GPH2338_Project_Heart_Disease

Chenggang Lyu

3/28/2021

Data clean and Descriptive Statistics

The dataset contains 303 rows/observations and 14 variables in total.

```
heart_ <- read.csv("heart.csv")

dim(heart_)
```

```
## [1] 303  14
```

```
head(heart_)
```

```
##   age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 1  63   1  3      145  233   1       0     150     0     2.3     0  0    1
## 2  37   1  2      130  250   0       1     187     0     3.5     0  0    2
## 3  41   0  1      130  204   0       0     172     0     1.4     2  0    2
## 4  56   1  1      120  236   0       1     178     0     0.8     2  0    2
## 5  57   0  0      120  354   0       1     163     1     0.6     2  0    2
## 6  57   1  0      140  192   0       1     148     0     0.4     1  0    1
##   target
## 1      1
## 2      1
## 3      1
## 4      1
## 5      1
## 6      1
```

```
summary(heart_)
```

```
##       age             sex               cp           trestbps
##  Min.   :29.00   Min.   :0.0000   Min.   :0.000   Min.   : 94.0
##  1st Qu.:47.50   1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:120.0
##  Median :55.00   Median :1.0000   Median :1.000   Median :130.0
##  Mean   :54.37   Mean   :0.6832   Mean   :0.967   Mean   :131.6
##  3rd Qu.:61.00   3rd Qu.:1.0000   3rd Qu.:2.000   3rd Qu.:140.0
##  Max.   :77.00   Max.   :1.0000   Max.   :3.000   Max.   :200.0
##       chol            fbs            restecg          thalach
##  Min.   :126.0   Min.   :0.0000   Min.   :0.0000   Min.   : 71.0
##  1st Qu.:211.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:133.5
##  Median :240.0   Median :0.0000   Median :1.0000   Median :153.0
##  Mean   :246.3   Mean   :0.1485   Mean   :0.5281   Mean   :149.6
##  3rd Qu.:274.5   3rd Qu.:0.0000   3rd Qu.:1.0000   3rd Qu.:166.0
##  Max.   :564.0   Max.   :1.0000   Max.   :2.0000   Max.   :202.0
##      exang           oldpeak         slope             ca
##  Min.   :0.0000   Min.   :0.00   Min.   :0.000   Min.   :0.0000
```

```
##  1st Qu.:0.0000   1st Qu.:0.00   1st Qu.:1.000   1st Qu.:0.0000
##  Median :0.0000   Median :0.80   Median :1.000   Median :0.0000
##  Mean   :0.3267   Mean   :1.04   Mean   :1.399   Mean   :0.7294
##  3rd Qu.:1.0000   3rd Qu.:1.60   3rd Qu.:2.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :6.20   Max.   :2.000   Max.   :4.0000
##       thal           target
##  Min.   :0.000   Min.   :0.0000
##  1st Qu.:2.000   1st Qu.:0.0000
##  Median :2.000   Median :1.0000
##  Mean   :2.314   Mean   :0.5446
##  3rd Qu.:3.000   3rd Qu.:1.0000
##  Max.   :3.000   Max.   :1.0000
```

For the follow analysis, some algorithms require the categorical variables to be in the form of factor. This step convert those categorical variables into factors. Since the dataset is pre-processed, there is not missing value.

```
# convert categorical variables into factors: sex, cp, fbs, restecg, exang, slope, ca, thal, target
heart_data <- heart_ %>%
  mutate(target = as.factor(target)) %>%
  mutate(sex = as.factor(sex)) %>%
  mutate(cp = as.factor(cp)) %>%
  mutate(fbs = as.factor(fbs)) %>%
  mutate(restecg = as.factor(restecg)) %>%
  mutate(exang = as.factor(exang)) %>%
  mutate(slope = as.factor(slope)) %>%
  mutate(ca = as.factor(ca)) %>%
  mutate(thal = as.factor(thal))
sum(is.na(heart_data))
```

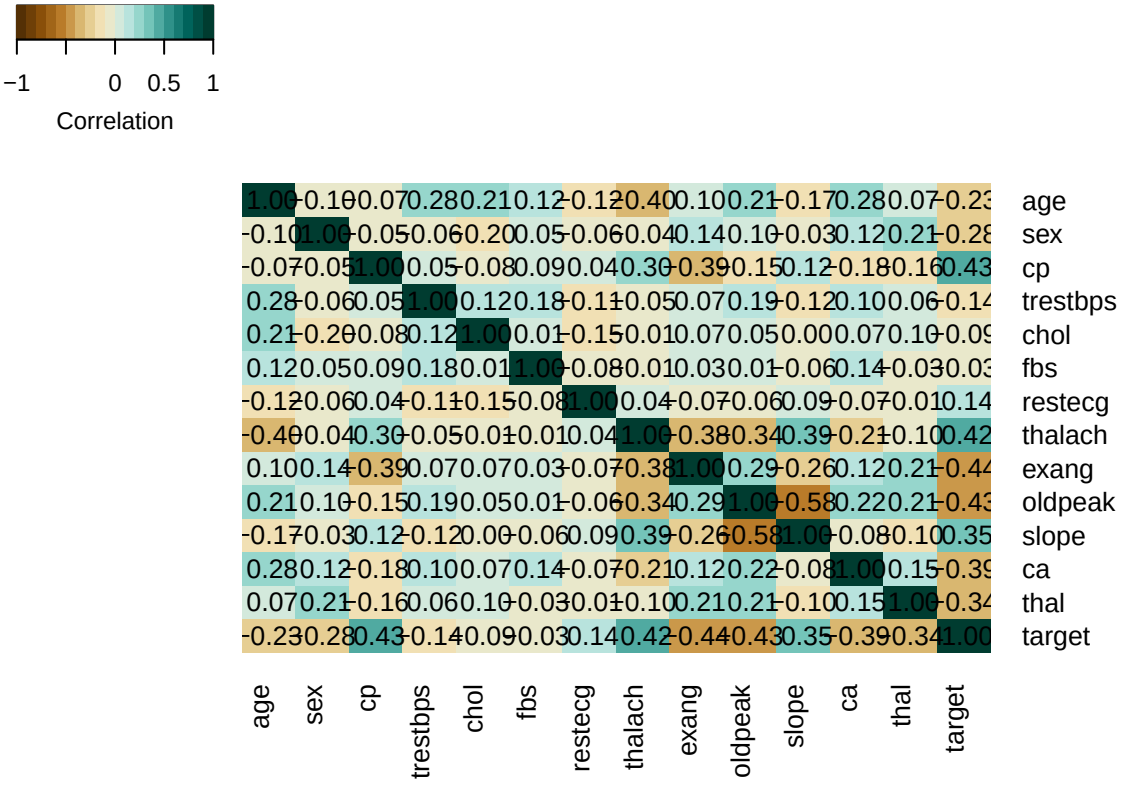
```
## [1] 0
```

EDA – visualization:

(a) heatmap for the correlation matrix

```
library(gplots)
library(RColorBrewer)

cor_mat <- cor(heart_)

# display.brewer.pal(10, "BrBG")
palettte_ <- brewer.pal(10, "BrBG")
color_ <- colorRampPalette(palettte_)(20)

text_ <- format(round(cor_mat, 2))

heatmap.2(cor_mat,
    trace="none",
    col=color_,
    density.info = "none",
    key.xlab ='Correlation',
```

```
    key.title = "",
    cexRow = 1, cexCol = 1,
    Rowv = F, Colv = F,
    margins = c(5, 5),
    cellnote = text_,
    notecol='black'
)
```

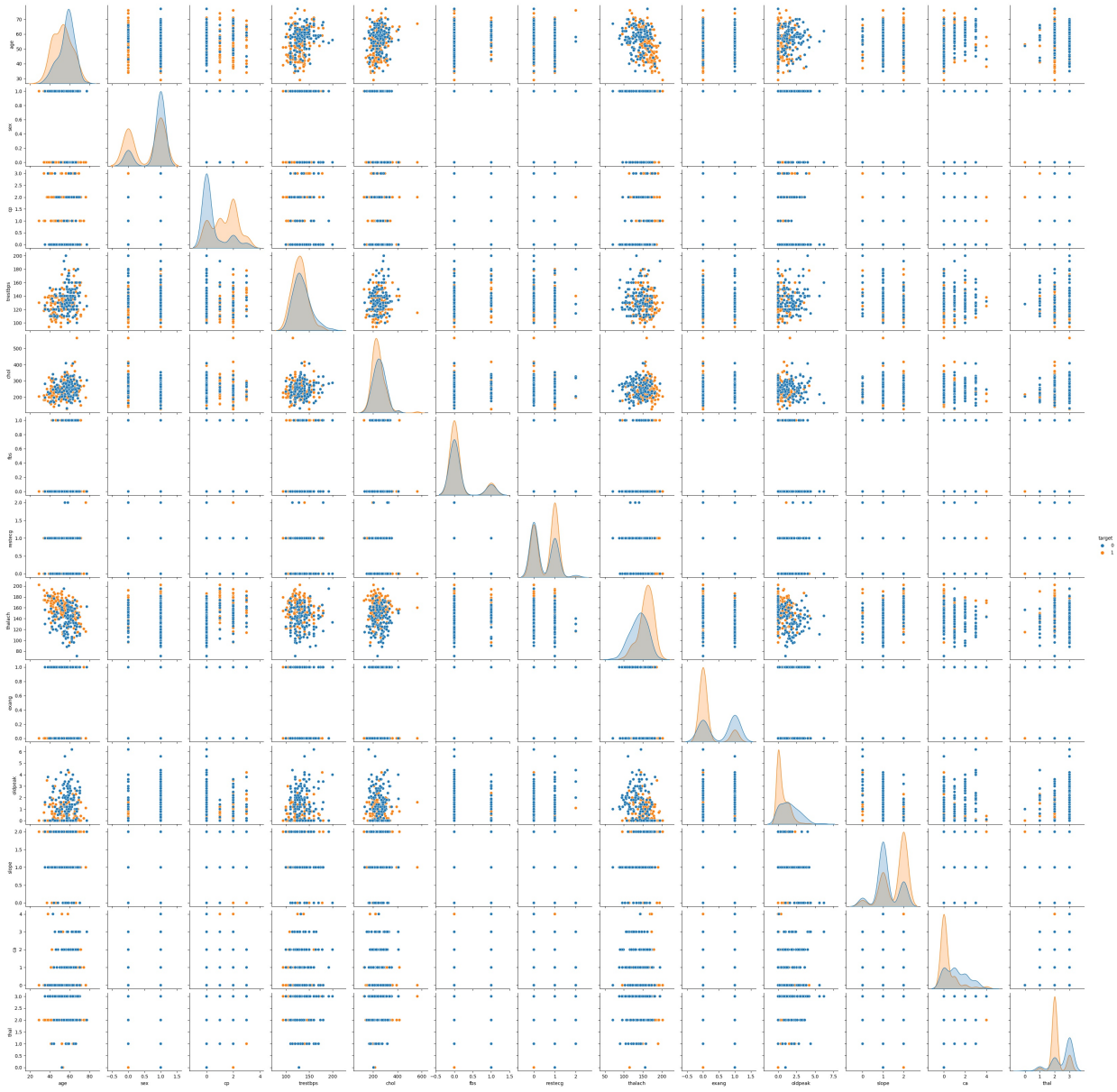


(b) pairplot in a glance

```
library("reticulate")
python_exe_path <- "/Users/missshihonghowru/anaconda3/envs/r-test/bin/python"
use_python(python_exe_path)

sns <- import('seaborn')
plt <- import('matplotlib.pyplot')
pd <- import('pandas')
```

```
sns$pairplot(r_to_py(heart_data), hue = 'target')
#display the plot
plt$savefig('pairplot.jpg')
```

