Heart Disease Prediction Data

- Classification Model

20171293 수학과 이석희

20171299 수학과 양승민

20171303 수학과 박예준

목차

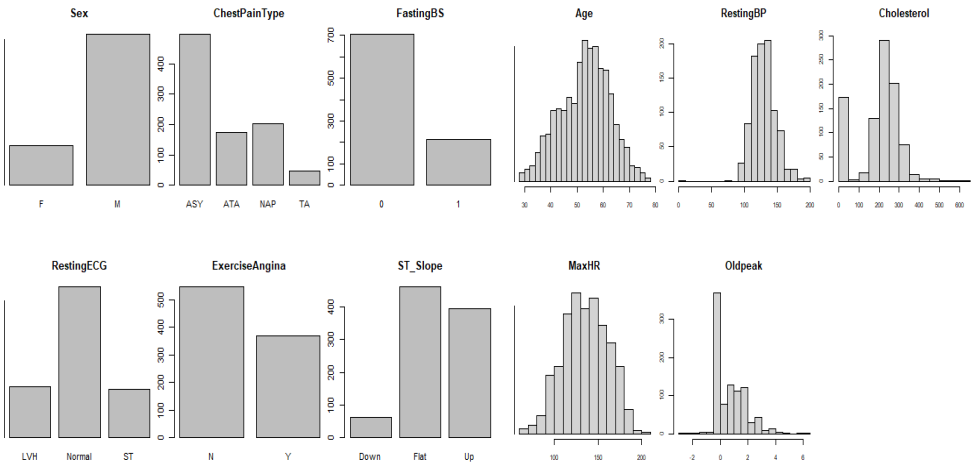
Exploratory Data Analysis

Feature Information

- 설명변수
  - Age : 나이 (years)
  - Sex : 성별 (M : 남자, F : 여자)
  - ChestPainType : 가슴통증 유형 (ATA, NAP, ASY, TA)
  - RestingBP : 안정혈압 (mm Hg)
  - Cholesterol : 혈청 콜레스테롤 수치 (mm/dl)
  - FastingBS : 공복 혈당 (1 : FastingBS > 120mg/dl, 0 : otherwise)
  - RestingECG : 심전도 검사 결과 (Normal, ST, LVH)
  - MaxHR : 최대 심박수
  - ExerciseAngina : 운동유발형 협심증 유무 (Y : 있음, N : 없음)
  - Oldpeak : ST_segment 하강 수치
  - ST_Slope : ST segment의 기울기 (Up, Flat, Down)

- 반응변수
  - HeartDisease : 심장질환 유무 (1 : 있음, 0 : 없음)

Distribution of features



범주형 설명변수 분포 연속형 설명변수 분포

Summary Statistics

```
> summary(heart)
      Age            Sex            ChestPainType        RestingBP      Cholesterol      FastingBS
 Min.   :28.00   Length:918         Length:918         Min.   :  0.0   Min.   :  0.0   Min.   :0.0000
 1st Qu.:47.00   Class :character   Class :character   1st Qu.:120.0   1st Qu.:173.2   1st Qu.:0.0000
 Median :54.00   Mode  :character   Mode  :character   Median :130.0   Median :223.0   Median :0.0000
 Mean   :53.51                                         Mean   :132.4   Mean   :198.8   Mean   :0.2331
 3rd Qu.:60.00                                         3rd Qu.:140.0   3rd Qu.:267.0   3rd Qu.:0.0000
 Max.   :77.00                                         Max.   :200.0   Max.   :603.0   Max.   :1.0000
  RestingECG            MaxHR       ExerciseAngina       Oldpeak          ST_Slope          HeartDisease
 Length:918         Min.   : 60.0   Length:918         Min.   :-2.6000   Length:918         Min.   :0.0000
 Class :character   1st Qu.:120.0   Class :character   1st Qu.: 0.0000   Class :character   1st Qu.:0.0000
 Mode  :character   Median :138.0   Mode  :character   Median : 0.6000   Mode  :character   Median :1.0000
                    Mean   :136.8                      Mean   : 0.8874                      Mean   :0.5534
                    3rd Qu.:156.0                      3rd Qu.: 1.5000                      3rd Qu.:1.0000
                    Max.   :202.0                      Max.   : 6.2000                      Max.   :1.0000
```

심장질환 데이터의 설명변수별 분포를 간단히 살펴보았다.

Outliers

```
# 결측값
heart.new1 <- heart[!heart[[5]]==0,]
str(heart.new1)
summary(heart.new1)
```

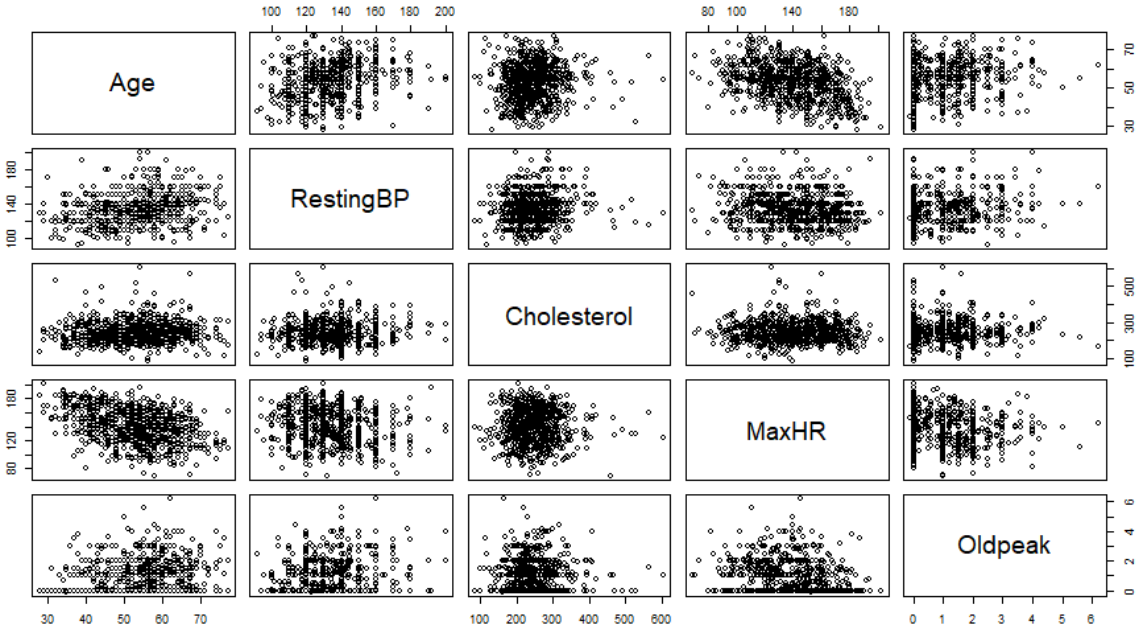
연속형 설명변수의 분포를 살펴보면 RestingBP와 Cholesterol에서 0의 값이 존재하는것을 알 수 있다. RestingBP와 Cholesterol이 0인 경우는 살아있는 생명체에서 나올 수 없는 특이값으로 간주하고 제거했다. 좌측부터 heart, heart.new의 summury 값으로인 0 이 제거된것을 확인할 수 있다.

