mp4_fliu311_R_code

Faqiang Liu

4/3/2021

This is the analysis part of Mini Project 4 - Predictive Analytics. This includes data importing, data cleaning, modeling, and performance evaluation.

1. Data Import

This project uses the Cleveland database taken from the UCI repository link.The complete list of 14 attributes (excluding 1 outcome variable) used and definition are as follows:
> age: age in years > sex: male or female > cp: chest pain type > trestbps: resting blood pressure > chol: cholesterol level > fbs: fasting blood sugar level > restecg: resting electrocardiographic results > thalach: maximum heart rate achieved > exang: exercise induced angina > oldpeak: ST depression induced by exercise relative to rest > slope: the slope of the peak exercise segment > ca: Number of major vessels colored by fluoroscopy > thal: thallium scan

```r
setwd("C:/Users/dlphia/Desktop/CS6440IHI/mp4/heart_disease_prediction")

options(warn = -1)

library(ggplot2)
library(dplyr)
library(readr)
library(corrplot)
library(caret)
library(pbkrtest)
library(ROCR)
library(tree)
library(randomForest)
library(rstanarm)
library(pROC)

heart <- read.csv("heart.csv", header = FALSE, sep = ",")

colnames(heart) <- c("Age", "Gender", "CP", "TBps", "Chol", "Fbs", "Recg", "Thalach",
                     "Exang", "Op", "Slope", "Ca", "Thal", "Heart")
```

2. Data Cleaning and pre-Processing

There are 14 variables from the dataset we are using in this project. All fields are converted to numeric, and null values are imputed with mean imputation.

```r
str(heart)
```

```
## 'data.frame':    920 obs. of  14 variables:
##  $ Age    : int  63 67 67 37 41 56 62 57 63 53 ...
##  $ Gender : int  1 1 1 1 0 1 0 0 1 1 ...
##  $ CP     : int  1 4 4 3 2 2 4 4 4 4 ...
##  $ TBps   : chr  "145" "160" "120" "130" ...
##  $ Chol   : chr  "233" "286" "229" "250" ...
##  $ Fbs    : chr  "1" "0" "0" "0" ...
##  $ Recg   : chr  "2" "2" "2" "0" ...
##  $ Thalach: chr  "150" "108" "129" "187" ...
##  $ Exang  : chr  "0" "1" "1" "0" ...
##  $ Op     : chr  "2.3" "1.5" "2.6" "3.5" ...
##  $ Slope  : chr  "3" "2" "2" "3" ...
##  $ Ca     : chr  "0" "3" "2" "0" ...
##  $ Thal   : chr  "6" "3" "7" "3" ...
##  $ Heart  : int  0 2 1 0 0 0 3 0 2 1 ...
```

```r
#convert variables to numeric values
heart$CP = as.numeric(as.character(heart$CP))
heart$TBps = as.numeric(as.character(heart$TBps))
heart$Chol = as.numeric(as.character(heart$Chol))
heart$Fbs = as.numeric(as.character(heart$Fbs))
heart$Recg = as.numeric(as.character(heart$Recg))
heart$Thalach = as.numeric(as.character(heart$Thalach))
heart$Exang = as.numeric(as.character(heart$Exang))
heart$Op = as.numeric(as.character(heart$Op))
heart$Slope = as.numeric(as.character(heart$Slope))
heart$Ca = as.numeric(as.character(heart$Ca))
heart$Thal = as.numeric(as.character(heart$Thal))
heart$Heart = as.numeric(as.character(heart$Heart))
str(heart)
```

```
## 'data.frame':    920 obs. of  14 variables:
##  $ Age    : int  63 67 67 37 41 56 62 57 63 53 ...
##  $ Gender : int  1 1 1 1 0 1 0 0 1 1 ...
##  $ CP     : num  1 4 4 3 2 2 4 4 4 4 ...
##  $ TBps   : num  145 160 120 130 130 120 140 120 130 140 ...
##  $ Chol   : num  233 286 229 250 204 236 268 354 254 203 ...
##  $ Fbs    : num  1 0 0 0 0 0 0 0 0 1 ...
##  $ Recg   : num  2 2 2 0 2 0 2 0 2 2 ...
##  $ Thalach: num  150 108 129 187 172 178 160 163 147 155 ...
##  $ Exang  : num  0 1 1 0 0 0 0 1 0 1 ...
##  $ Op     : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
##  $ Slope  : num  3 2 2 3 1 1 3 1 2 3 ...
##  $ Ca     : num  0 3 2 0 0 0 2 0 1 0 ...
##  $ Thal   : num  6 3 7 3 3 3 3 3 7 7 ...
##  $ Heart  : num  0 2 1 0 0 0 3 0 2 1 ...
```

```r
#impute for na values, using mean imputation
heart$Age[which(is.na(heart$Age))] = mean(heart$Age, na.rm = TRUE)
heart$Gender[which(is.na(heart$Gender))] = mean(heart$Gender, na.rm = TRUE)
heart$CP[which(is.na(heart$CP))] = mean(heart$CP, na.rm = TRUE)
heart$TBps[which(is.na(heart$TBps))] = mean(heart$TBps, na.rm = TRUE)
heart$Chol[which(is.na(heart$Chol))] = mean(heart$Chol, na.rm = TRUE)
heart$Fbs[which(is.na(heart$Fbs))] = mean(heart$Fbs, na.rm = TRUE)
heart$Recg[which(is.na(heart$Recg))] = mean(heart$Recg, na.rm = TRUE)
heart$Thalach[which(is.na(heart$Thalach))] = mean(heart$Thalach, na.rm = TRUE)
heart$Exang[which(is.na(heart$Exang))] = mean(heart$Exang, na.rm = TRUE)
heart$Op[which(is.na(heart$Op))] = mean(heart$Op, na.rm = TRUE)
heart$Slope[which(is.na(heart$Slope))] = mean(heart$Slope, na.rm = TRUE)
heart$Ca[which(is.na(heart$Ca))] = mean(heart$Ca, na.rm = TRUE)
heart$Thal[which(is.na(heart$Thal))] = mean(heart$Thal, na.rm = TRUE)
heart$Heart[which(is.na(heart$Heart))] = mean(heart$Heart, na.rm = TRUE)
```