Output image:

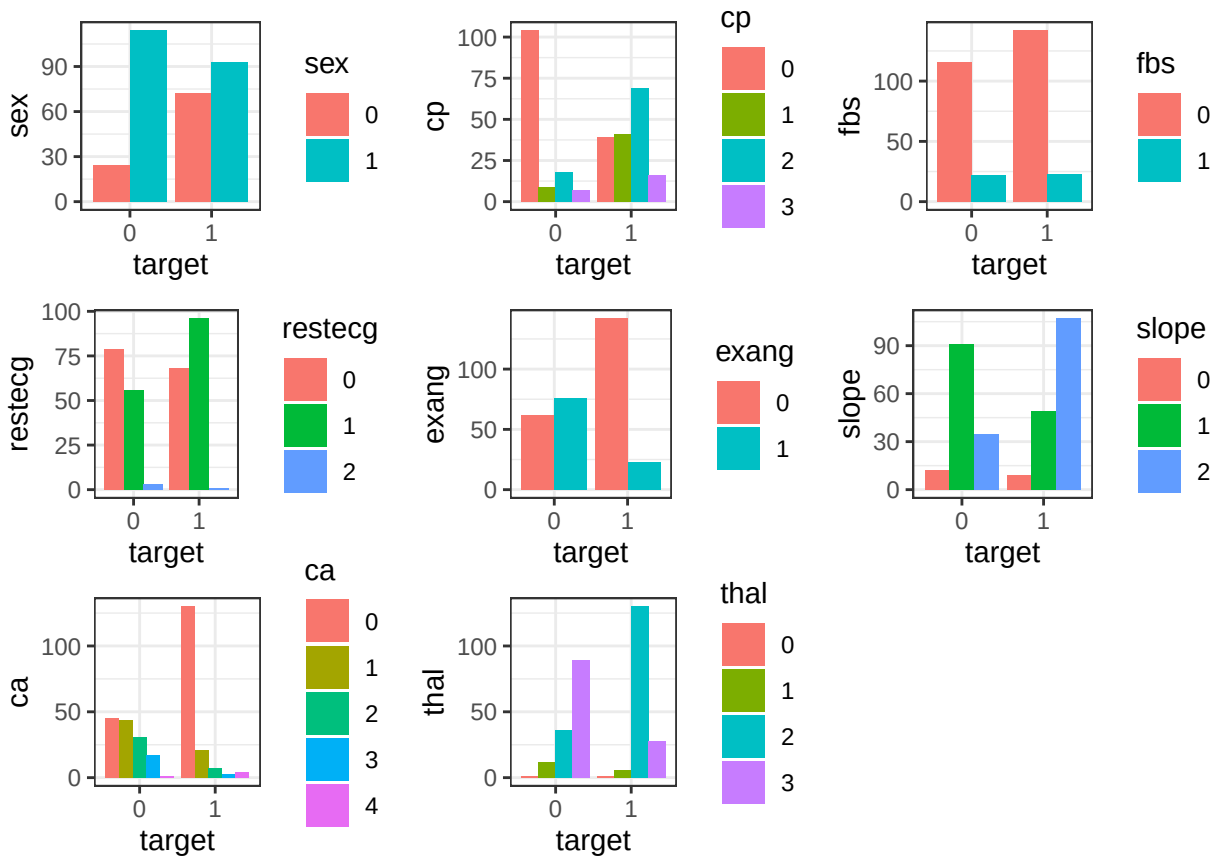


(c) Barplots for categorical variables vs. target

```
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

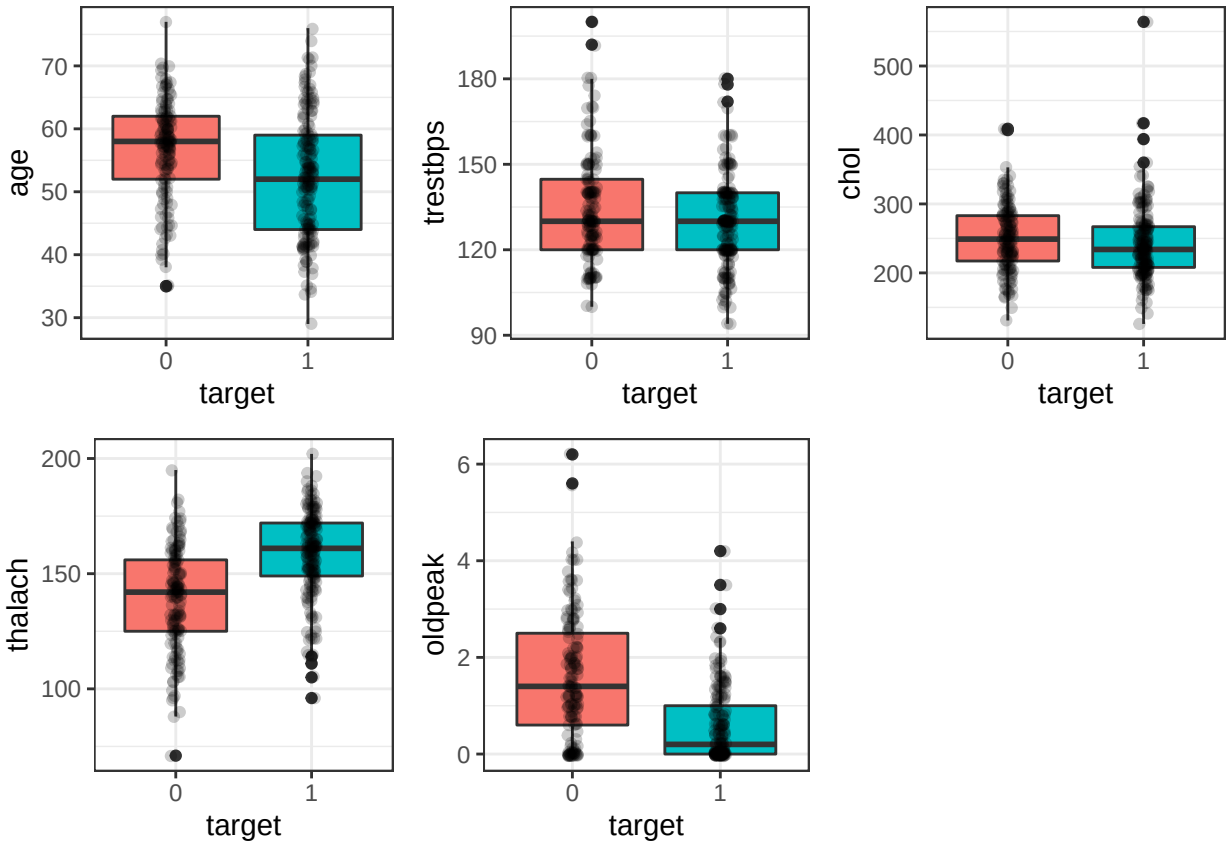
```r
# sex, cp, fbs, restecg, exang, slope, ca, thal
p1 <- ggplot(heart_data, aes(x = target, fill = sex)) +
      geom_bar(position = "dodge") + theme_bw() + labs(y="sex")
p2 <- ggplot(heart_data, aes(x = target, fill = cp)) +
      geom_bar(position = "dodge") + theme_bw() + labs(y="cp")
p3 <- ggplot(heart_data, aes(x = target, fill = fbs)) +
      geom_bar(position = "dodge") + theme_bw() + labs(y="fbs")
p4 <- ggplot(heart_data, aes(x = target, fill = restecg)) +
      geom_bar(position = "dodge") + theme_bw() + labs(y="restecg")
p5 <- ggplot(heart_data, aes(x = target, fill = exang)) +
      geom_bar(position = "dodge") + theme_bw() + labs(y="exang")
p6 <- ggplot(heart_data, aes(x = target, fill = slope)) +
      geom_bar(position = "dodge") + theme_bw() + labs(y="slope")
p7 <- ggplot(heart_data, aes(x = target, fill = ca)) +
      geom_bar(position = "dodge") + theme_bw() + labs(y="ca")
p8 <- ggplot(heart_data, aes(x = target, fill = thal)) +
      geom_bar(position = "dodge") + theme_bw() + labs(y="thal")

grid.arrange(p1, p2, p3, p4, p5, p6, p7, p8, nrow=3, ncol = 3)
```



(d) Boxplots for continuous variables vs. target

```r
# age, trestbps, chol, thalach, oldpeak
box1 <- ggplot(heart_data, aes(x = target, y = age, fill = target)) +
  geom_boxplot() +
```

```
  geom_jitter(width = .04, alpha = .2) +
  guides(fill = "none") + theme_bw()
box2 <-ggplot(heart_data, aes(x = target, y = trestbps, fill = target)) +
  geom_boxplot() +
  geom_jitter(width = .04, alpha = .2) +
  guides(fill = "none") + theme_bw()
box3 <-ggplot(heart_data, aes(x = target, y = chol, fill = target)) +
  geom_boxplot() +
  geom_jitter(width = .04, alpha = .2) +
  guides(fill = "none") + theme_bw()
box4 <- ggplot(heart_data, aes(x = target, y = thalach, fill = target)) +
  geom_boxplot() +
  geom_jitter(width = .04, alpha = .2) +
  guides(fill = "none") + theme_bw()
box5 <- ggplot(heart_data, aes(x = target, y = oldpeak, fill = target)) +
  geom_boxplot() +
  geom_jitter(width = .04, alpha = .2) +
  guides(fill = "none") + theme_bw()
grid.arrange(box1,box2,box3,box4,box5, nrow=2, ncol = 3)
```



Data preprocess

In this step, we randomly divide our dataset into train and test set following a partition ratio of 5:1. Since the dataset contains 303 observations in total, the number of observations in the train and test set would roughly be 250 and 53, respectively.