⇒

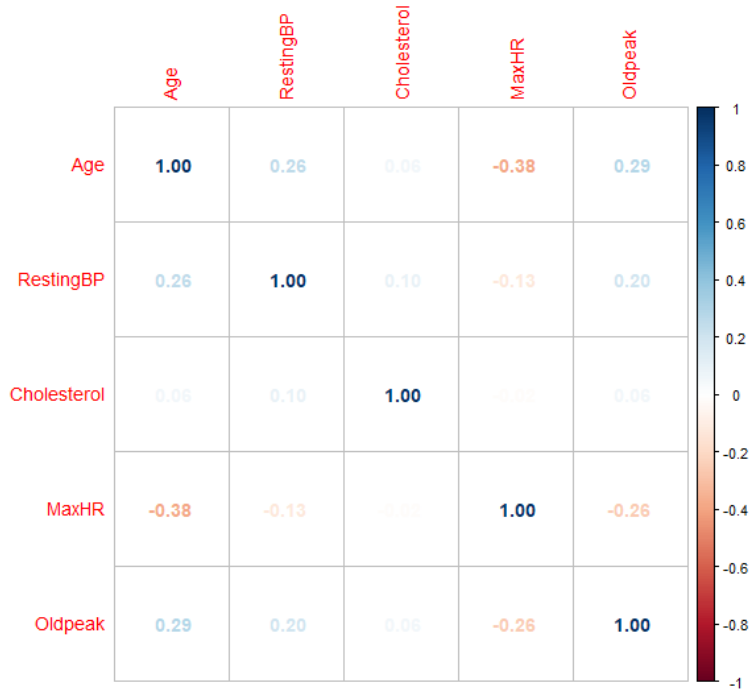
```
  RestingBP     Cholestero
 Min.   : 92   Min.   : 85
 1st Qu.:120   1st Qu.:207
 Median :130   Median :237
 Mean   :133   Mean   :244
 3rd Qu.:140   3rd Qu.:275
 Max.   :200   Max.   :603
```

Multicollinearity

연속형 설명변수들 간에 다중공선성이 있는지를 파악하기 위해서 산점도 행렬을 작성한 결과, 선형적인 상관관계는 보이지 않았다.



실제로 각 설명변수들 간의 상관계수를 계산하여 출력한 결과, 상관계수 값들이 작아 다중공선성이 없는 것으로 판단되었다.



Split training set / test set

```
set.seed(1)
index <- sample(1:nrow(heart), size = nrow(heart)*0.8)

train <- heart[index,]
test <- heart[-index,]
```

training set 80%, test set 20%로 설정하였다. 모델별로 요구하는 설명변수와 반응변수의 자료형이 다르므로, 각 모델마다 상이한 방법으로 train data와 test data를 나누었다.

Modeling

SVM(Support Vector Machine)

SVM은 분류 성능이 우수하기 때문에 높은 AUC 값이 예상된다.

우선, SVM의 경우 설명변수는 수치형, 반응변수가 범주형이어야 하므로 자료 유형을 변환해준다.

```
heart.new[heart.new$Sex=="F","Sex"]<-0
heart.new[heart.new$Sex=="M","Sex"]<-1
heart.new$Sex<-as.numeric(heart.new$Sex)
```

Sex에서 여자를 0, 남자를 1에 할당하였다. 나머지 문자형 변수인 ChestPainType, RestingECG, ExerciseAngina, ST_Slope도 이와 같이 수치형 변수로 변환하였다.

위 과정을 마친 후 이상치를 제거한 heart 데이터셋을 변수 heart.new에 할당하였다. 그러고 training 데이터와 test 데이터로 나눠 각각 train, test 변수에 할당하였다.

(1) Support Vector Classifier

SVM의 중요 인수는 gamma 값과 C 값이 있는데 gamma는 값이 높아질수록 경계면이 더 비선형적으로 변하고, C는 마진의 크기를 조정하여 커질수록 이상치의 존재 가능성을 작게본다. (이상치를 더 인정하지 않고 분류한다)

gamma와 C 모두 너무 낮으면 과소적합을, 너무 높은 값으로 설정되면 과대적합의 우려가 있으므로 e1071 패키지의 tune.svm 함수를 이용하여 분류가 잘되는 적절한 값을 찾는다.

```
> result1<-tune.svm(HeartDisease~.,data=train,cost=2^(0:4),kernel="linear")
> result1$best.parameters
  cost
1    1
```

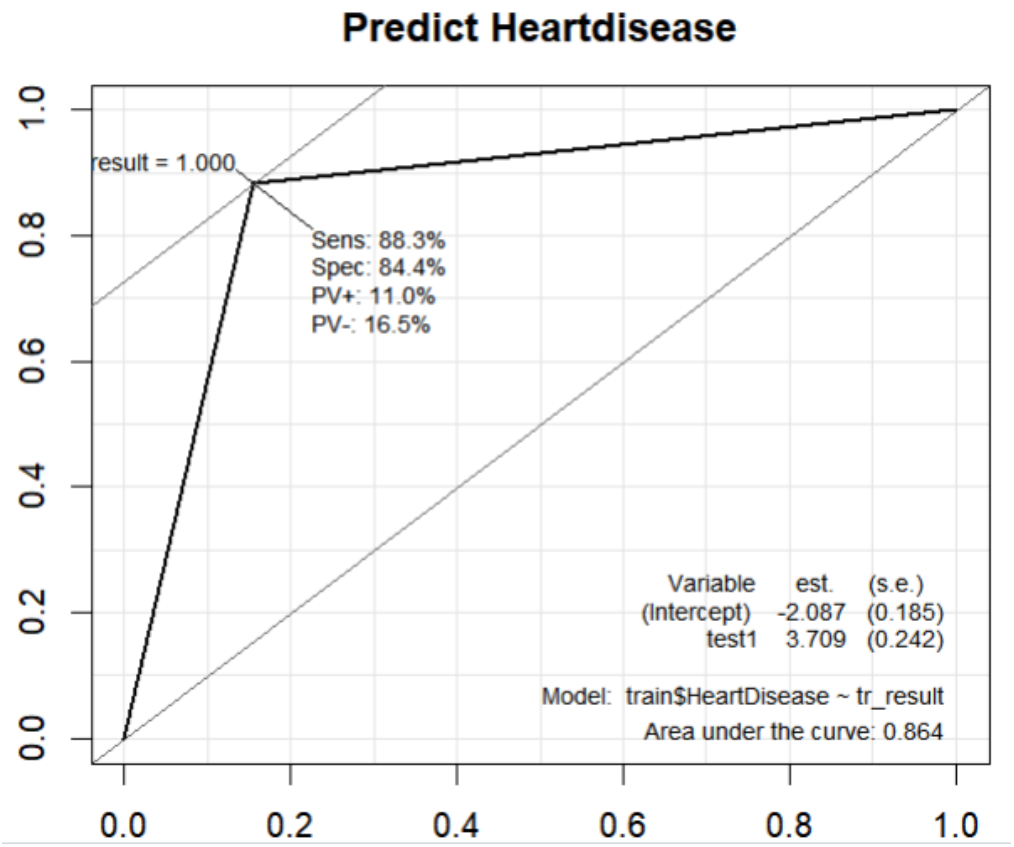
SVM 모델을 학습시킨 후 파라미터를 튜닝하면 kernel이 linear일 때 cost는 1이 적정치로 나온다.

```
> classifier = svm(formula = HeartDisease ~ .,data = train,type = 'C-classification',kernel = 'linear', gamma=0.0312
5, cost=1, cross=10)
> # Predicting the test set results
> svm.pred = predict(classifier, newdata = test[-12])
> # Making the Confusion Matrix
> cm = table(test[, 12], svm.pred)
> cm
   svm.pred
     0  1
  0 65 10
  1  7 68
> sum(cm[row(cm) == col(cm)])/sum(cm)
[1] 0.8866667
```

따라서 각 파라미터를 적용한 후, classifier 인수에 svm을 적합시키고 test 데이터로 예측한 결과와 비교한 오분류표를 만들어 보았다. svm 함수에 cross 인수를 이용하여 10-fold 방식 하에 구한 수치이다.