The “goal” field indicates if patient has heart disease. And categorical variable is converted to binary, with 0/1 representing pass/fail respectively. Feature scaling is conducted to scale the data to the interval between zero and one. This is make sure features with lower values are evaluated equally in the model as those with higher value range.

```r
#convert categorical outcome variable to binary, pass/fail
heart$Heart[heart$Heart == "2"] <- "1"
heart$Heart[heart$Heart == "3"] <- "1"
heart$Heart[heart$Heart == "4"] <- "1"
heart$Heart = as.numeric(as.character(heart$Heart))

##scalling,
library(caret)
#The preProcess option "range", scales the data to the interval between zero and one.
preprocessParamsh <- preProcess(heart, method = c("range"))
print(preprocessParamsh)
```

```
## Created from 920 samples and 14 variables
##
## Pre-processing:
##   - ignored (0)
##   - re-scaling to [0, 1] (14)
```

```r
transformedh <- predict(preprocessParamsh, heart)

heart = transformedh
```

Data screen show:

```r
head(heart)
```

```
##         Age Gender        CP  TBps      Chol Fbs Recg   Thalach Exang        Op
## 1 0.7142857      1 0.0000000 0.725 0.3864013   1    1 0.6338028     0 0.5568182
## 2 0.7959184      1 1.0000000 0.800 0.4742952   0    1 0.3380282     1 0.4659091
## 3 0.7959184      1 1.0000000 0.600 0.3797678   0    1 0.4859155     1 0.5909091
```

```
## 4 0.1836735      1 0.6666667 0.650 0.4145937   0     0 0.8943662      0 0.6931818
## 5 0.2653061      0 0.3333333 0.650 0.3383085   0     1 0.7887324      0 0.4545455
## 6 0.5714286      1 0.3333333 0.600 0.3913765   0     0 0.8309859      0 0.3863636
##   Slope        Ca Thal Heart
## 1   1.0 0.0000000 0.75     0
## 2   0.5 1.0000000 0.00     1
## 3   0.5 0.6666667 1.00     1
## 4   1.0 0.0000000 0.00     0
## 5   0.0 0.0000000 0.00     0
## 6   0.0 0.0000000 0.00     0
```

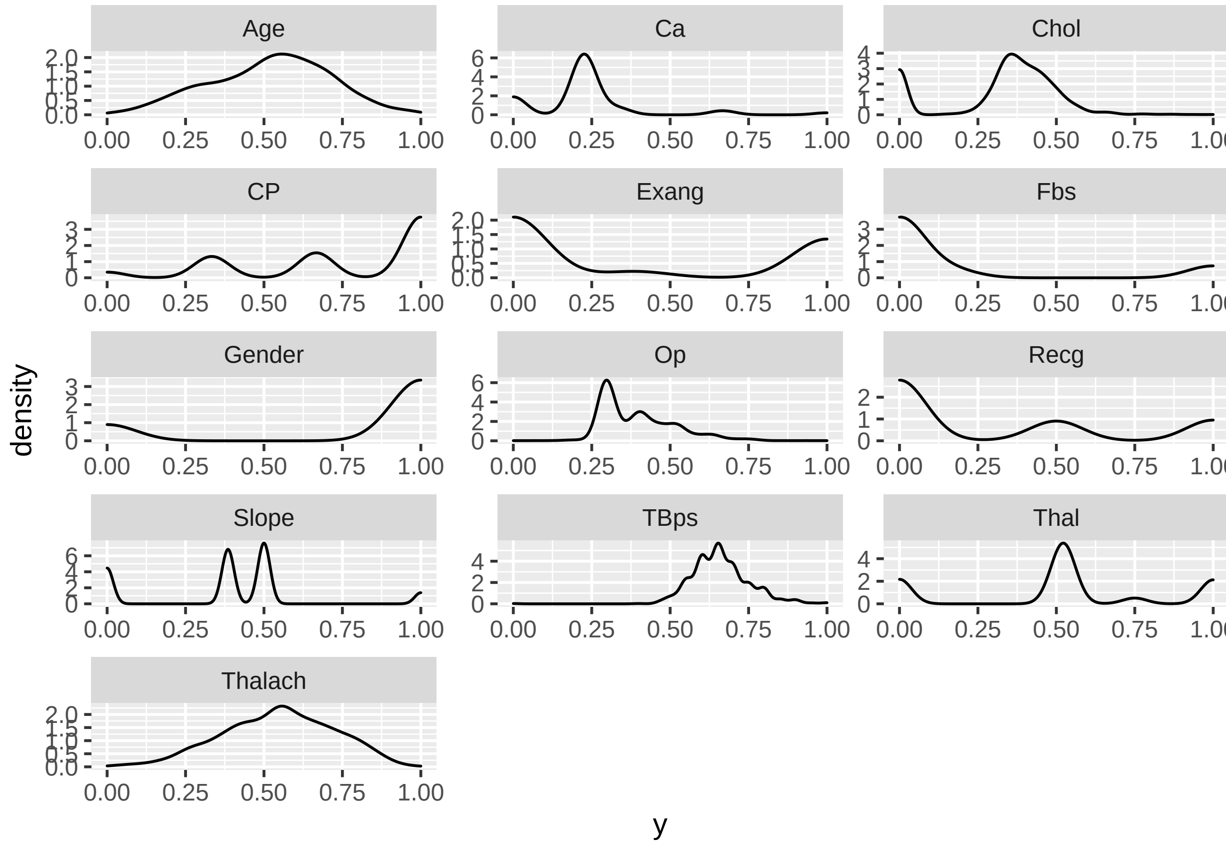
```r
summary(heart)
```

```
##       Age             Gender             CP              TBps
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.3878   1st Qu.:1.0000   1st Qu.:0.6667   1st Qu.:0.6000
##  Median :0.5306   Median :1.0000   Median :1.0000   Median :0.6500
##  Mean   :0.5206   Mean   :0.7891   Mean   :0.7500   Mean   :0.6607
##  3rd Qu.:0.6531   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.7000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       Chol             Fbs              Recg           Thalach
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2948   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.4225
##  Median :0.3665   Median :0.0000   Median :0.0000   Median :0.5493
##  Mean   :0.3302   Mean   :0.1663   Mean   :0.3023   Mean   :0.5461
##  3rd Qu.:0.4428   3rd Qu.:0.0000   3rd Qu.:0.5000   3rd Qu.:0.6761
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##      Exang              Op             Slope              Ca
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.2955   1st Qu.:0.3854   1st Qu.:0.2255
##  Median :0.0000   Median :0.3864   Median :0.3854   Median :0.2255
##  Mean   :0.3896   Mean   :0.3953   Mean   :0.3854   Mean   :0.2255
##  3rd Qu.:1.0000   3rd Qu.:0.4659   3rd Qu.:0.5000   3rd Qu.:0.2255
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##       Thal            Heart
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.5219   1st Qu.:0.0000
##  Median :0.5219   Median :1.0000
##  Mean   :0.5219   Mean   :0.5533
##  3rd Qu.:0.7500   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000
```