```r
# train - test split
set.seed(64)
n_obs <- dim(heart_data)[1]
train_idx <- sample(n_obs, 250)
# val_idx <- sample(seq(n_obs)[-train_idx], 50)
test_idx <- seq(n_obs)[-train_idx]
```

In the following 2 parts, we will train a model with the parameters being selected based on cross-validation performances for each algorithm. Then in the last part, we will report the training and testing accuracy for each of those models, and select the best single model based on the testing accuracy.

Algorithm - Tree-based Approaches

```r
library(ISLR)
library(leaps)
set.seed(64)
Y <- heart_data$target
X <- heart_data[, 1:13]

train_data <- heart_data[train_idx, ]
test_data <- heart_data[test_idx, ]
```

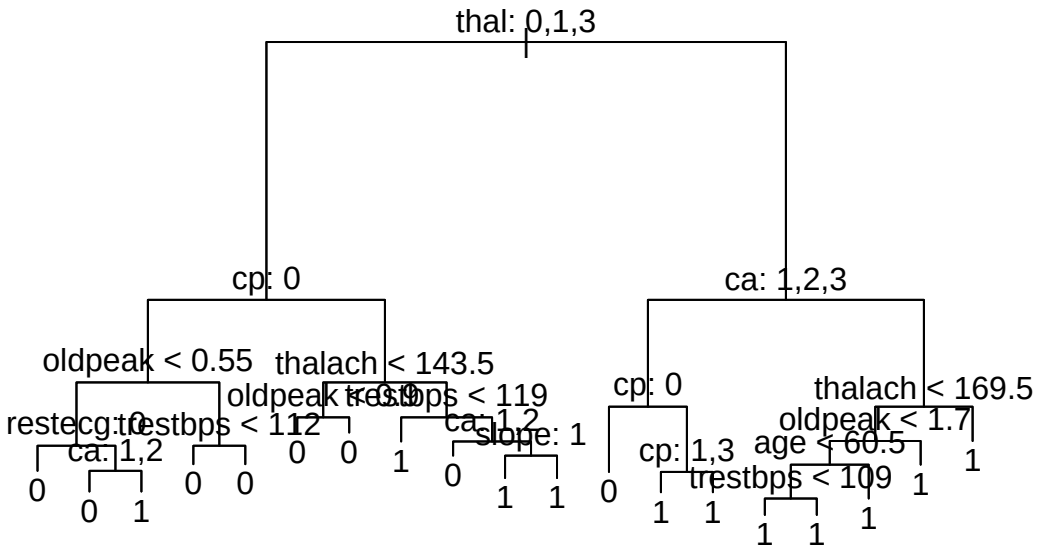
(a) Decision Tree (classification tree)

First, we fit a single tree using all variables, which grows into 19 leafs.

```r
set.seed(64)
library(tree)
# attach(heart_data)
Cl_tree <- tree(target ~ ., data = train_data)
summary(Cl_tree)
```

```
##
## Classification tree:
## tree(formula = target ~ ., data = train_data)
## Variables actually used in tree construction:
## [1] "thal"     "cp"       "oldpeak"  "restecg"  "ca"       "trestbps"
## [7] "thalach"  "slope"    "age"
## Number of terminal nodes:  19
## Residual mean deviance:  0.4683 = 108.2 / 231
## Misclassification error rate: 0.116 = 29 / 250
```

```r
## 19 terminal nodes
## training error rate :0.116
plot(Cl_tree)
text(Cl_tree, pretty=0)
```




```
##prediction on testing set
set.seed(64)
Cl_pred = predict(Cl_tree, test_data, type="class")
# table(test_data$target, Cl_pred)

## testing set accuracy :0.755
round(mean(Cl_pred == Y[test_idx]), 3)
```

```
## [1] 0.755
```

Next, we prune the tree using cross validation.

```
##Cross-validation
set.seed(64)
cv_Cl_tree <- cv.tree(Cl_tree, FUN=prune.misclass)
cv_Cl_tree$size
```

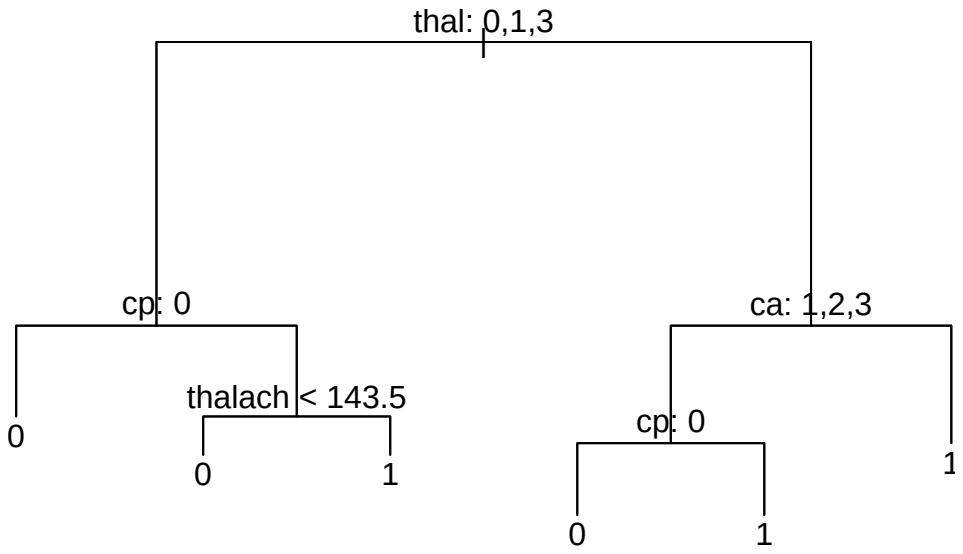
```
## [1] 19 11  9  6  4  2  1
```

```
cv_Cl_tree$dev
```

```
## [1]  55  54  54  54  64  73 113
```

Size of 11, 9, and 6 give lowest cross-validation errors. We choose the smallest size, which is 6.

```
## new classification tree: prune_Cl_tree
## Prediction on testing data
set.seed(64)
prune_Cl_tree <- prune.misclass(Cl_tree, best = 6)
plot(prune_Cl_tree)
text(prune_Cl_tree, pretty = 0)
```




```
prune_Cl_pred.test <- predict(prune_Cl_tree, test_data, type="class")
## new testing set accuracy :0.83
#table(test_data$target, prune_Cl_pred.test)
acc.test.dt <- round(mean(Y[test_idx] == prune_Cl_pred.test), 3)

prune_Cl_pred.tr <- predict(prune_Cl_tree, train_data, type="class")
## new training set accuracy :0.852
#table(train_data$target, prune_Cl_pred.tr)
acc.tr.dt <- round(mean(Y[train_idx] == prune_Cl_pred.tr), 3)
```

(b) Random Forest

In random forests, due to the availiblity of the out-of-bag samples, there is no need for cross-validation or a separate validation set to get an unbiased estimate of the validation set error.

```
set.seed(64)
require(randomForest)

## mtry = sqrt(p) is a canonical choice for classification
p <- 13
rf_heart <- randomForest(target ~ ., data = train_data, mtry = sqrt(p), importance = TRUE)
rf_heart
```

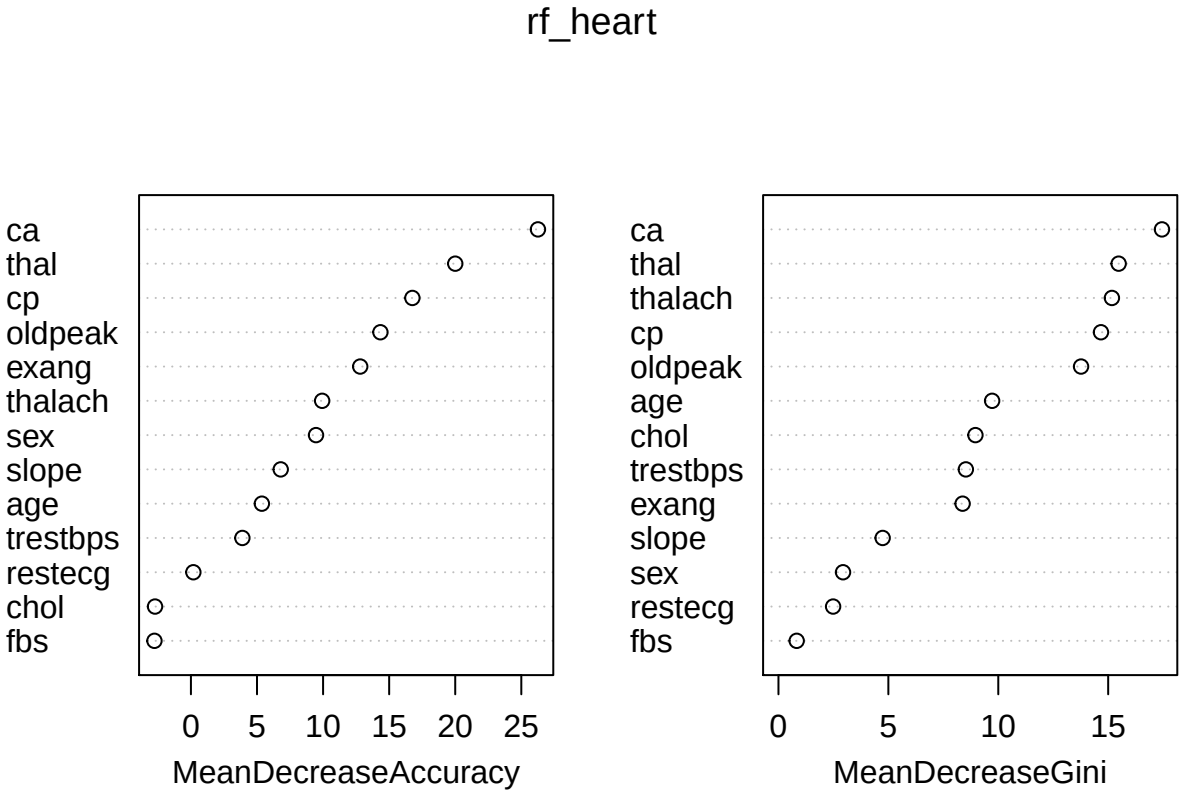
```
##
## Call:
##  randomForest(formula = target ~ ., data = train_data, mtry = sqrt(p),      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 17.6%
## Confusion matrix:
##    0   1 class.error
## 0 88  25   0.2212389
## 1 19 118   0.1386861
```

