변수를 사용한 것을 알 수 있다. 이는Decision Tree를 통해 만든 모델에서는 5개의 변수를 이용한 거와는 상반된 결과이다. 두 모델에서 공통적으로 사용된 변수는 blueTowerKills, blueDeath, blueKillingSpree 이렇게 세 변수이다.

이렇게 DataSet 1, DataSet 2 으로 각각 만든 Logistic Regression, Decision Tree, 두 모델간의 변수 선택에 차이가 있는 점은 다음과 같다 생각한다. Decision Tree의 모델의 분개 과정은 CART 알고리즘을 바탕으로 진행된다. 해당 알고리즘은 분개의 기준을 Gini Index로 삼아 Gini Index가 작아지는 방향으로 분개하게 된다. 거기에 [Q3-2]에서 찾은 Hyperparameter의 조합으로 분개의 한계를 정해놓아 많은 변수가 선택되어지지 않은 것으로 판단된다.

이와 반대로 Forward Selection 방식으로 변수를 줄여 만든 Logistic Regression은 변수가 하나도 선택되어 있지 않은 모델에서부터 하나의 변수를 추가하면서 해당 변수를 선택함으로써 AIC를 최소화, 즉 우도를 가장 크게 만드는 최적의 모델을 만들어간다. 두 모델에서의 변수를 선택하는 기준의 차이와 방법의 차이가 있어 변수가 다르게 선택되었다고 생각한다.

만약 둘다 pruning과 forward selection을 거치지 않았더라도 Decision Tree는 주어진 데이터 그룹을 순도 높게 분류하는 알고리즘이고, Logistic Regression은 승산의 개념을 도입해 변수들을 대입하여 1~0 사이의 수를 만들어내는 것이므로 이 둘의 근본적인 결과물의 차이에서 변수 선택의 차이가 나왔다고 볼 수 있다.