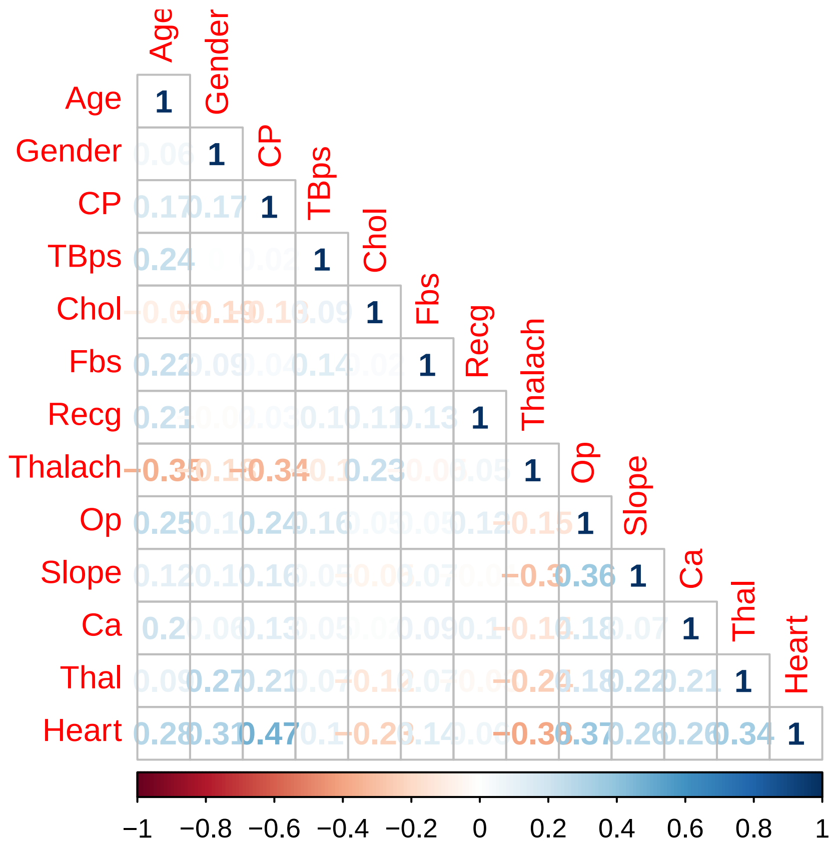
Relationships between the risk factors and risk of heart disease are plotted; this also helps visually evaluate importance of risk factors in predicting risk (a variable is more important if it varies as with outcome variable).For example, we can tell from the plot of CP, that it is possitively correlated with risk of heart disease, making it a predictor potentially of high importance.

```r
 #Relationship between Cp, heart
library(tidyr)

gather(heart, x, y, Age:Thal) %>%
  ggplot(aes(x = y, color = Heart, fill = Heart)) +
```

```r
  geom_density(alpha = 0.3) +
  facet_wrap( ~ x, scales = "free", ncol = 3)
```




```r
#check correlation between independent variables
corrplot(cor(heart[, -9]), type = "lower", method = "number")
```



As we can tell from the level of correlation, the 13 predictors are relatively independent of each other. (In cases of having predictors of high correlation, which we call 'multicollinearity' in regression analysis, we either remove the highly correlated predictors from the model, or use PCA to reduce dimensions of the model.)

3. Model fitting

The project tests a variety of ML based mathematical models to select a specific number of features in a suite of different statistical tests, and to predict risk of heart disease. The types of models evaluated here include logistic regression, Bayesian regression, decision tree, random forest, and boosting.
A first step is to split data, randommly, into training dataset (70%), to build model, and testing dataset (30%), to evaluate model performance.

```
set.seed(314159)
indexh <- createDataPartition(heart$Heart, p = 0.7, list = F) #70% for training
trainh <- heart[indexh,]
testh  <- heart[-indexh,]
```

3.1. Logistic Regression

Logistic regression is similar to linear regression, predicting outcome with a function of explanatory variables (heart disease risk factors). It is used in machine learning for classification problems, yielding binary outputs between pass/fail.

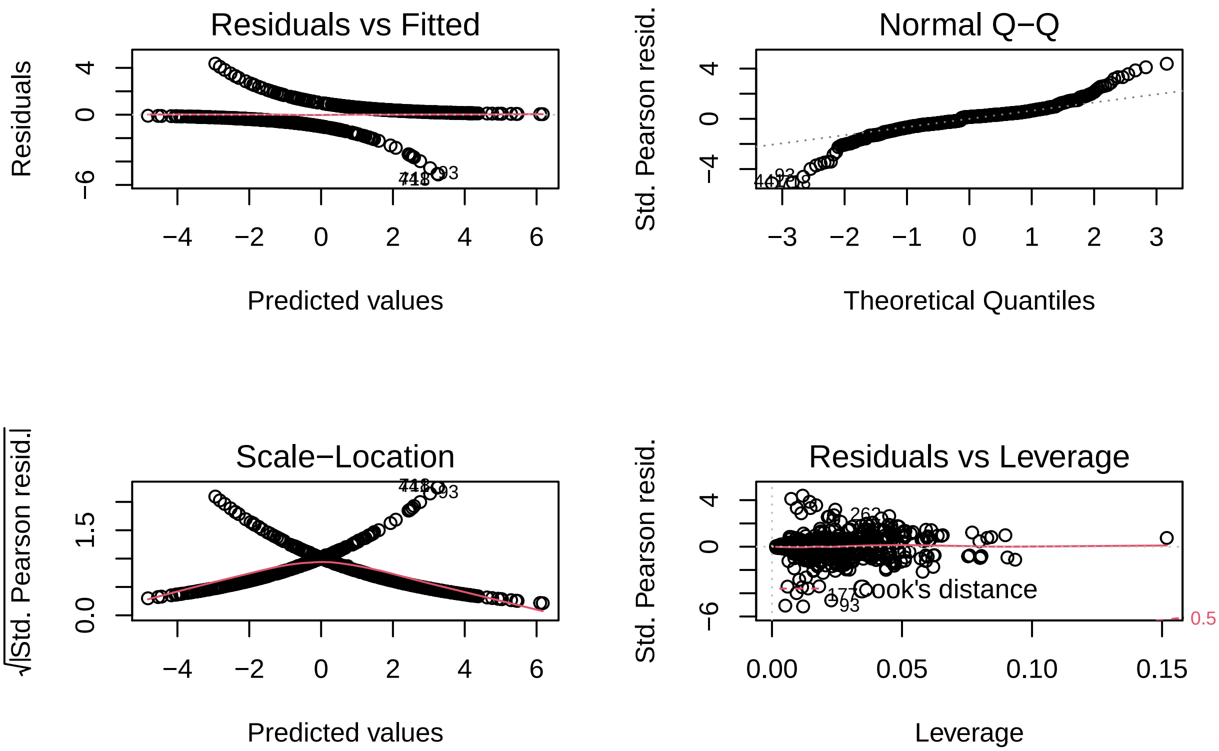
```r
#Logistic Regression
mod_finh <- glm(Heart~.,
                data = trainh, family = binomial(link = "logit"))
summary(mod_finh) #check for significance level in output, which indicates importance of risk factors
```

```
##
## Call:
## glm(formula = Heart ~ ., family = binomial(link = "logit"), data = trainh)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5683  -0.5844   0.2128   0.5888   2.4477
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.5383     1.1864  -4.668 3.04e-06 ***
## Age           1.1453     0.6603   1.734 0.082836 .
## Gender        1.1557     0.2991   3.864 0.000112 ***
## CP            2.3281     0.3820   6.094 1.10e-09 ***
## TBps          0.1775     1.2215   0.145 0.884461
## Chol         -2.0060     0.6369  -3.149 0.001636 **
## Fbs           0.4076     0.3240   1.258 0.208474
## Recg          0.2511     0.3016   0.833 0.405022
## Thalach      -1.4294     0.7385  -1.936 0.052929 .
## Exang         1.0569     0.2699   3.916 9.00e-05 ***
## Op            5.3521     1.1509   4.650 3.31e-06 ***
## Slope         0.6569     0.5334   1.232 0.218111
## Ca            2.6981     0.8169   3.303 0.000957 ***
## Thal          0.8885     0.3638   2.443 0.014585 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 885.58  on 643  degrees of freedom
## Residual deviance: 521.54  on 630  degrees of freedom
## AIC: 549.54
##
## Number of Fisher Scoring iterations: 5
```

```r
summary(residuals(mod_finh))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.56829 -0.58440  0.21277  0.02002  0.58878  2.44771
```

```r
par(mfrow = c(2, 2))
plot(mod_finh)
```