```
## OOB estimate of Error rate: 17.6%
```

```
## Prediction on testing set
set.seed(64)
yhat_rf_heart <- predict(rf_heart, newdata = test_data)
importance(rf_heart)
```

```
##                   0          1 MeanDecreaseAccuracy MeanDecreaseGini
## age       1.0662106  6.2133026            5.3675270        9.7206324
## sex       3.3634680  9.3916710            9.4648830        2.9376629
## cp       13.8766762 11.4506049           16.7561155       14.6716691
## trestbps  5.2902493  0.9132430            3.8912640        8.5250930
## chol     -1.9463663 -1.7501069           -2.7158699        8.9603722
## fbs      -2.6785630 -1.5356428           -2.7614903        0.8290427
## restecg   0.2676238  0.0149263            0.1841232        2.4863655
## thalach   6.2600242  8.1374022            9.9302652       15.1700700
## exang    10.1106892  7.9046529           12.8090421        8.3777171
## oldpeak  11.3583985  9.4203003           14.3388169       13.7658337
## slope     8.1229835  1.9939014            6.7933980        4.7400300
## ca       18.8945212 22.4418020           26.2580348       17.4464514
## thal     15.6989411 14.4085109           20.0007272       15.4760656
```

```
varImpPlot(rf_heart)
```




```
#table(test_data$target, yhat_rf_heart)
```

```r
## testing set accuracy : 0.792
round(mean(Y[test_idx] == yhat_rf_heart), 3)
```

```
## [1] 0.792
```

Neither of Mean Decrease in Accuracy and Mean Decrease in Gini (index) are bulletproof metrics and they may suffer when comparing numeric and categorical variables, or comparing numeric variables if the scales are vastly different. We cannot compare them directly. Here we would follow up all the variables except `fbs` and `restecg` since they performed poorly in both measures.

```r
set.seed(64)
rf_heart_new <- randomForest(target ~ age + sex + cp + trestbps + chol + thalach
                             + exang + oldpeak + slope + ca + thal,
                             data = train_data, mtry = sqrt(p), importance = TRUE)
rf_heart_new
```

```
##
## Call:
##  randomForest(formula = target ~ age + sex + cp + trestbps + chol +      thalach + exang + oldpeak +
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 17.6%
## Confusion matrix:
##    0   1 class.error
## 0 87  26   0.2300885
## 1 18 119   0.1313869
```

```r
## OOB estimate of Error rate: 17.6%
```

```r
set.seed(64)
## New prediction on testing set
yhat_rf_heart.test <- predict(rf_heart_new, newdata = test_data)
#table(test_data$target, yhat_rf_heart.test)
acc.test.rf = round(mean(Y[test_idx] == yhat_rf_heart.test), 3)

## New prediction on training set
yhat_rf_heart.tr <- predict(rf_heart_new, newdata = train_data)
#table(train_data$target, yhat_rf_heart.tr)
acc.tr.rf = round(mean(Y[train_idx] == yhat_rf_heart.tr), 3)
## training set accuracy : 1
## testing set accuracy : 0.811
```

New random forest model performs better on testing data.

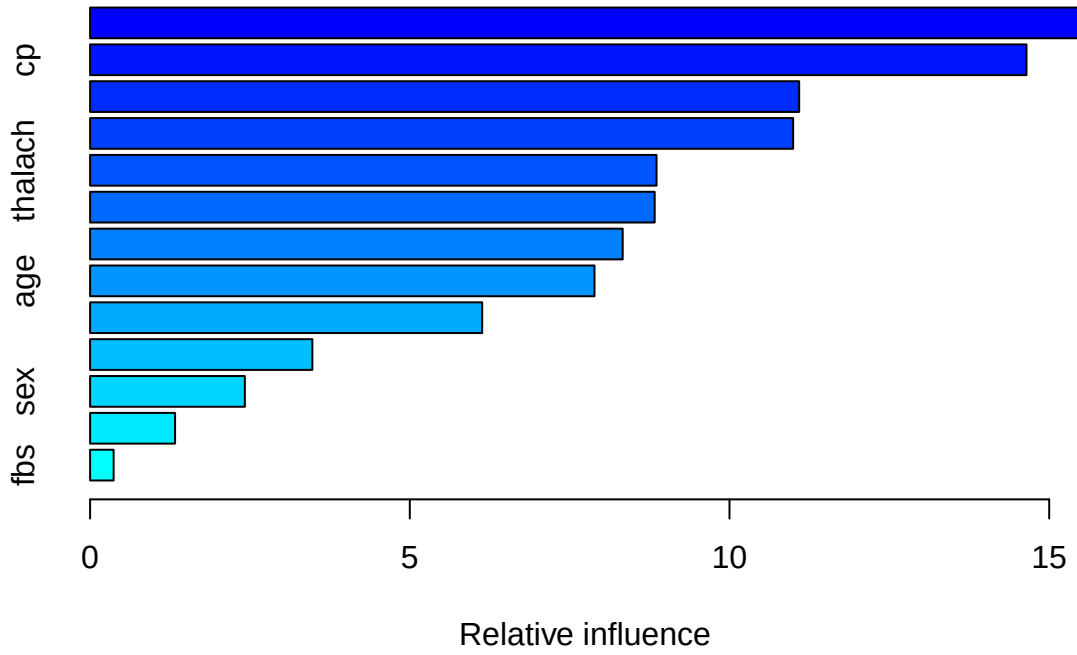
(c) Gradient Boosting

```
## Rstudio crashes when classification response was set to factor.
## I create a new gbm_heart_data to solve this problem
set.seed(64)
gbm_heart_data <- heart_data %>%
  mutate(target = ifelse(target == 1, 1, 0))

gbm_train_data <- gbm_heart_data[train_idx, ]
gbm_test_data <- gbm_heart_data[test_idx, ]
```

```
set.seed(64)
library(gbm)

gbm_heart <- gbm(target ~ ., data = gbm_train_data, distribution = "bernoulli",
                 n.trees = 5000, shrinkage = 0.01, cv.folds = 5, n.minobsinnode = 5)

print(gbm_heart)
```

```
## gbm(formula = target ~ ., distribution = "bernoulli", data = gbm_train_data,
##     n.trees = 5000, n.minobsinnode = 5, shrinkage = 0.01, cv.folds = 5)
## A gradient boosted model with bernoulli loss function.
## 5000 iterations were performed.
## The best cross-validation iteration was 760.
## There were 13 predictors of which 13 had non-zero influence.
```

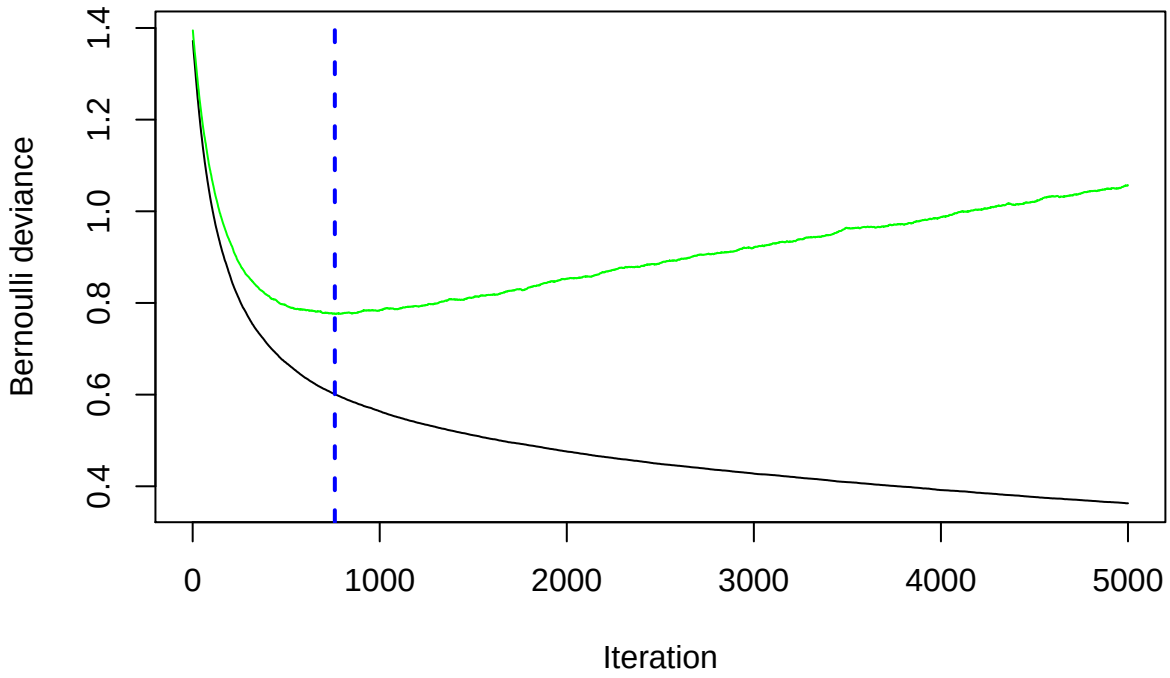
```
summary(gbm_heart)
```




```
##               var   rel.inf
## ca             ca 15.639112
## cp             cp 14.644934
## thal         thal 11.089603
```

```
## oldpeak   oldpeak 10.995375
## thalach   thalach  8.858822
## chol         chol  8.829865
## trestbps trestbps  8.329625
## age           age  7.887992
## exang       exang  6.131729
## slope       slope  3.475885
## sex           sex  2.420186
## restecg   restecg  1.328983
## fbs           fbs  0.367889
```