이제 training 데이터를 SVM으로 심장질환을 예측한 결과와 실제 심장질환 여부를 비교한 오분류표를 만들어 정확도를 계산하면 약 0.86이 나오고 ROC Curve를 그리면 아래와 같다.

```
> svm.result<-predict(classifier,train)
> t<-table(svm.result,train$HeartDisease);t

svm.result   0   1
         0 266  33
         1  49 248
> sum(t[row(t) == col(t)])/sum(t)
[1] 0.8624161
> library(Epi)
> par(oma=c(1,1,1,1),mar=c(1,3,3,1))
> ROC(test=tr_result,stat=train$HeartDisease,plot='ROC',AUC=T, main="Predict Heartdisease")
```



(2) Support Vector Machine

```
> result<-tune.svm(HeartDisease~.,data=train, gamma=2^(-5:0),cost=2^(0:4),kernel="radial")
> result$best.parameters
  gamma cost
3 0.125    1
```

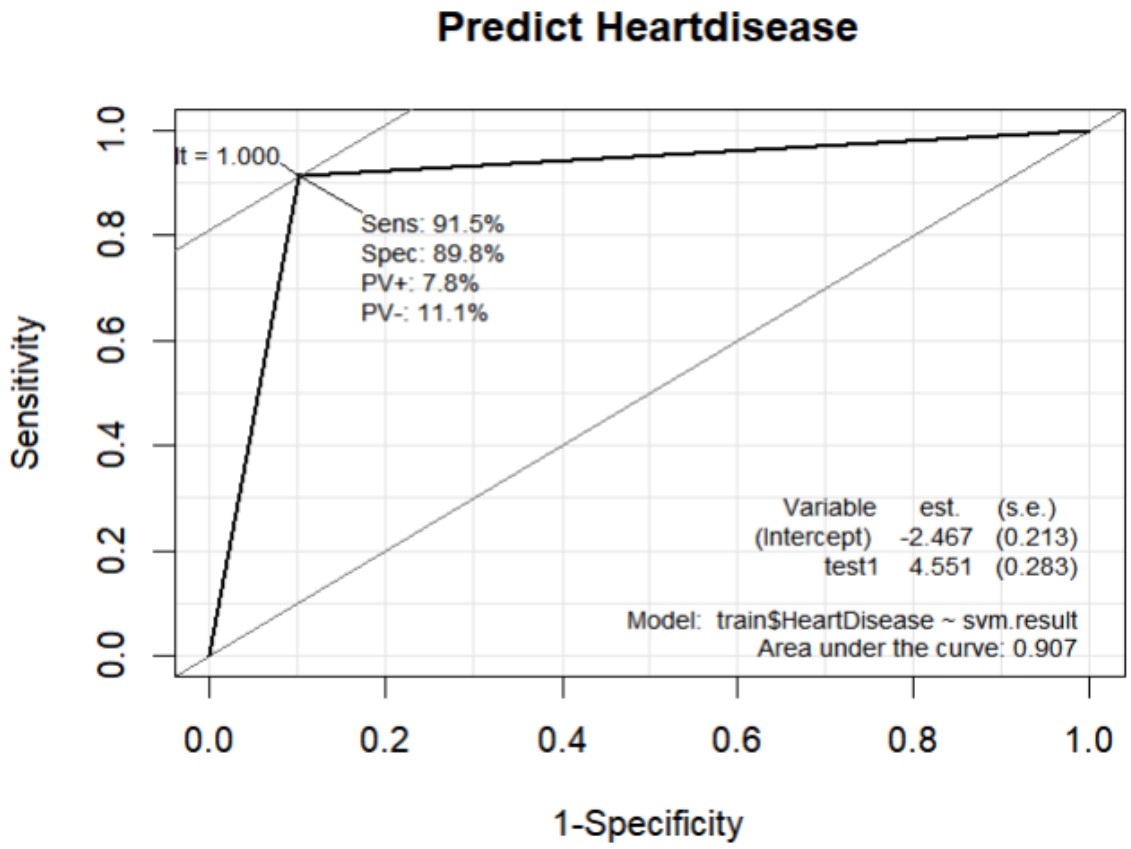
(1)과 마찬가지로 SVM 모델을 학습시킨 후, 파라미터를 튜닝하면 gamma 값은 0.125, cost 값은 1이 적정치로 나온다.

```
> classifier = svm(formula = HeartDisease ~ .,data = train,type = 'C-classification',kernel = 'radial', gamma=0.125,
 cost=1, cross=10)
> # Predicting the test set results
> svm.pred = predict(classifier, newdata = test[-12])
> # Making the Confusion Matrix
> cm = table(test[, 12], svm.pred)
> cm
   svm.pred
     0  1
  0 64 11
  1  7 68
> sum(cm[row(cm) == col(cm)])/sum(cm)
[1] 0.88
```

위와 같은 방식으로 오분류표를 만들면 0.88의 정확도가 나온다.

이제 training 데이터를 SVM으로 심장질환을 예측한 결과와 실제 심장질환 여부를 비교한 오분류표를 만들어 정확도를 계산하면 약 0.91이 나오고 ROC Curve를 그리면 아래와 같다.

```
> svm.result<-predict(classifier,train)
> t<-table(svm.result,train$HeartDisease);t

svm.result   0   1
         0 283  24
         1  32 257
> sum(t[row(t) == col(t)])/sum(t)
[1] 0.9060403
> library(Epi)
> par(oma=c(1,1,1,1),mar=c(1,3,3,1))
> ROC(test=tr_result,stat=train$HeartDisease,plot='ROC',AUC=T, main="Predict Heartdisease")
```



AUC 값의 결과 heart 데이터셋은 SVM으로 분류할 때, kernel은 linear보다 radial이 적합한 것으로 보이며 약 0.91로 분류 정확도가 높다.

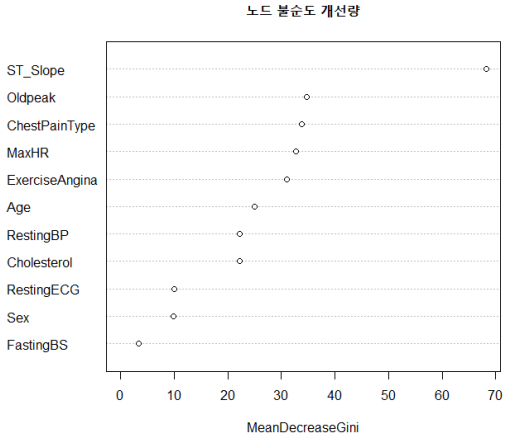
Random Forest

Random Forest는 Decision Tree의 약점을 보완하고 Overfitting을 피할 수 있다는 장점을 가지고 있기 때문에 좋은 값을 추정할 수 있을거라 예상한다.