The Q-Q plot tests normality of residue.
The residual vs leverage plot helps to find influential cases, if any outliers that are influential in the regression model. We look at the upper right or lower right for if any cases as influential outliers.

Accuracy of model is evaluated with test data, checking % of correct predictions of original model comparing to outcome in test data.

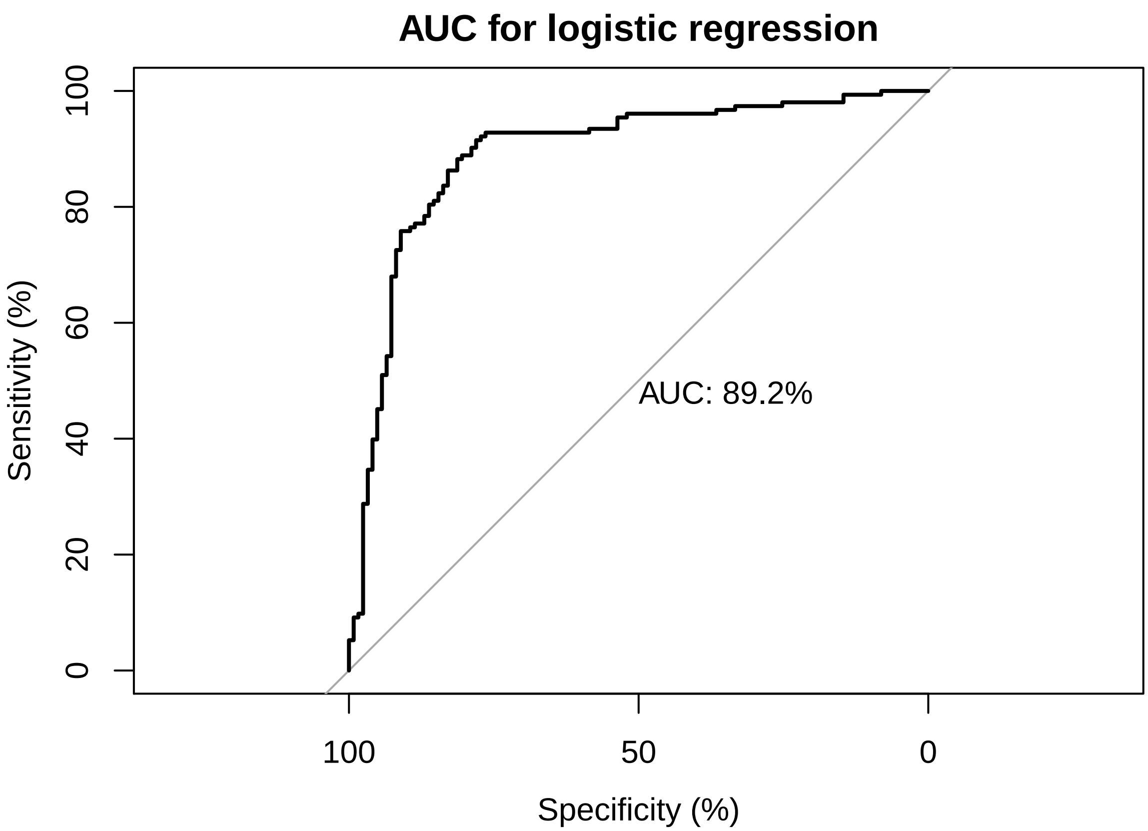
```
#evaluate accuracy on test data
testh_pred <- predict(mod_finh, testh, type = "response")
pred_testhh <- as.data.frame(cbind(testh$Heart, testh_pred))
colnames(pred_testhh) <- c("Original", "testh_pred")
pred_testhh$outcome <- ifelse(pred_testhh$testh_pred > 0.5, 1, 0)

#calculate accuracy
acc_lgh <- confusionMatrix(factor(testh$Heart), factor(pred_testhh$outcome)) $overall['Accuracy']
print(paste('logistic regression model accuracy is', round(acc_lgh, 2) * 100, '%'))
```

```
## [1] "logistic regression model accuracy is 85 %"
```

```
#AUC curve
par(mfrow = c(1, 1))
plot.roc(testh$Heart, testh_pred, percent = TRUE, print.auc = TRUE,
         main = "AUC for logistic regression")
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```



AUC curve is a performance measurement for the classification problems.It measures the ability of a classifier to distinguish between classes.

3.2. Bayesian Logistic Regression

Bayesian logistic regression is the Bayesian counterpart to the common logistic regression (Sean M O'Brien, 2004). In the Bayesian approach, out belief is updated and proportional to the prior likelihood. We arrive at a distribution of result estimation, rather than a single point estimate.