```
## Determine the optimal number of iterations by cross-validation: 760
gbm_heart_iter <- gbm.perf(gbm_heart, plot.it = TRUE, method = "cv")
```

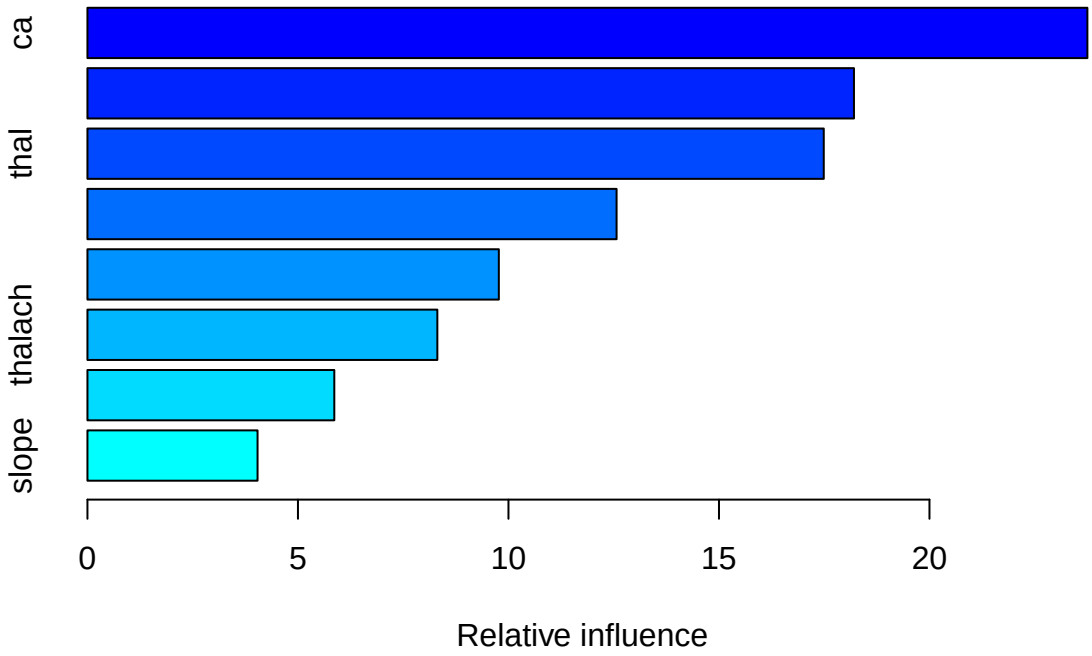


We can see that `ca` and `cp` are the most important variables for this model. The validation deviance stops to decrease at 760-th iteration. Next, let's try to exclude least important variables `fbs`, `restecg`, `age`, `sex` and `chol`.

```
set.seed(64)
gbm.1 <- gbm(target ~ .-fbs - restecg -age -sex -chol, data = gbm_train_data,
             distribution = "bernoulli", n.trees = 1000,
             shrinkage = 0.01, cv.folds = 5, n.minobsinnode = 5)

print(gbm.1)
```

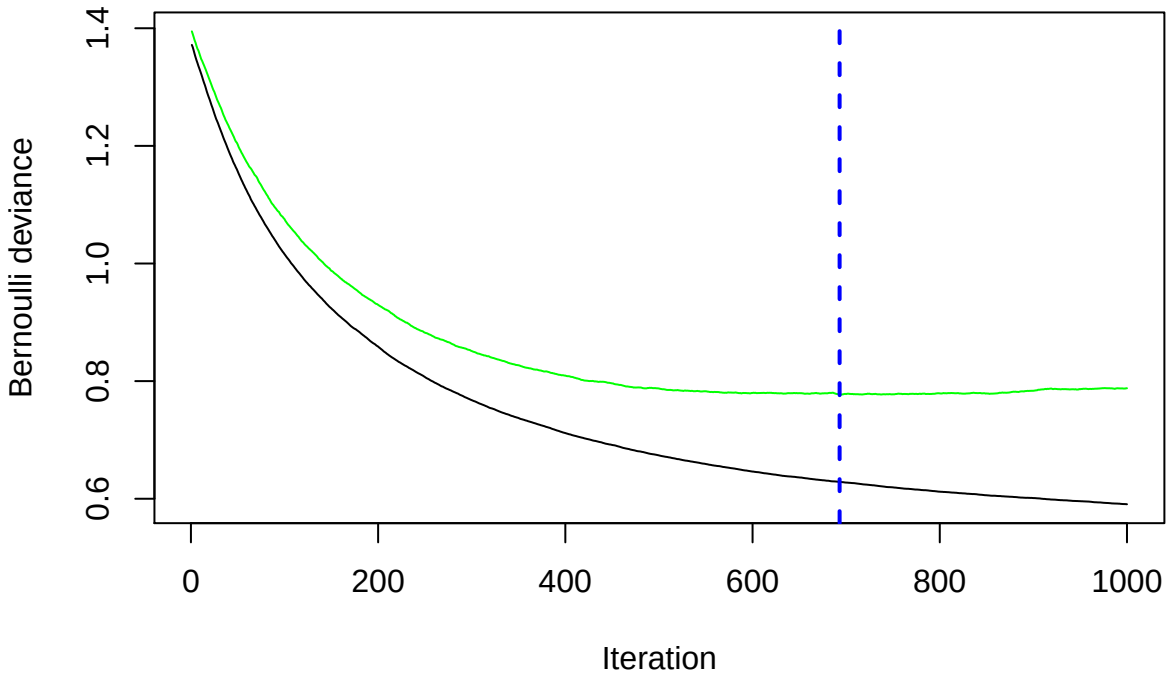
```
## gbm(formula = target ~ . - fbs - restecg - age - sex - chol,
##     distribution = "bernoulli", data = gbm_train_data, n.trees = 1000,
##     n.minobsinnode = 5, shrinkage = 0.01, cv.folds = 5)
## A gradient boosted model with bernoulli loss function.
## 1000 iterations were performed.
## The best cross-validation iteration was 693.
## There were 8 predictors of which 8 had non-zero influence.
```

```r
summary(gbm.1)
```




```
##                var   rel.inf
## ca              ca 23.750690
## cp              cp 18.206412
## thal         thal 17.488581
## oldpeak   oldpeak 12.568092
## exang       exang  9.771773
## thalach   thalach  8.310852
## trestbps trestbps  5.863310
## slope       slope  4.040290
```

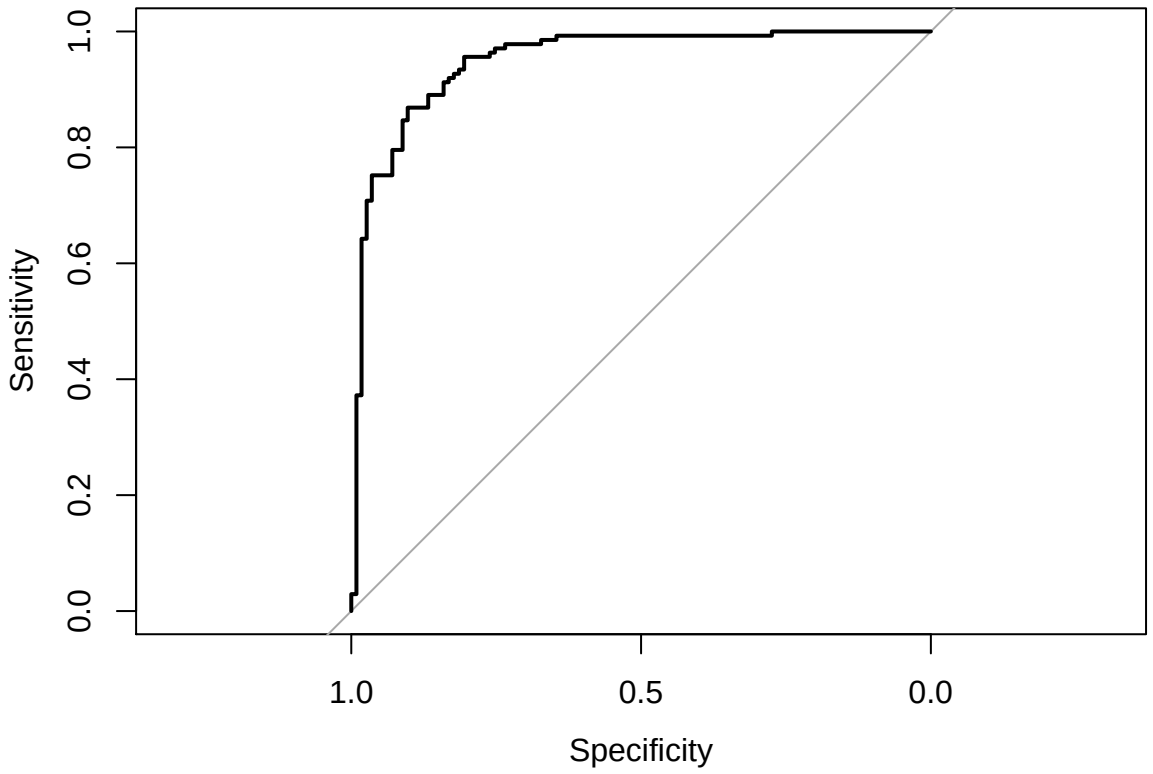
```r
## Determine the optimal number of iterations by cross-validation: 693
gbm_heart_iter <- gbm.perf(gbm.1, plot.it = TRUE, method = "cv")
```




```
## Prediction on training data
yhat_gbm_heart.tr <- predict(gbm_heart, newdata = gbm_train_data, n.trees = gbm_heart_iter)
# Prediction on testing data
yhat_gbm_heart.test <- predict(gbm_heart, newdata = gbm_test_data, n.trees = gbm_heart_iter)
```

In this step, we plot the ROC curve on the training set to get the best critical value for the maximum accuracy.