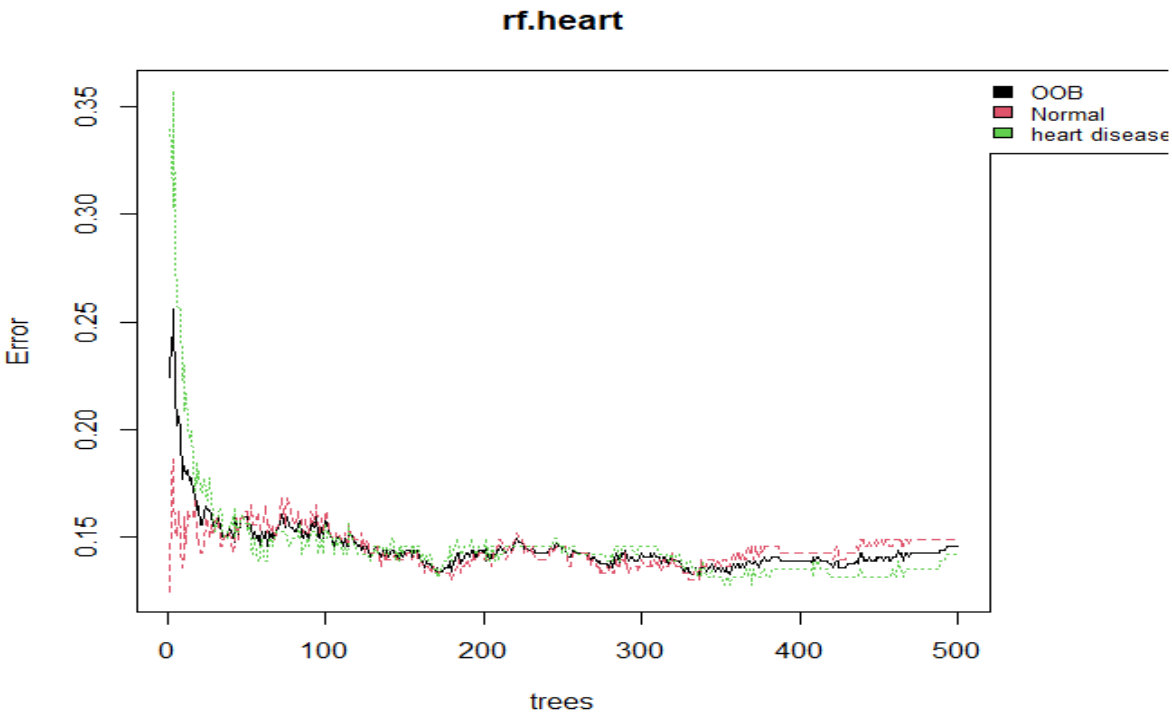
(1) Random Forest Modeling

```
> (rf.heart<- randomForest(HeartDisease ~., data=train))

Call:
 randomForest(formula = HeartDisease ~ ., data = train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 14.6%
Confusion matrix:
              Normal heart disease class.error
Normal           268            47   0.1492063
heart disease     40           241   0.1423488
```

이 모델의 오분류율은 14.6%이다. 정상을 268명으로 예측했지만 47명을 심장질환으로 오분류 하였고, 심장질환을 241명으로 예측했지만 40명을 정상으로 오분류하였다.

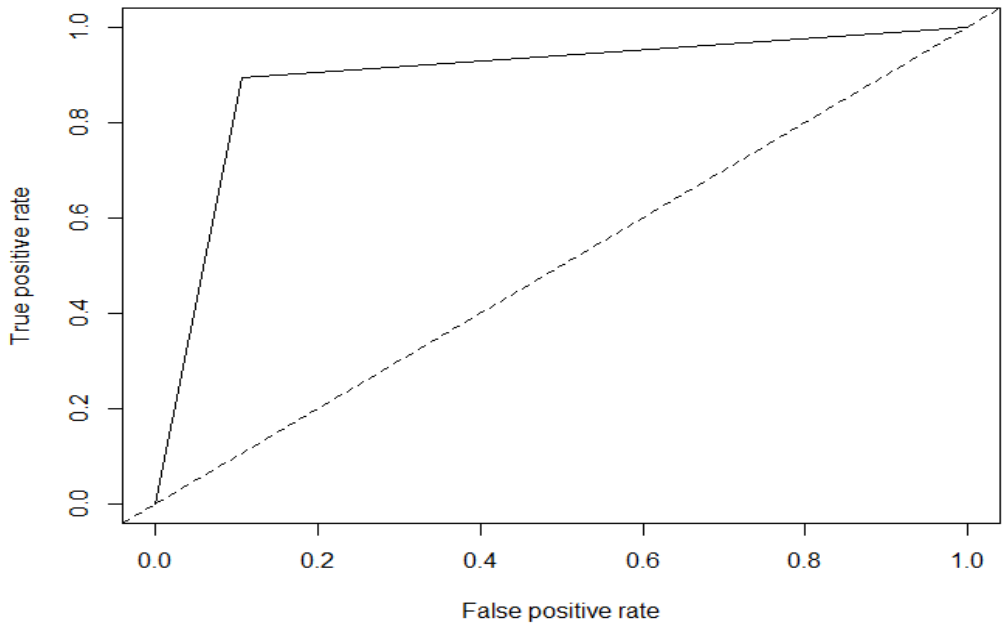


변수중요도는 다음과 같으며 이는 변수가 정확도와 노드불순도 개선에 얼마만큼 기여하는지로 정해진다. 변수중요도는 ST_Slope > Oldpeak > ChestPainType > MaxHR > ExerciseAngina > Age > RestingBP > Cholesterol > RestingECG > Sex > FastingBS 순이다.



plot(rf.heart)을 실행하면 다음과 같은 그래프가 나온다.

모델 훈련간 사용되지 않은 데이터를 사용한 에러추정치인 OOB와, Normal, Heart Disease 세가지 경우간의 에러 추정치의 차이가 크지 않다.