```r
#Bayesian Logistic Regression
#prior distribution, assuming student t dis
prior_disth <- student_t(df = 7, location = 0, scale = 2.5)
bayes_modh <- stan_glm(Heart~., data = trainh,
                       family = binomial(link = "logit"),
                       prior = prior_disth, prior_intercept = prior_disth,
                       seed = 314159)
```

```
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 1).
## Chain 1:
```

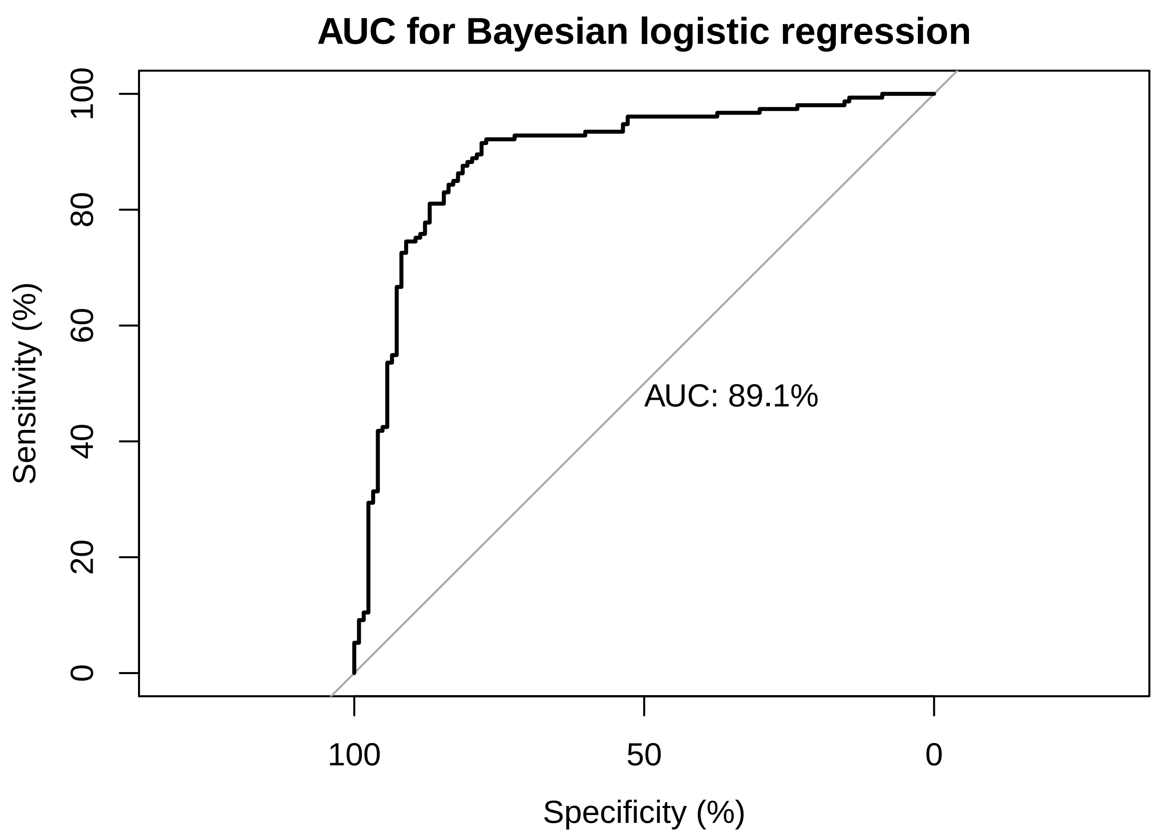
```
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.481 seconds (Warm-up)
## Chain 1:                0.554 seconds (Sampling)
## Chain 1:                1.035 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.519 seconds (Warm-up)
## Chain 2:                0.537 seconds (Sampling)
## Chain 2:                1.056 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
```

```
## Chain 3:
## Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.49 seconds (Warm-up)
## Chain 3:                0.517 seconds (Sampling)
## Chain 3:                1.007 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'bernoulli' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 0 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
## Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
## Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
## Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
## Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
## Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
## Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
## Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
## Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
## Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
## Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
## Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.5 seconds (Warm-up)
## Chain 4:                0.544 seconds (Sampling)
## Chain 4:                1.044 seconds (Total)
## Chain 4:
```

```r
bayes_resh <- data.frame(residuals(bayes_modh))
bayes_resh$indexh <- seq.int(nrow(bayes_resh))

pred <- posterior_linpred(bayes_modh, newdata = testh, transform = TRUE)
fin_predh <- colMeans(pred)
testh_prediction <- as.integer(fin_predh >= 0.5)

acc_bayesh <- confusionMatrix(factor(testh$Heart), factor(testh_prediction)) $overall['Accuracy']
print(paste('Bayesian logistic regression model accuracy is', round(acc_bayesh, 2) * 100, '%'))
```

```
## [1] "Bayesian logistic regression model accuracy is 85 %"
```

```
plot.roc(testh$Heart, fin_predh, percent = TRUE, print.auc = TRUE,
         main = "AUC for Bayesian logistic regression")
```

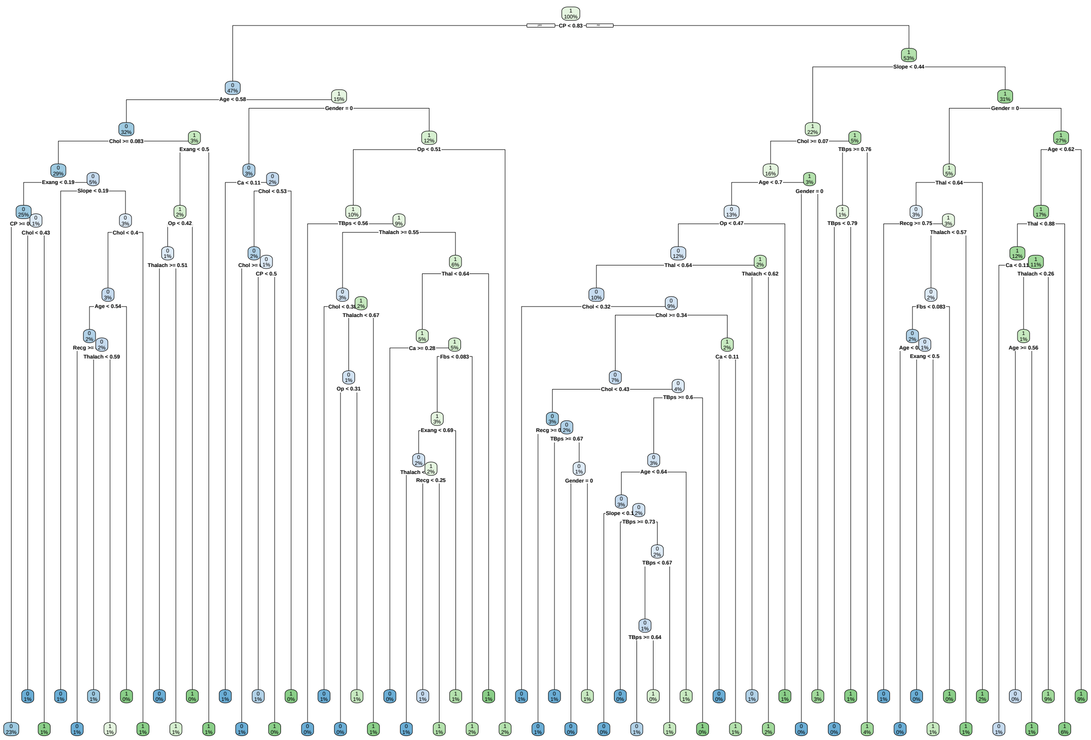


3.3. Decision Tree

Decision tree is where each leaf node acting as a classification model, and all branches combine to give an overall class label. Trees are constructed pri-oritizing the highest information gain, until all leaf nodes are pure.