```
## Plot the ROC curve to get the best critical value for the maximum accuracy
## the selection of the best critical value should based on training set
library(stats)
library(pROC)
heart_roc <- roc(gbm_train_data$target, yhat_gbm_heart.tr)
plot(heart_roc)
```




```
# Area under the curve: 0.9

##Call the optimal threshold
##Use this threshold to get the corresponding prediction category
roc_threshold <- unlist(coords(heart_roc, "best")["threshold"])
```

```
## Calculate Accuracy
## Training set accuracy: 0.884
yhat_gbm_heart_class.tr <- ifelse(yhat_gbm_heart.tr > roc_threshold, 1, 0)
acc.tr.gbm = round(mean(gbm_train_data$target== yhat_gbm_heart_class.tr), 3)
#table(gbm_train_data$target, yhat_gbm_heart_class.tr)


## Testing set accuracy: 0.774
yhat_gbm_heart_class.test <- ifelse(yhat_gbm_heart.test > roc_threshold, 1, 0)
acc.test.gbm = round(mean(gbm_test_data$target== yhat_gbm_heart_class.test), 3)
#table(gbm_test_data$target, yhat_gbm_heart_class.test)
```

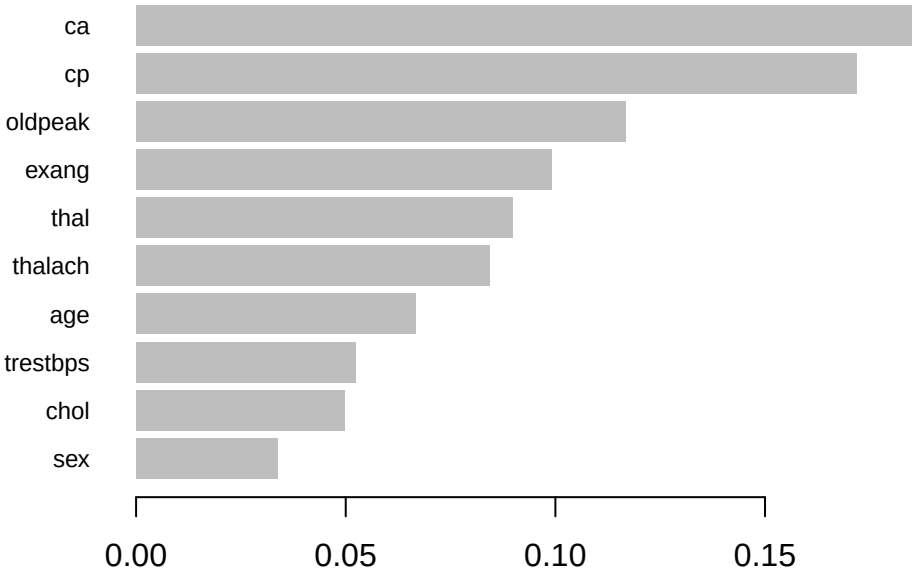
(d) Extreme Gradient Boosting

```
##XGBoost only works with matrices that contain all numeric variables, so we use heart_ dataset
set.seed(64)
require(xgboost)

xgb_train_data <- heart_[train_idx, ]
xgb_test_data <- heart_[test_idx, ]
```

```r
##isolate y variable
xgb_train_y <- xgb_train_data$target
xgb_test_y <- xgb_test_data$target

## isolate x variables
xgb_train_x <- data.matrix(xgb_train_data[,1:13])
xgb_test_x <- data.matrix(xgb_test_data[,1:13])
```

```r
## train a model using training data
set.seed(64)
## create parameter list
params <- list(eta = 0.1,
               max_depth = 6,
               subsample = 0.8,
               colsample_bytree = 0.9,
               min_child_weight = 1,
               gamma = 0,
               objective = "binary:logistic",
               booster = "gbtree",
               eval_metric = "auc")

xgb_heart <- xgb.cv(params = params,
                    data = xgb_train_x,
                    label = xgb_train_y,
                    nrounds = 2000,
                    nfold = 5,
                    verbose = 0)

max_test <- which.max(xgb_heart$evaluation_log$test_auc_mean)

## train final model
xgb_heart_final <- xgboost(params = params,
                           data = xgb_train_x,
                           label = xgb_train_y,
                           nrounds = max_test,
                           verbose = 0)
```

```r
##create importance matrix
importance_matrix <- xgb.importance(model = xgb_heart_final)

# variable importance plot
xgb.plot.importance(importance_matrix, top_n = 10, measure = "Gain")
```



We can see that `ca` and `cp` are the 2 most important variables in this model.

```
##evaluate model: train
xgb_heart_pred.tr <- predict(xgb_heart_final, newdata = xgb_train_x)
xgb_heart_pred.tr <- ifelse(xgb_heart_pred.tr > 0.5, 1,0)

#table(xgb_heart_pred.tr, xgb_train_y)
##check the accuracy on training data: 0.984

acc.tr.xgb <- round(mean(xgb_heart_pred.tr == xgb_train_y), 3)


##evaluate model: test
xgb_heart_pred.test <- predict(xgb_heart_final, newdata = xgb_test_x)
xgb_heart_pred.test <- ifelse(xgb_heart_pred.test > 0.5, 1,0)

#table(xgb_heart_pred.test, xgb_test_y)
##check the accuracy on testing data: 0.774

acc.test.xgb <- round(mean(xgb_heart_pred.test == xgb_test_y), 3)

#require(caret)
#confusionMatrix(table(xgb_heart_pred.test, xgb_test_y))
```

Algorithm - Other Approaches

For the rest of the algorithms, KNN, SVM, and NN require the input variables to be standardized.

```
library(ISLR)
library(leaps)
set.seed(64)

# standardize
# numeric_vars <- c("age", "trestbps", "chol", "thalach", "oldpeak")
```

```r
attach(heart_data)
heart_data$age <- (age - mean(age))/sd(age)
heart_data$trestbps <- (trestbps - mean(trestbps))/sd(trestbps)
heart_data$chol <- (chol - mean(chol))/sd(chol)
heart_data$thalach <- (thalach - mean(thalach))/sd(thalach)
heart_data$oldpeak <- (oldpeak - mean(oldpeak))/sd(oldpeak)
detach(heart_data)

Y <- heart_data$target
X <- heart_data[, 1:13]
train_data <- heart_data[train_idx, ]
test_data <- heart_data[test_idx, ]
```

(a) Logsitic Regression

Let's first define a function for calculating accuracy. It intakes the fitted model, and output the training accuracy and the testing accuracy.

```r
cal_acc <- function(mod_name) {
  probs <- predict(mod_name, train_data, type = "response")
  pred = rep(0, length(probs))
  pred[probs > 0.5] = 1
  probs.test <- predict(mod_name, test_data, type = "response")
  pred.test = rep(0, length(probs.test))
  pred.test[probs.test > 0.5] = 1
  acc.tr <- round(mean(pred == Y[train_idx]), 3)
  acc.test <- round(mean(pred.test == Y[test_idx]), 3)
  return(c(acc.tr, acc.test))
}
```

```r
library(boot)
set.seed(64)
lr.fit.full = glm(target ~ ., data = train_data, family = binomial)
cv.glm(train_data, lr.fit.full, K=5)$delta[1]
```

```
## [1] 0.1370464
```

We try to allievate the overfitting issue by subsetting some important predictors using backward stepwise strategy. The 'cp' indicator suggests a 12-variables model as the best choice, while the 'bic' indicator suggests a 8-variables model.

```r
lr.bwd = regsubsets(target ~ ., data = train_data, nvmax = 20, method = "backward")
bwd.summary = summary(lr.bwd)
which.min(bwd.summary$cp)
```

```
## [1] 12
```

```r
which.min(bwd.summary$bic)
```

```
## [1] 8
```

```
coefficients(lr.bwd, id = 12)
```

```
## (Intercept)        sex1         cp2         cp3    restecg1     thalach
##  1.64595091 -0.12454393  0.16776954  0.17132703  0.08222370  0.05094371
##      exang1     oldpeak      slope1         ca1         ca2         ca3
## -0.18312800 -0.07183878 -0.09804828 -0.28383095 -0.34595220 -0.30077706
##       thal2
##  0.18884409
```

```
coefficients(lr.bwd, id = 8)
```

```
## (Intercept)         cp2     thalach      exang1     oldpeak         ca1
##  1.56623020  0.15895547  0.06494195 -0.22057386 -0.07407109 -0.30491706
##         ca2         ca3       thal2
## -0.34839834 -0.34527629  0.23514680
```

We will now try each of them:

```
set.seed(64)
lr.fit.bic = glm(target ~ cp+exang+oldpeak+slope+ca+thal, data = train_data, family = binomial)
err.bic <- cv.glm(train_data, lr.fit.bic, K=5)$delta[1]
err.bic
```

```
## [1] 0.1256099
```

```
# cal_acc(lr.fit.bic)
```

```
set.seed(64)
lr.fit.cp = glm(target ~ sex+cp+chol+restecg+thalach+exang+oldpeak+slope+ca+thal, data = train_data, fa
err.cp <- cv.glm(train_data, lr.fit.cp, K=5)$delta[1]
err.cp
```

```
## [1] 0.1325376
```

```
# cal_acc(lr.fit.cp)
```

Based on the cross-validation error rate, the 6-variables model suggested by "bic" indictor over-perform other models.

```
acc.lr <- cal_acc(lr.fit.bic)
acc.tr.lr <- acc.lr[1]
acc.test.lr <- acc.lr[2]
```

(b) Linear Discriminant Analysis

```r
library(MASS)

acc_cal.lda <- function(mod_name) {

  pred <- predict(mod_name, train_data)$class
  pred.test <- predict(mod_name, test_data)$class
  acc.tr <- round(mean(pred == Y[train_idx]), 3)
  acc.test <- round(mean(pred.test == Y[test_idx]), 3)
  return(c(acc.tr, acc.test))
}
```

For the following algorithms, we would use the self-implemented 5-folds cross-validation based on this division.

```r
library(caret)
set.seed(64)
folds = createFolds(Y[train_idx], k = 5)
```

Firstly, define a function for doing cross-validation.

```r
cv_lda <- function(test.id, formula_) {
  training_fold = train_data[-test.id, ]
  test_fold = train_data[test.id, ]

  lda.fit = lda(formula_, data=training_fold)
  lda.pred <- predict(lda.fit, test_fold)$class
  accuracy = round(mean(test_fold[, 14] == lda.pred), 3)

  return(accuracy)
}
```

Next, we evaluate cross-validation error rate based on the following 3 combinations of predictors. These 3 combinations are the two combinations suggested by stepwise selection, and all variables respectively.