생성된 ROC Curve는 위와 같으며 AUC값은 0.8933(89.33%)로 모델의 성능은 나쁘지 않다.

```
# ROC curve
library(ROCR)
rf.pred = predict(rf.heart,test[,-12],type = "class")
roc.pred =prediction(as.numeric(rf.pred),as.numeric(test[,12]))
plot(performance(roc.pred,"tpr","fpr"))
abline(a=0,b=1,lty=2,col="black")

# AUC
performance(roc.pred,"auc")@y.values
```

```
> performance(roc.pred,"auc")@y.values
[[1]]
[1] 0.8933333
```

(2) Random forest Cross-Validation

```
set.seed(1)
K = 10
R = 3
cv <- cvFolds(NROW(heart.new1), K=K, R=R)

grid <- expand.grid(ntree=c(50, 100, 150, 200), mtry=c(1:11))
rf.result <- foreach(g=1:NROW(grid), .combine=rbind) %do% {
  foreach(r=1:R, .combine=rbind) %do% {
    foreach(k=1:K, .combine=rbind) %do% {
      validation_idx <- cv$subsets[which(cv$which == k), r]
      train <- heart.new1[-validation_idx, ]
      validation <- heart.new1[validation_idx, ]
      # 모델 훈련
      train$HeartDisease = as.factor(train$HeartDisease)
      m <- randomForest(HeartDisease ~.,
                        data=train,
                        ntree=grid[g, "ntree"],
                        mtry=grid[g, "mtry"])
      # 예측
      predicted <- predict(m, newdata=validation)
      # 성능 평가
      precision <- sum(predicted == validation$HeartDisease) / NROW(predicted)
      return(data.frame(g=g, precision=precision))
    }
  }
}
```

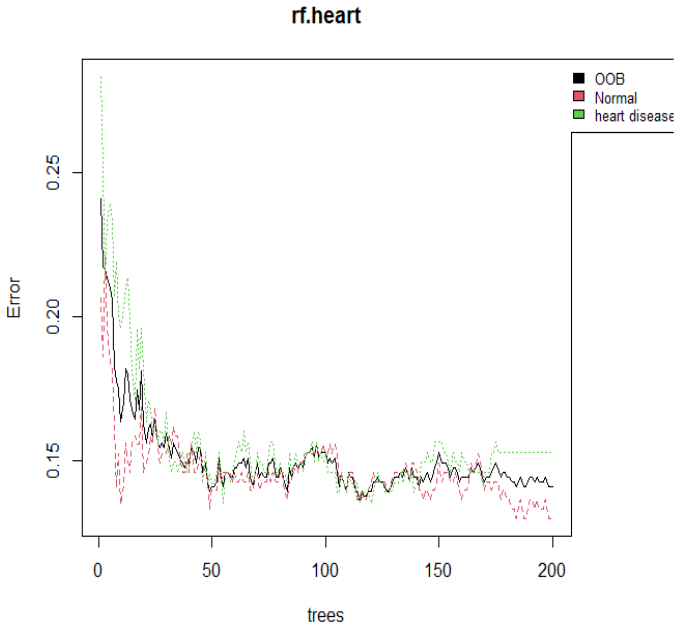
ntree가 너무 커지면 모델의 값이 유사해지므로 ntree는 50,100,150,200 중에서 선택하고 mtry는 1 1 부터 11까지의 값을 선택하였다. 이를 실행하여 나오는 rf.result값의 결과를 살펴보기 위하여 ddply함수로 mean_precision을 관찰하면 g=8일때의 값이 가장 크므로 grid [8,]을 관찰하면 최적의 ntree와 mtry의 값을 얻을 수 있다.

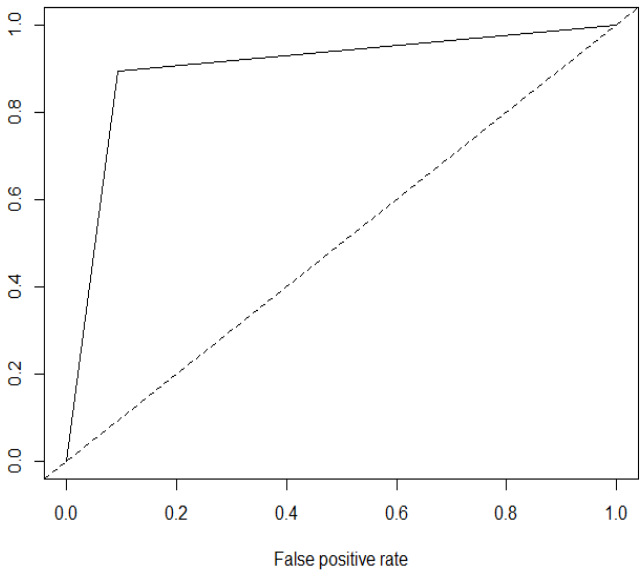
```
> ddply(rf.result, .(g), summarize, mean_precision=mean(precision))
    g mean_precision
1   1      0.8552012
2   2      0.8578318
3   3      0.8605285
4   4      0.8601141
5   5      0.8583483
6   6      0.8614715
7   7      0.8618859
8   8      0.8672553
9   9      0.8632132
10 10      0.8619339
11 11      0.8659099

> grid[8,]
  ntree mtry
8   200    2
```

따라서 ntree = 200 , mtry =2 로 다시 모델링하면 더 좋은 결과를 얻을 수 있을것으로 예상된다.

이 값으로 위의 과정을 다시 반복하여 AUC값을 구할 수 있다.

```
(rf.heart <- randomForest(HeartDisease ~., data=train,ntree=200,mtry=2))
```






```
> performance(roc.pred,"auc")@y.values
[[1]]
[1] 0.8933333
```

구해진 AUC값은 0.8933으로 이전과 동일하다. 따라서 기존의 default값으로 조정된 ntree, mtry 값과 교차검증으로 찾은 값의 AUC의 차이는 없다.

Boosting

gbm()함수를 사용한 generalized boosted classification model을 적합하기 위해 설명변수의 자료형을 범주형으로 변환하였다.

```
> for (i in c(2,3,7,9,11)){
+   heart.new[[i]] <- as.factor(heart.new[[i]])
+ }
> str(heart.new)
'data.frame':   746 obs. of  12 variables:
 $ Age           : int  40 49 37 48 54 39 45 54 37 48 ...
 $ Sex           : Factor w/ 2 levels "F","M": 2 1 2 1 2 2 1 2 2 1 ...
 $ ChestPainType : Factor w/ 4 levels "ASY","ATA","NAP",..: 2 3 2 1 3 3 2 2 1 2 ...
 $ RestingBP     : int  140 160 130 138 150 120 130 110 140 120 ...
 $ Cholesterol   : int  289 180 283 214 195 339 237 208 207 284 ...
 $ FastingBS     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ RestingECG    : Factor w/ 3 levels "LVH","Normal",..: 2 2 3 2 2 2 2 2 2 2 ...
 $ MaxHR         : int  172 156 98 108 122 170 170 142 130 120 ...
 $ ExerciseAngina: Factor w/ 2 levels "N","Y": 1 1 1 2 1 1 1 1 2 1 ...
 $ Oldpeak       : num  0 1 0 1.5 0 0 0 0 1.5 0 ...
 $ ST_Slope      : Factor w/ 3 levels "Down","Flat",..: 3 2 3 2 3 3 3 3 2 3 ...
 $ HeartDisease  : int  0 1 0 1 0 0 0 0 1 0 ...
```