```
#Decision Tree
library(rpart)
library(rpart.plot)

set.seed(314159)
fit <- rpart(Heart~.,
             data = trainh,
             method = "class",
             control = rpart.control(xval = 10,
                                     minbucket = 2,
                                     cp = 0),
```

```
             parms = list(split = "information"))

rpart.plot(fit, extra = 100)
```



```
pred_dt_testh <- predict(fit, testh, type = "class")
acc_dth <- confusionMatrix(factor(testh$Heart), factor(pred_dt_testh)) $overall['Accuracy']
print(paste('Decision tree model accuracy is', round(acc_dth, 2) * 100, '%'))
```

```
## [1] "Decision tree model accuracy is 76 %"
```

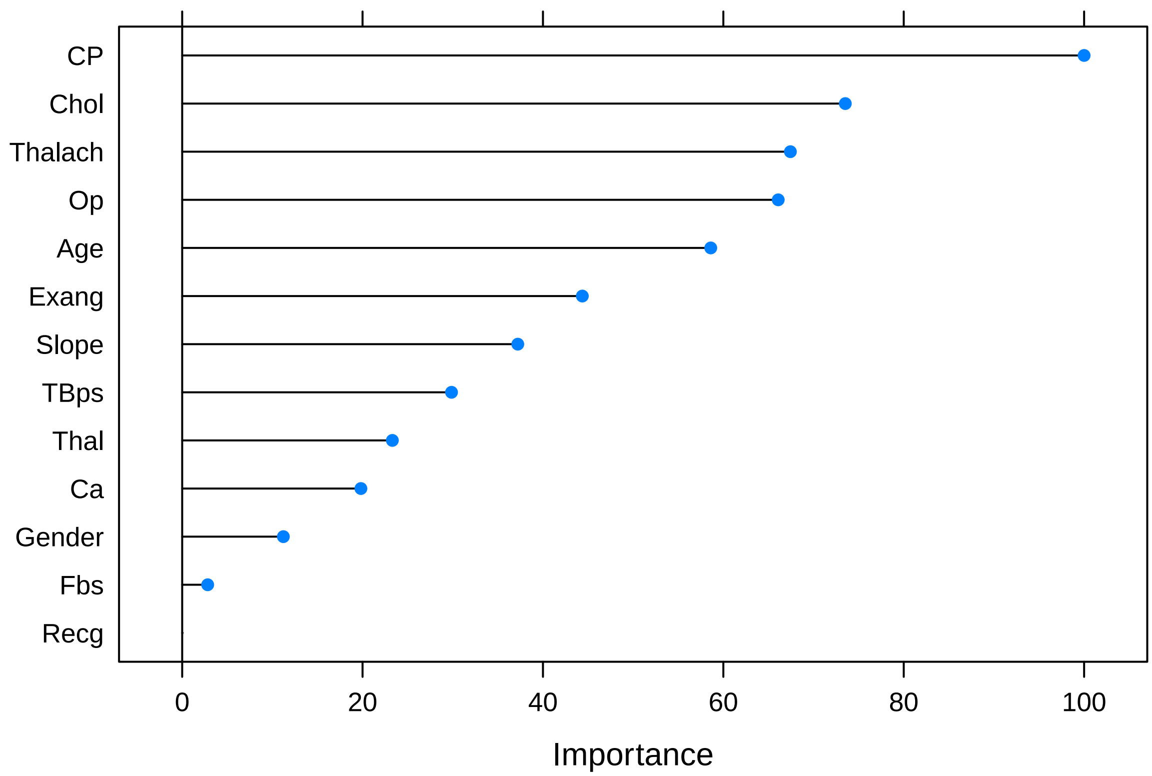
Decision tree tends to overfit with training data, and hence underperforms with testing data. Random Forest is a robust technique that employs multiple trees to vote an overall decision.

3.4. Random Forest

Random forest uses an ensemble of trees vote a "conclusion". It is generally more robust to outliers and overfitting, but also are more difficult to inter-pret, as to understand the most contributing variable(s) for the prediction.

```
#Random Forest
#random forest is an ensemble of decision trees, and usually takes longer to calculate
```

```r
heart$Heart <- as.factor(heart$Heart)
trainh$Heart <- as.factor(trainh$Heart)
testh$Heart <- as.factor(testh$Heart)

set.seed(314159)
model_rfh <- caret::train(Heart~.,
                          data = trainh,
                          method = "rf",
                          preProcess = c("scale", "center"),
                          trControl = trainControl(method ="repeatedcv",
                                                   number = 10,
                                                   repeats = 10,
                                                   savePredictions =TRUE,
                                                   verboseIter = FALSE))