```r
allVars <- colnames(train_data)
predictorVars <- allVars[!allVars%in%'target']
predictorVars <- paste(predictorVars, collapse = "+")
f.full <- as.formula(paste("target~", predictorVars, collapse = "+"))
acc.full <- lapply(folds, cv_lda, f.full)
acc.full <- mean(as.numeric(acc.full))
acc.full # 0.844
```

```
## [1] 0.844
```

```r
predictorVars_cp <- c("sex", "cp", "chol", "restecg", "thalach", "exang", "oldpeak", "slope", "ca", "tha
predictorVars_cp <- paste(predictorVars_cp, collapse = "+")
f.cp <- as.formula(paste("target~", predictorVars_cp, collapse = "+"))
acc.cp <- lapply(folds, cv_lda, f.cp)
acc.cp <- mean(as.numeric(acc.cp))
acc.cp  # 0.852
```

```
## [1] 0.852
```

```
predictorVars_bic <- c("cp", "exang", "oldpeak", "slope", "ca", "thal")
predictorVars_bic <- paste(predictorVars_bic, collapse = "+")
f.bic <- as.formula(paste("target~", predictorVars_bic, collapse = "+"))
acc.bic <- lapply(folds, cv_lda, f.bic)
acc.bic <- mean(as.numeric(acc.bic))
acc.bic  # 0.84
```

```
## [1] 0.84
```

Based on the cross-validation accuracy, the second lda model containing 10 predictors has the best performance. Thus, we select this model as the final lda model. Next we evaluate the training and testing accuracy of this model.

```
lda.fit <- lda(f.cp, train_data)
acc.lda <- acc_cal.lda(lda.fit)
acc.tr.lda <- acc.lda[1]
acc.test.lda <- acc.lda[2]
```

(c) K-Nearest Neighbors Classifier

In order to select the parameter k for KNN, we first define a function for cross-validation.

```
cv_knn <- function(test.id, k = 3) {
  training_fold = train_data[-test.id, ]
  test_fold = train_data[test.id, ]
  knn.pred = knn(training_fold[, 1:13], test_fold[1:13], training_fold[, 14], k=k)
  accuracy = round(mean(test_fold[, 14] == knn.pred), 3)
  return(accuracy)
}
```

Next step is selecting k

```
library(class)
set.seed(64)

k_vec <- seq(1, 19, 2)
knn.acc.cv <- rep(0, length(k_vec))
for (i in 1:length(k_vec)) {
  ki <- k_vec[i]
  acc.knn <- lapply(folds, cv_knn, ki)
  knn.acc.cv[i] <- mean(as.numeric(acc.knn))
}
tibble(k = k_vec, acc.CV = knn.acc.cv)
```

```
## # A tibble: 10 x 2
##        k acc.CV
##    <dbl>  <dbl>
##  1     1  0.764
##  2     3  0.804
##  3     5  0.82
##  4     7  0.804
```

```
##  5     9  0.808
##  6    11  0.816
##  7    13  0.808
##  8    15  0.812
##  9    17  0.788
## 10    19  0.804
```

When k=5, the knn yields the highest cross-validation accuracy (0.820)

```
knn.pred.tr = knn(X[train_idx, ], X[train_idx, ], Y[train_idx], k=5)
acc.tr.knn <- round(mean(knn.pred.tr == Y[train_idx]), 3)  # train.accuracy
knn.pred.test = knn(X[train_idx, ], X[test_idx, ], Y[train_idx], k=5)
acc.test.knn <- round(mean(knn.pred.test == Y[test_idx]), 3) # test.accuracy
```

(d) Support Vector Machine

```
set.seed(64)
library(e1071)

cal_acc.svm <- function(svm.fit) {
  svm.pred = predict(svm.fit, train_data)
  acc.tr <- round(mean(svm.pred == Y[train_idx]), 3)
  svm.pred.test = predict(svm.fit, test_data)
  acc.test <- round(mean(svm.pred.test== Y[test_idx]), 3)
  return(c(acc.tr, acc.test))
}
```

In order to select parameter combinations for SVM, we first define a function for cross-validation.

```
cv_svm <- function(test.id, gamma=1, cost=1) {
  training_fold = train_data[-test.id, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = train_data[test.id, ] # here we describe the test fold individually
  classifier = svm(target ~ ., data = training_fold, gamma = gamma, cost = cost, kernel="radial")
  y_pred = predict(classifier, newdata = test_fold)
  accuracy = round(mean(test_fold[, 14] == y_pred), 3)
  return(accuracy)
}
```

Then, we search for an appropriate range of gamma value throught 5-folds cross-validation.

```
search_area.gamma <- c(0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 10)
vec.acc.cv <- rep(0, length(search_area.gamma))

for (i in 1:length(search_area.gamma)) {
  gamma <- search_area.gamma[i]
  acc <- lapply(folds, cv_svm, gamma)
  vec.acc.cv[i] <- mean(as.numeric(acc))
}
gamma_acc <- tibble(gamma = search_area.gamma, acc.CV = vec.acc.cv)
gamma_acc
```

```
## # A tibble: 9 x 2
##   gamma acc.CV
##   <dbl>  <dbl>
## 1  0.01  0.82
## 2  0.02  0.828
## 3  0.05  0.82
## 4  0.1   0.836
## 5  0.2   0.832
## 6  0.5   0.808
## 7  1     0.624
## 8  2     0.556
## 9 10     0.548
```

It seems like $0.01 \sim 0.5$ is the appropriate range for the value of gamma. The next step is to search for a good value of "cost", trying to alleviate overfitting.

```
set.seed(64)
search_area.cost <- c( 0.001, 0.005, 0.01, 0.1, 1, 10, 100, 1000)

results_table.svm <- tibble(cost = search_area.cost)
for (gamma in c(0.01, 0.02, 0.05, 0.1, 0.2, 0.5)) {
  vec.acc.cv <- rep(0, length(search_area.cost))
  for (i in 1:length(search_area.cost)) {
    cost <- search_area.cost[i]
    acc <- lapply(folds, cv_svm, gamma, cost)
    vec.acc.cv[i] <- mean(as.numeric(acc))
  }
  results_table.svm <- cbind(results_table.svm, vec.acc.cv)
}
colnames(results_table.svm) <- c('cost', 'gamma0.01','gamma0.02','gamma0.05', 'gamma0.1', 'gamma0.2', 'g
results_table.svm
```

```
##    cost gamma0.01 gamma0.02 gamma0.05 gamma0.1 gamma0.2 gamma0.5
## 1 1e-03     0.548     0.548     0.548    0.548    0.548    0.548
## 2 5e-03     0.548     0.548     0.548    0.548    0.548    0.548
## 3 1e-02     0.548     0.548     0.548    0.548    0.548    0.548
## 4 1e-01     0.564     0.748     0.792    0.788    0.744    0.548
## 5 1e+00     0.820     0.828     0.820    0.836    0.832    0.808
## 6 1e+01     0.824     0.840     0.812    0.788    0.784    0.816
## 7 1e+02     0.844     0.796     0.752    0.756    0.784    0.816
## 8 1e+03     0.764     0.744     0.764    0.756    0.784    0.816
```

When gamma = 0.01; cost = 100, svm yields the best cross-validation accurarcy: 0.844.

```
svm.fit.best = svm(target ~ ., data = train_data, gamma = 0.01, cost = 100)
acc.svm = cal_acc.svm(svm.fit.best)
acc.tr.svm <- acc.svm[1]
acc.test.svm <- acc.svm[2]
```

(e) Neural Networks

```r
library(grid)
library(neuralnet)

# preprocess for NN:
data.nn <- data.frame(scale(heart_[, 1:13]))
data.nn$target <- heart_[, 14]

train_data.nn <- data.nn[train_idx, ]
test_data.nn <- data.nn[test_idx, ]
```

Similarily, the first step is to define a function for doing cross-validation.

```r
cv_nn <- function(test.id, hidden_layer=c(5, 3), lr=0.01) {
  training_fold = train_data.nn[-test.id, ]
  test_fold = train_data.nn[test.id, ]

  allVars <- colnames(data.nn)
  predictorVars <- allVars[!allVars%in%'target']
  predictorVars <- paste(predictorVars, collapse = "+")
  f <- as.formula(paste("target~", predictorVars, collapse = "+"))
  net.fit <- neuralnet(f, training_fold, hidden= hidden_layer, learningrate = lr,
                       act.fct = "logistic", linear.output=F)
  probs.nn <- compute(net.fit, test_fold[, 1:13])
  pred.nn <- rep(0, length(test.id))
  pred.nn[probs.nn$net.result > 0.5] = 1
  accuracy <- round(mean(pred.nn == test_fold[, 14]), 3)
  return(accuracy)
}
```

Since the dataset has a small sample size, training a neural network won't take long. Thus, I decide to search for a good combination of hidden layers and the learning rate through cross-validation.

```r
set.seed(64)

hidden_1 <- 3:8
hidden_2 <- 2:3
learning_rates <- c(0.001, 0.005, 0.01, 0.05, 0.1)

# acc.cv <- rep(0, length(hidden_2))
results_table.nn <- tibble(h1 = hidden_1)
for (lr in learning_rates) {
  results_table_lr <- tibble(h1 = hidden_1)
  for (h2 in hidden_2) {
    acc.cv <- c()
    for (h1 in hidden_1) {
      hidden_layers <- c(h1, h2)
      acc <- lapply(folds, cv_nn, hidden_layers, lr)
      acc.cv <- c(acc.cv, mean(as.numeric(acc)))
    }
    results_table_lr <- cbind(results_table_lr, acc.cv)
  }
  colnames(results_table_lr) <- paste(c("h1", "h2-2", "h2-3"), lr, sep="-")
```

```
  results_table.nn <- cbind(results_table.nn, results_table_lr[, 2:3])
}
results_table.nn
```

```
##   h1 h2-2-0.001 h2-3-0.001 h2-2-0.005 h2-3-0.005 h2-2-0.01 h2-3-0.01
## 1  3      0.808      0.836      0.776      0.792     0.812     0.804
## 2  4      0.808      0.784      0.828      0.776     0.812     0.816
## 3  5      0.800      0.804      0.776      0.832     0.808     0.784
## 4  6      0.788      0.784      0.812      0.784     0.812     0.784
## 5  7      0.772      0.820      0.784      0.792     0.800     0.788
## 6  8      0.816      0.792      0.772      0.804     0.772     0.792
##   h2-2-0.05 h2-3-0.05 h2-2-0.1 h2-3-0.1
## 1     0.816     0.796    0.788    0.788
## 2     0.788     0.848    0.764    0.816
## 3     0.772     0.764    0.732    0.812
## 4     0.824     0.820    0.796    0.796
## 5     0.776     0.784    0.796    0.788
## 6     0.804     0.816    0.784    0.800
```

When we have 2 hidden layers with the number of neurons in each layer being 4 and 3, and learning rate = 0.05, the neural netwrok yields best cross-validation accuracy (0.848).