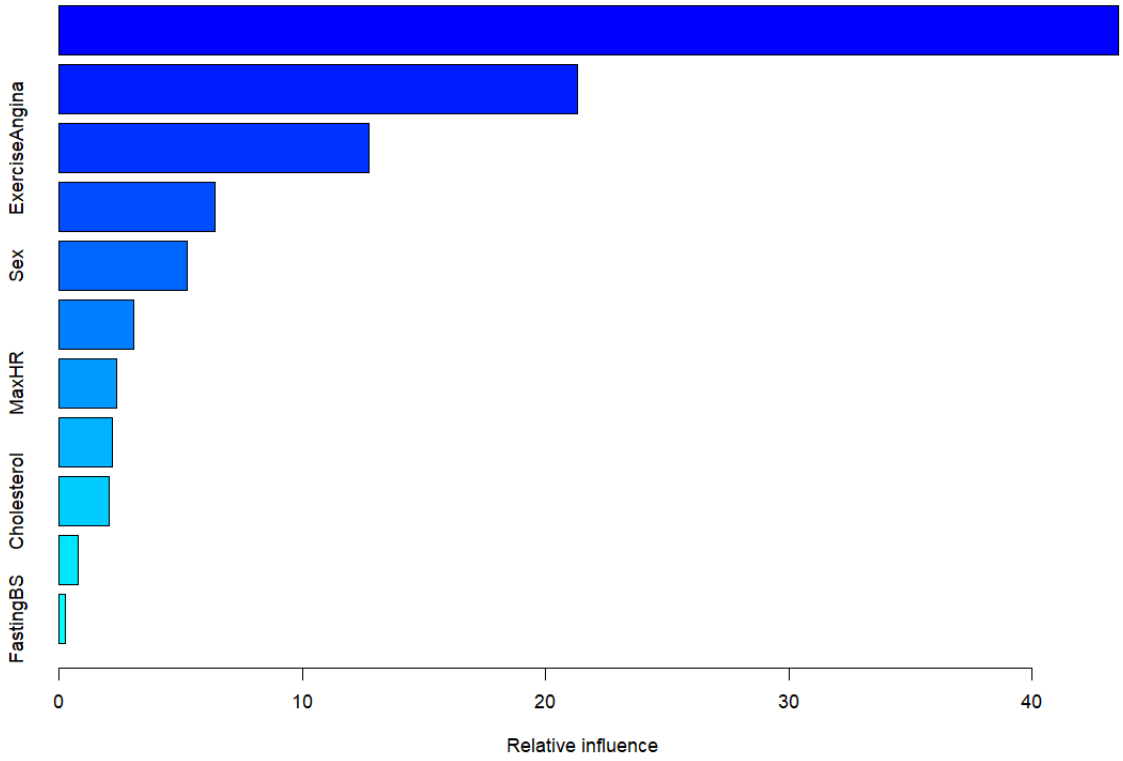
(1) Boosting Model(parameters : default values)

첫 번째로, 훈련 세트를 이용하여 boosting model을 적합하였다.

```
> set.seed(1)
> library(gbm)
> boost.heart <- gbm(HeartDisease~., data=train,
+                    distribution="bernoulli", cv.folds=10)
> boost.heart
gbm(formula = HeartDisease ~ ., distribution = "bernoulli",
    data = train, cv.folds = 10)
A gradient boosted model with bernoulli loss function.
100 iterations were performed.
The best cross-validation iteration was 90.
There were 11 predictors of which 11 had non-zero influence.
```

해당 모델은 반응 변수의 class가 0또는 1인 이진 분류에 해당하므로 distribution을 "bernoulli"로 지정하였고, 10-folds CV를 사용하기 위해 cv.folds 인수를 10으로 지정하였다. 나머지 parameter들은 기본값을 사용하였다. 이 모델의 feature importance는 아래와 같다.

```
> summary(boost.heart, new=TRUE)
                               var    rel.inf
ST_Slope                 ST_Slope 43.5351699
ChestPainType       ChestPainType 21.3041656
ExerciseAngina ExerciseAngina 12.7249622
Oldpeak               Oldpeak  6.4082807
Sex                       Sex  5.2654112
Age                       Age  3.0880523
MaxHR                   MaxHR  2.3533314
RestingBP           RestingBP  2.2096344
Cholesterol       Cholesterol  2.0504041
RestingECG         RestingECG  0.7964405
FastingBS           FastingBS  0.2641477
```



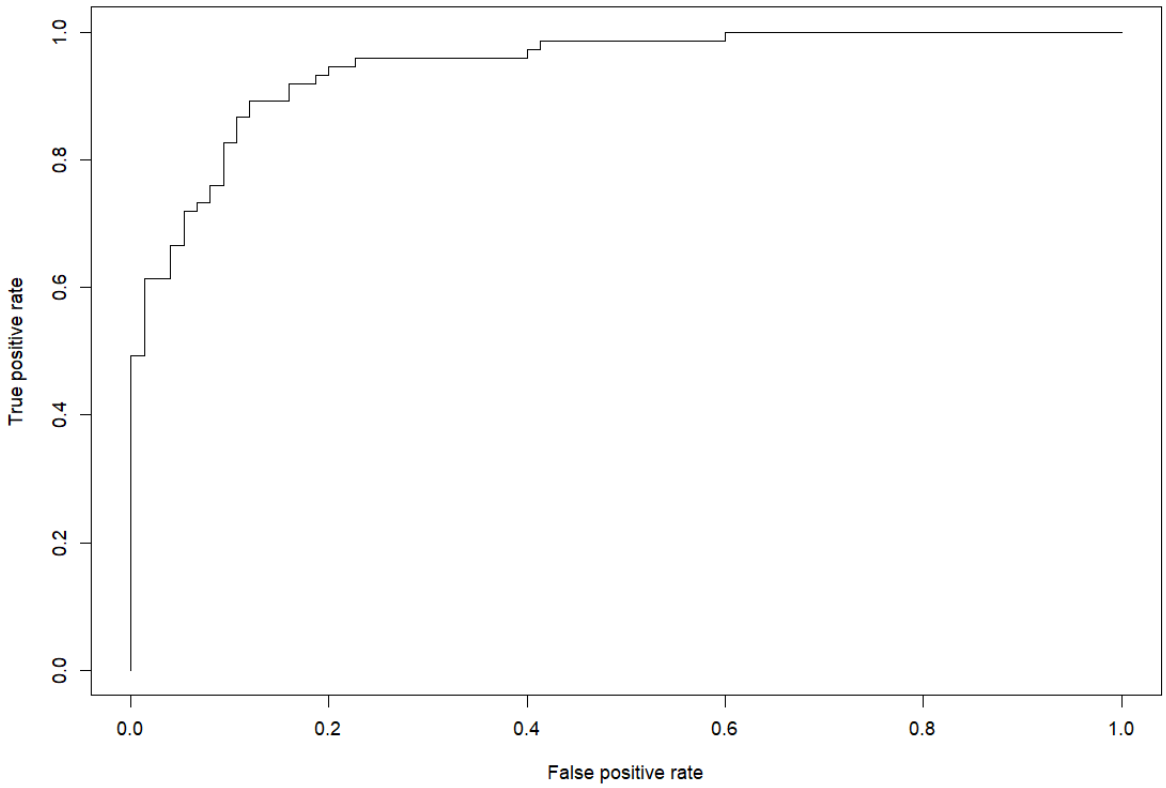
위의 출력 결과를 보면 ST_Slope의 변수 중요도가 약 43.54으로 가장 높았고, FastingBS가 약 0.26로 가장 낮았다.