importanceh <- varImp(model_rfh, scale = TRUE)
plot(importanceh)
```



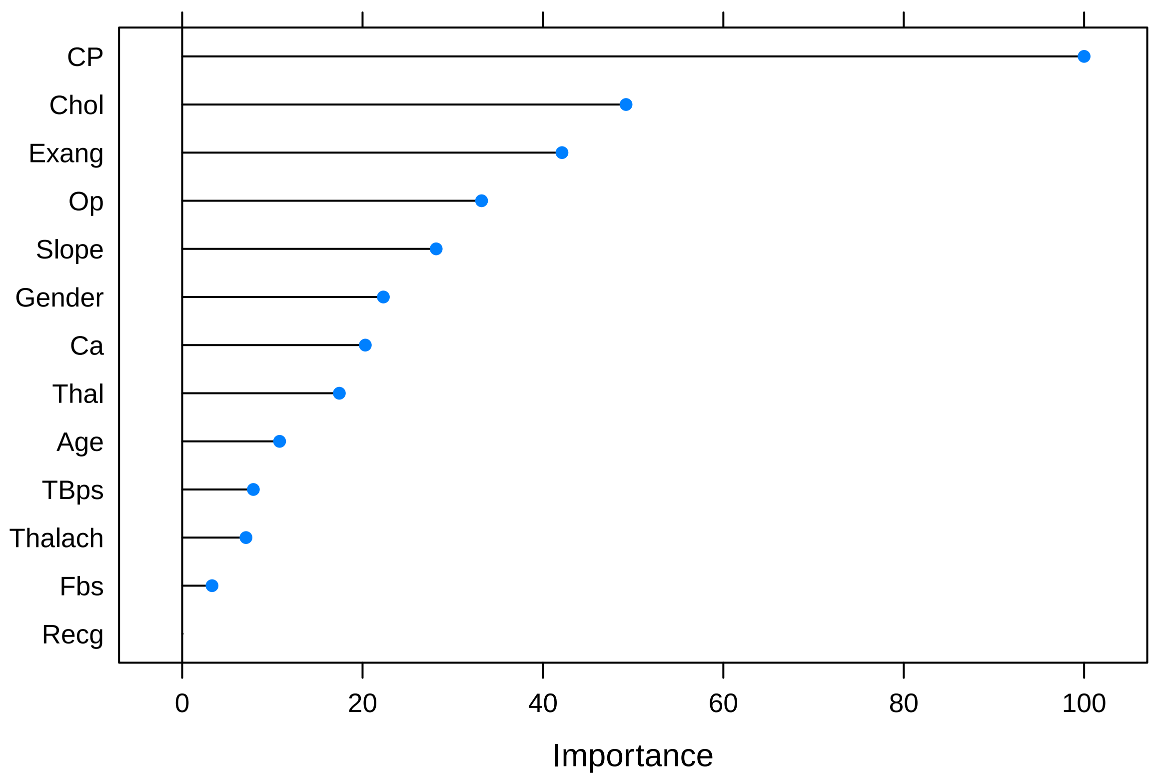

```r
acc_rfh <- confusionMatrix(predict(model_rfh, testh), testh$Heart) $overall['Accuracy']
print(paste('Random Forest model accuracy is', round(acc_rfh, 2) * 100, '%'))
```

```
## [1] "Random Forest model accuracy is 81 %"
```

3.5. Extreme gradient boosting

Extreme gradient boosting, as one of the most favorite algorithms at Kaggle, is a machine learning technique that optimizes model in a step-wise way by giving weights to the more important predictors.

```
#Extreme gradient boosting
model_xgbh <- caret::train(Heart~.,
                           data = trainh,
                           method = "xgbTree",
                           preProcess = c("scale", "center"),
                           trControl = trainControl(method = "repeatedcv",
                                                    number = 10,
                                                    repeats = 10,
                                                    savePredictions = TRUE,
                                                    verboseIter = FALSE))

imphortance <- varImp(model_xgbh, scale = TRUE)
plot(imphortance)
```

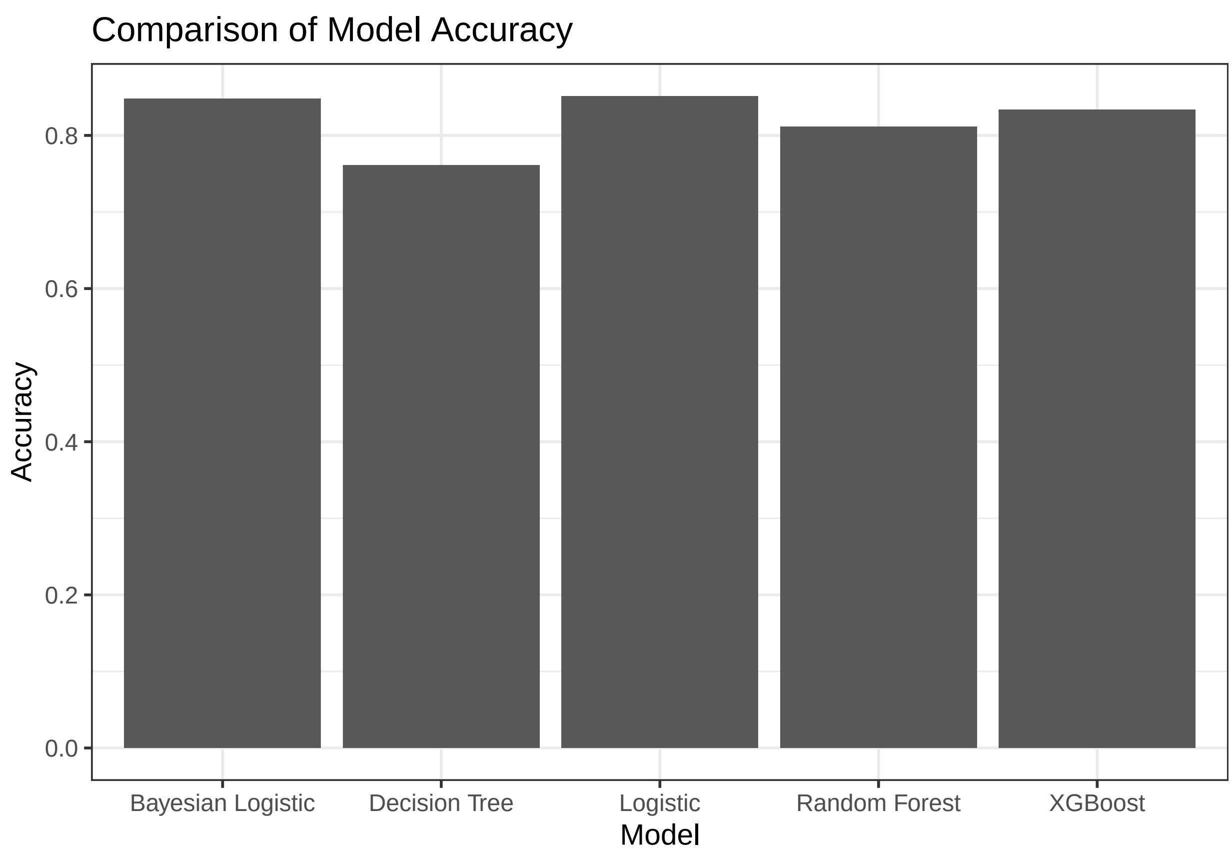