```
set.seed(64)

allVars <- colnames(data.nn)
predictorVars <- allVars[!allVars%in%'target']
predictorVars <- paste(predictorVars, collapse = "+")
f <- as.formula(paste("target~", predictorVars, collapse = "+"))
net.best <- neuralnet(f, train_data.nn, hidden= c(4, 3), learningrate = 0.05,
                      act.fct = "logistic", linear.output=F)
plot(net.best)
# train acc
probs.nn <- compute(net.best, train_data.nn[, 1:13])
pred.nn.tr <- rep(0, length(train_idx))
pred.nn.tr[probs.nn$net.result > 0.5] = 1
acc.tr.nn <- round(mean(pred.nn.tr == Y[train_idx]), 3)

# test acc
probs.nn.test <- compute(net.best, test_data.nn[, 1:13])
pred.nn.test <- rep(0, length(test_idx))
pred.nn.test[probs.nn.test$net.result > 0.5] = 1
acc.test.nn <- round(mean(pred.nn.test == Y[test_idx]), 3)

acc.tr.nn
```

```
## [1] 0.964
```

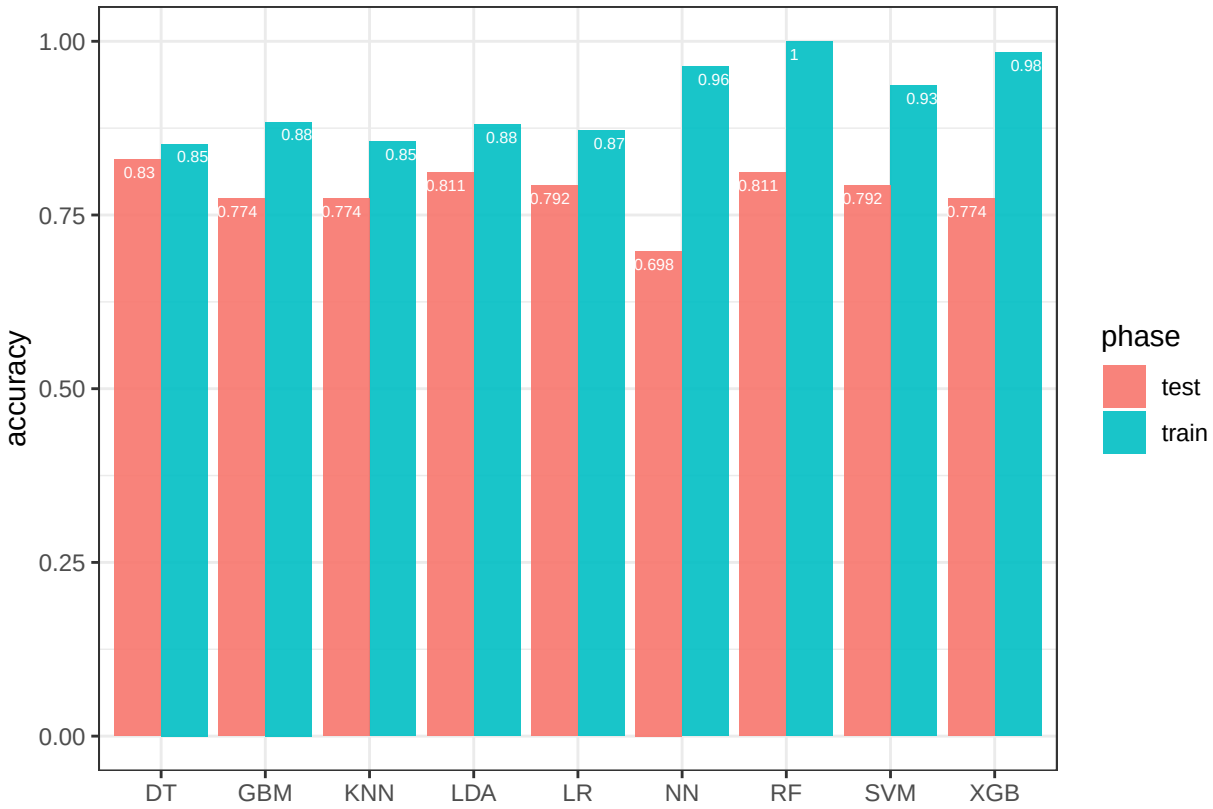
```
acc.test.nn
```

```
## [1] 0.698
```

Model Selection

Selection of best Single Model

```
train_accuracy =c(acc.tr.dt, acc.tr.rf, acc.tr.gbm, acc.tr.xgb,
                  acc.tr.lr, acc.tr.lda, acc.tr.knn, acc.tr.svm, acc.tr.nn)
test_accuracy = c(acc.test.dt, acc.test.rf, acc.test.gbm, acc.test.xgb,
                  acc.test.lr, acc.test.lda, acc.test.knn, acc.test.svm, acc.test.nn)


ACC <- tibble(accuracy=c(train_accuracy, test_accuracy),
              algo=rep(c('DT', 'RF','GBM','XGB',
                         'LR', 'LDA', 'KNN', 'SVM', 'NN'), 2),
              phase=factor(rep(c('train', 'test'), each=9))
              )


ggplot(ACC, aes(algo, accuracy)) +
  geom_bar(aes(fill = phase), position = "dodge", stat="identity", alpha=0.9) +
  geom_text(aes(label=accuracy), vjust=1.6, hjust = rep(c(-0.4, 1.2), each=9), color="white", size=2) +
  theme_bw() +
  labs(x="")
```



From the histogram, we observe that the DT model has the highest test accuracy(0.83), while other more complex models suffer from severe overfitting issue although they have higher training accuracy.

Model Ensemble

First Step get predictions of each algorithm for both training and testing set.

```
# lr.fit.bic
probs <- predict(lr.fit.bic, train_data, type = "response")
pred.tr.lr = rep(0, length(probs))
pred.tr.lr[probs > 0.5] = 1
probs.test <- predict(lr.fit.bic, test_data, type = "response")
pred.test.lr = rep(0, length(probs.test))
pred.test.lr[probs.test > 0.5] = 1

# lda.fit.best
lda.pred.tr <- predict(lda.fit, train_data)$class
lda.pred.test <- predict(lda.fit, test_data)$class

# svm.fit.best
svm.pred.tr = predict(svm.fit.best, train_data)
svm.pred.test = predict(svm.fit.best, test_data)
```

```
# convert all factors into numeric
prune_Cl_pred.test = ifelse(prune_Cl_pred.test == 1, 1, 0)
prune_Cl_pred.tr = ifelse(prune_Cl_pred.tr == 1, 1, 0)
yhat_rf_heart.test = ifelse(yhat_rf_heart.test == 1, 1, 0)
yhat_rf_heart.tr = ifelse(yhat_rf_heart.tr == 1, 1, 0)
yhat_gbm_heart_class.test = ifelse(yhat_gbm_heart_class.test == 1, 1, 0)
yhat_gbm_heart_class.tr = ifelse(yhat_gbm_heart_class.tr == 1, 1, 0)
xgb_heart_pred.test = ifelse(xgb_heart_pred.test == 1, 1, 0)
xgb_heart_pred.tr = ifelse(xgb_heart_pred.tr == 1, 1, 0)
pred.tr.lr = ifelse(pred.tr.lr == 1, 1, 0)
pred.test.lr = ifelse(pred.test.lr == 1, 1, 0)
lda.pred.tr = ifelse(lda.pred.tr == 1, 1, 0)
lda.pred.test = ifelse(lda.pred.test == 1, 1, 0)
knn.pred.tr = ifelse(knn.pred.tr == 1, 1, 0)
knn.pred.test = ifelse(knn.pred.test == 1, 1, 0)
svm.pred.tr = ifelse(svm.pred.tr == 1, 1, 0)
svm.pred.test = ifelse(svm.pred.test == 1, 1, 0)
pred.nn.tr = ifelse(pred.nn.tr == 1, 1, 0)
pred.nn.test = ifelse(pred.nn.test == 1, 1, 0)

train_prediction <- cbind(dt = prune_Cl_pred.tr, rf = yhat_rf_heart.tr,
                          gbm = yhat_gbm_heart_class.tr, xgb=xgb_heart_pred.tr,
                          lr = pred.tr.lr, lda = lda.pred.tr,
                          knn = knn.pred.tr, svm = svm.pred.tr, nn = pred.nn.tr)


# define a function for finding the majority class
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]
}
bagged_pred.tr <- rep(0, length(train_idx))
for (i.row in 1:dim(train_prediction)[1]) {
  i.pred <- getmode(train_prediction[i.row, ])
```

```
  bagged_pred.tr[i.row] = i.pred
}
round(mean(bagged_pred.tr == Y[train_idx]), 3)
```

```
## [1] 0.928
```

```
test_prediction <- cbind(dt = prune_Cl_pred.test, rf = yhat_rf_heart.test,
                         gbm = yhat_gbm_heart_class.test, xgb=xgb_heart_pred.test,
                         lr = pred.test.lr, lda = lda.pred.test,
                         knn = knn.pred.test, svm = svm.pred.test, nn = pred.nn.test)
                         #nn1 = pred.nn.test, nn2 = pred.nn.test, dt1 = prune_Cl_pred.test)
bagged_pred.test <- rep(0, length(test_idx))
for (i.row in 1:dim(test_prediction)[1]) {
  i.pred <- getmode(test_prediction[i.row, ])
  bagged_pred.test[i.row] = i.pred
}
round(mean(bagged_pred.test == Y[test_idx]), 3)
```

```
## [1] 0.792
```

```
test_prediction <- cbind(dt = prune_Cl_pred.test,
                         rf = yhat_rf_heart.test,
                         lda = lda.pred.test)
bagged_pred.test <- rep(0, length(test_idx))
for (i.row in 1:dim(test_prediction)[1]) {
  i.pred <- getmode(test_prediction[i.row, ])
  bagged_pred.test[i.row] = i.pred
}
round(mean(bagged_pred.test == Y[test_idx]), 3)
```

```
## [1] 0.811
```

Majority voting does not produce results that exceed the best single model. One of the possible reasons is that the sample size is relatively small, so the effect of ensemble on accuracy improvement is not well demonstrated. Secondly, the single models envolved in ensemble are not equally good, and some of them face serious overfitting problems.