다음으로 predict.gbm() 함수를 사용하여 test data에 대한 모델의 예측 결과를 구하고, 이를 실제 test data와 비교하는 confusion matrix를 작성하였다.

```
> heartdisease <- test$HeartDisease
> boost.pred <- predict.gbm(boost.heart, test, ntrees=100, type="response")
Using 90 trees...

> boost.prediction <- rep(0, length(boost.pred))
> boost.prediction[boost.pred>0.5] <- 1
> table(heartdisease, boost.prediction)
            boost.prediction
heartdisease  0  1
           0 66  9
           1  8 67
```

위의 confusion matrix에서 정확도는 약 0.8867를 얻을 수 있었고, ROC-Curve는 아래와 같다.

```
> library(ROCR)
> pred <- prediction(boost.pred, heartdisease)
> plot(performance(pred, "tpr", "fpr"))
```



위 ROC-Curve에 대한 AUC값을 구한 결과, 그 값은 약 0.9454로 다른 방법으로 적합한 모델들 보다 성능이 좋은 것을 알 수 있었다.

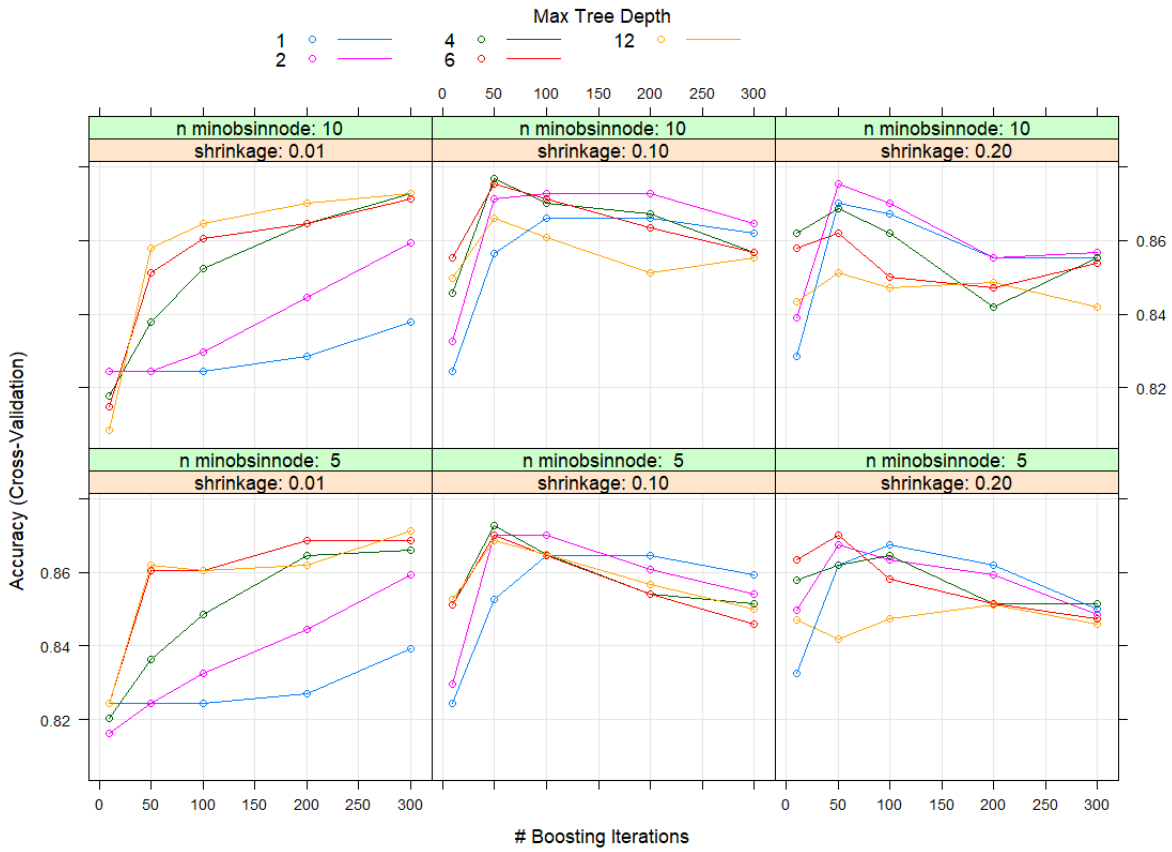
```
> auc.boost <- performance(pred, measure="auc")
> attributes(auc.boost)$y.values   # AUC = 0.9454
[[1]]
[1] 0.9454222
```

(2) Hyperparameter tuning

Boosting에서 튜닝이 필요한 파라미터들은 크게 3가지가 있다. 첫 번째로, 트리의 개수를 의미하는 n.trees이다. 이 값이 너무 크면 overfitting의 우려가 있기 때문에 CV를 통해서 결정해야 한다. 두 번째는 각 트리의 split 개수를 의미하는 interaction.depth이고, 마지막 세 번째는 boosting learning 속도를 제어하는 shrinkage이다.

파라미터 튜닝을 위해서 하이퍼파라미터들의 범위를 직접 지정하여 성능을 비교하는 Grid Search를 이용하였다. n.trees는 10, 50, 100, 200, 300의 값들을, interaction.depth는 1, 2, 4, 6, 12를, shrinkage는 0.01, 0.1, 0.2의 값들을 비교하였다.

```r
# grid search - hyperparemeter tuning
set.seed(1)
library(caret)
heart.new[[12]] <- as.factor(heart.new[[12]])
train.data <- train[,-12]
train.classes <- train$HeartDisease
traincontrol <- trainControl(method="cv", number=10, returnResamp="all", search="grid")
tunegrid <- expand.grid(n.trees=c(10,50,100,200,300),
                        interaction.depth=c(1,2,4,6,12),
                        shrinkage=c(0.01,0.1,0.2),
                        n.minobsinnode=c(5,10))
boost.grid <- caret::train(HeartDisease~., data=heart.new, method="gbm",
                           tuneGrid=tunegrid, trControl=traincontrol, verbose=FALSE)
plot(boost.grid)
```



Grid Search를 통해 구한 Accuracy가 가장 높은 모델의 parameter 값들은 besttune 속성에 저장되어 있다. 그 값들을 확인한 결과 n.trees=50, interaction.depth=4, shrinkage=0.1을 얻을 수 있었다.

```
> boost.grid$bestTune
   n.trees interaction.depth shrinkage n.minobsinnode
77      50                 4       0.1             10
```