```
acc_xbgh <- confusionMatrix(predict(model_xgbh, testh), testh$Heart) $overall['Accuracy'];
print(paste('Extreme gradient boosting model accuracy is', round(acc_xbgh, 2) * 100, '%'))
```

```
## [1] "Extreme gradient boosting model accuracy is 83 %"
```

4. Comparison of Model Performance

```
##summarize accuracy of various models
accuracyh <- data.frame(Model = c("Logistic", "Bayesian Logistic", "Decision Tree",
                                  "Random Forest", "XGBoost"),
                        Accuracy = c(acc_lgh, acc_bayesh, acc_dth,
                                     acc_rfh, acc_xbgh))

acc_heart = ggplot(accuracyh, aes(x = Model, y = Accuracy)) + geom_bar(stat = 'identity') + theme_bw() +

acc_heart
```



Model | Accuracy ——————————- | ————- Logistic Regression | 0.8405797 Bayesian Logistic Regression | 0.8478261 Decision Tree | 0.7608696 Random Forest | 0.8115942 XGBoost | 0.8333333

```
#AUC curve is a performance measurement for the classification problems.It measures the ability of a cl
par(mfrow = c(2, 3))
plot.roc(testh$Heart, testh_pred, percent = TRUE, print.auc = TRUE,
         main = "AUC for logistic regression")
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot.roc(testh$Heart, fin_predh, percent = TRUE, print.auc = TRUE,
         main = "AUC for Bayesian logistic regression")
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
plot.roc(testh$Heart, as.numeric(pred_dt_testh), percent = TRUE, print.auc = TRUE,
         main = "AUC for Decision Tree model")
```

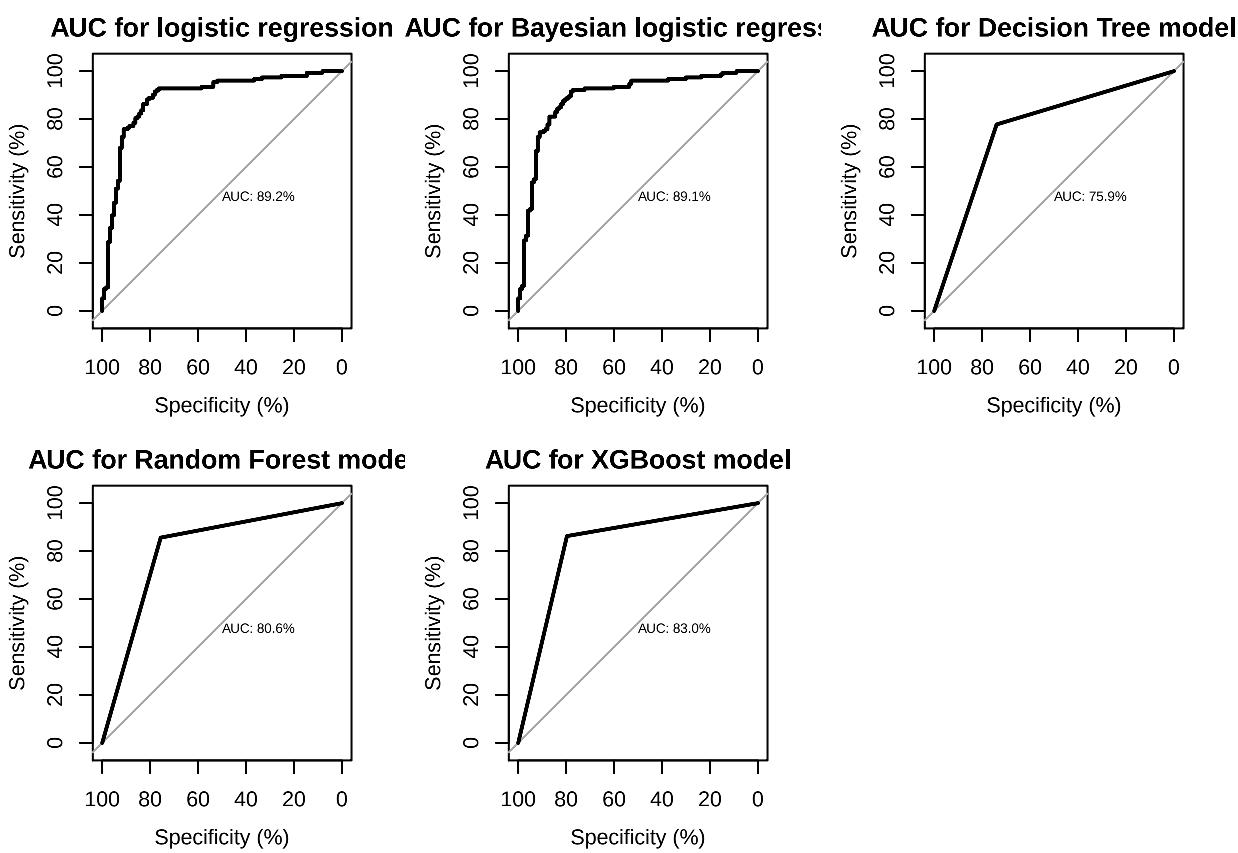
```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
plot.roc(testh$Heart, as.numeric(predict(model_rfh, testh)), percent = TRUE, print.auc = TRUE,
         main = "AUC for Random Forest model")
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```r
plot.roc(testh$Heart, as.numeric(predict(model_xgbh, testh)), percent = TRUE, print.auc = TRUE,
         main = "AUC for XGBoost model")
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



Based on above analysis, two models turn out to be most accurate, and highly reliable: Bayesian logistic

regression and Extreme gradient boosting. The aver-age accuracy is around 85%, and most relevant risk factors are CP (chest pain type), Chol (serum cholesterol level), Exang (exercise induced angina) and Op (ST depression induced by exercise relative to rest).
Decision tree models tend to overfit with training data's randomness, and hence underperforms with testing data.Random forest is often a good substitute, however, is also more difficult to interprate, especially when trying to look to models for more insight. A final product of an interactive application was made with R Shiny. The app was based on XGBoost as the optimal algorithm. The dashboard will take input values of the above risk factors to calculate heart disease risk; it would also dis-play plots to visualize relationships between risk factors and possibility of heart disease. The app can also be deployed to the Shiny cloud to be publicly accessible.

end_of_file