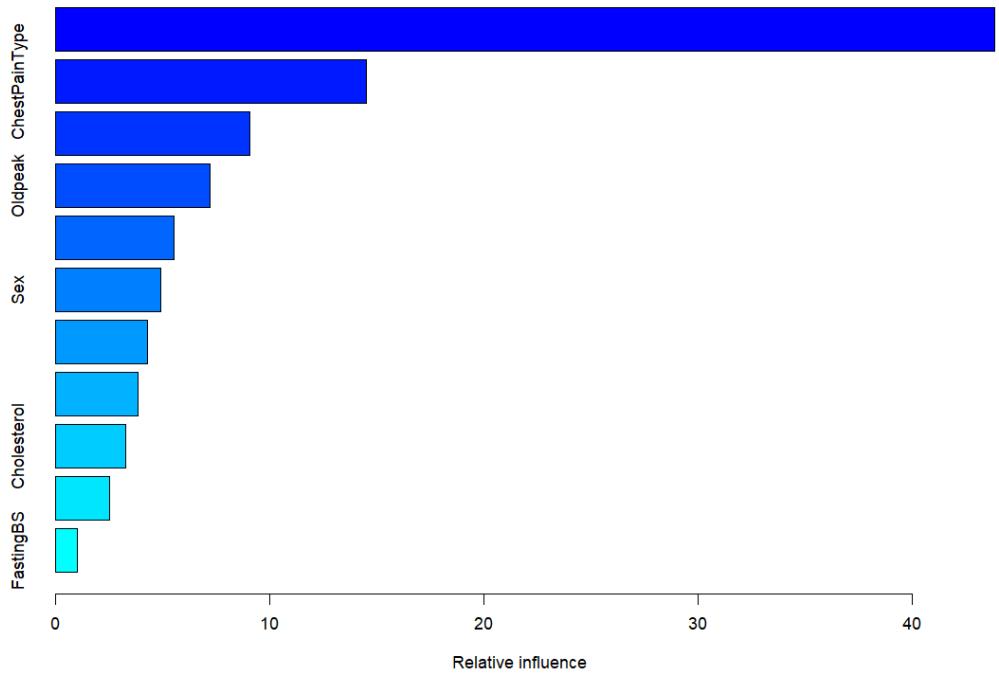
(3) Best Model

(1)에서와 동일한 방법으로, 위에서 구한 최적의 파라미터 값들을 사용하여 모델을 적합시킨 결과는 아래와 같다.

```
> # Best Model
> set.seed(1)
> best.boost.heart <- gbm(HeartDisease~., data=train,
+                     distribution="bernoulli", cv.folds=10,
+                     n.trees=50, interaction.depth=4, shrinkage=0.1)
> best.boost.heart
gbm(formula = HeartDisease ~ ., distribution = "bernoulli",
    data = train, n.trees = 50, interaction.depth = 4, shrinkage = 0.1,
    cv.folds = 10)
A gradient boosted model with bernoulli loss function.
50 iterations were performed.
The best cross-validation iteration was 47.
There were 11 predictors of which 11 had non-zero influence.
```

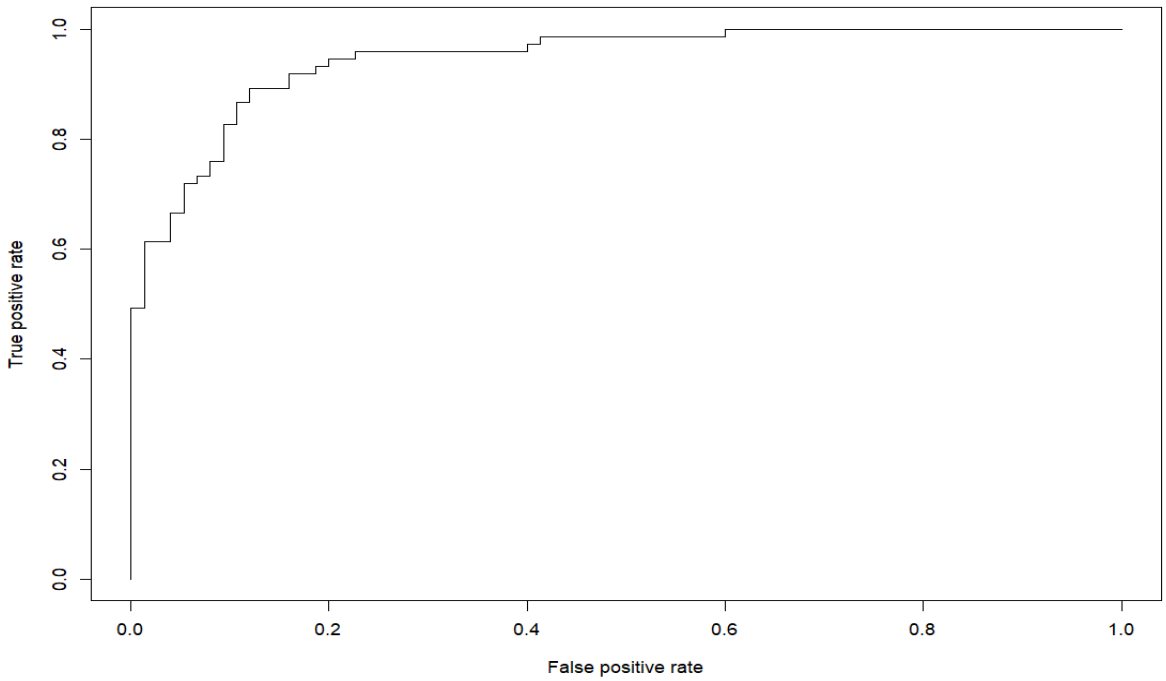
위 모델의 변수 중요도를 살펴보면 (1)에서의 모형과 비교했을 때 ST_Slope의 중요도 값은 약 43.82로 거의 동일했고, FastingBS의 중요도는 1.02로 (1)에서의 모형과 비교했을 때 증가한 것을 알 수 있었다.

```
> summary(best.boost.heart)
                             var   rel.inf
ST_Slope               ST_Slope 43.817207
ChestPainType     ChestPainType 14.503501
MaxHR                     MaxHR  9.054951
Oldpeak                 Oldpeak  7.199548
RestingBP             RestingBP  5.546115
Sex                         Sex  4.899882
ExerciseAngina   ExerciseAngina  4.289303
Age                         Age  3.847837
Cholesterol         Cholesterol  3.286142
RestingECG           RestingECG  2.534591
FastingBS             FastingBS  1.020923
```



Test data를 사용하여 구한 predict.gbm()의 결과와 실제 test data를 이용하여 ROC-Curve를 그리고, AUC 값을 구한 결과는 아래와 같다.

```
> best.boost.pred <- predict.gbm(best.boost.heart, test, ntrees=50, type="response")
Using 47 trees...

> best.pred <- prediction(best.boost.pred, heartdisease)
> plot(performance(pred, "tpr", "fpr"))
```




```
> best.auc.boost <- performance(best.pred, measure="auc")
> attributes(best.auc.boost)$y.values    # AUC = 0.9461
[[1]]
[1] 0.9461333
```

그 결과, best model에서의 AUC값은 약 0.9461로 (1)에서 구한 모델과 비교했을 때 더 좋은 성능을 보이고 있음을 알 수 있었다.

Conclusion

심장질환 데이터셋을 이용하여 SVM, Random Forest, Boosting 총 세 가지의 모델들을 분석해보았다. AUC 값은 SVM, Random Forest가 각각 0.907, 0.9로 근소한 차이를 보였으며 Boosting은 0.9454로 다른 모델들보다 성능이 우수했다. 랜덤포레스트는 train data를 추가해도 모델 성능 개선이 어렵다는 단점이 존재하여 10-fold CV를 이용한 AUC 값과 기존의 값에 큰 차이가 없었다. SVM의 경우 다른 모델에 비해 parameter 조정시간이 오래 걸리고 결과의 해석이 어려웠다. boosting은 이상치에 취약하지만 제거함으로써 단점을 보완했고 SVM과 마찬가지로 학습 속도에 시간이 걸리나 가장 높은 예측 정확도를 보였다.

따라서 심장질환 여부를 예측하기에 boosting이 가장 적합한 모델로 판단되었다.

역할

공통 : 보고서 작성

조장 이석희 : Random Forest, EDA 및 시각화

양승민 : Support Vector Machine

박예준 : Boosting Classification Model