Random Forest

Lisa Skalon

11 june 2020

```
library(ggplot2)
library(randomForest)
library(MASS)
library(tree)
library(dplyr)
library(tidyr)
library(caret)
library(psych)
```

We are going to analyze open dataset with heart disease data, available at UCI (https://archive.ics.uci.edu/ml/datasets/Heart+Disease). The description of variables:

1. **age**: age in years
2. **sex**: sex (1 = male; 0 = female)
3. **cp**: chest pain type
    - Value 1: typical angina
    - Value 2: atypical angina
    - Value 3: non-anginal pain
    - Value 4: asymptomatic
4. **trestbps** : resting blood pressure (in mm Hg on admission to the hospital)
5. **chol**: serum cholestoral in mg/dl
6. **fbs**: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. **restecg**: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. **thalach**: maximum heart rate achieved
9. **exang**: exercise induced angina (1 = yes; 0 = no)
10. **oldpeak**: ST depression induced by exercise relative to rest
11. **slope**: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. **ca**: number of major vessels (0-3) colored by flourosopy
13. **thal**: 3 = normal; 6 = fixed defect; 7 = reversable defect
14. **num**: 0 = no heart disease, > 0 = heart disease

The variable wich we are going to predict is **num** - the presence or the absence of heart disease.

We will use randomForest algorithm - algorithm which works by aggregating the predictions made by multiple decision trees of varying depth. Every decision tree in the forest is trained on a subset of the dataset called the bootstrapped dataset.

```
# read df
df <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleve
names(df) <- c( "age", "sex", "cp", "trestbps", "chol","fbs", "restecg",
                  "thalach","exang", "oldpeak","slope", "ca", "thal", "num")

# deal with na
str(df)
```

```
## 'data.frame':    303 obs. of  14 variables:
##  $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
##  $ sex     : num  1 1 1 1 0 1 0 0 1 1 ...
##  $ cp      : num  1 4 4 3 2 2 4 4 4 4 ...
##  $ trestbps: 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 ...
##  $ restecg : 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 ...
##  $ oldpeak : 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 ...
##  $ num     : int  0 2 1 0 0 0 3 0 2 1 ...
```

```
sum(is.na(df))
```

```
## [1] 6
```

```
df <- drop_na(df)

# change num to binary as in the description
df$num[df$num > 0] <- 1


# change data types
df <- transform(df, trestbps = as.factor(sex), cp = as.factor(cp), fbs = as.factor(fbs),
                exang = as.factor(exang), restecg = as.factor(restecg),
                slope = as.factor(slope), ca = as.factor(ca), thal=as.factor(thal),
                sex = as.factor(sex), num=as.factor(num))


summary(df)
```

```
##       age        sex     cp      trestbps      chol       fbs     restecg
##  Min.   :29.00   0: 96   1: 23   0: 96    Min.   :126.0   0:254   0:147
##  1st Qu.:48.00   1:201   2: 49   1:201    1st Qu.:211.0   1: 43   1:  4
##  Median :56.00           3: 83            Median :243.0           2:146
```

```
##  Mean   :54.54          4:142           Mean   :247.4
##  3rd Qu.:61.00                          3rd Qu.:276.0
##  Max.   :77.00                          Max.   :564.0
##     thalach       exang       oldpeak      slope   ca       thal    num
##  Min.   : 71.0   0:200   Min.   :0.000   1:139   0:174   3:164   0:160
##  1st Qu.:133.0   1: 97   1st Qu.:0.000   2:137   1: 65   6: 18   1:137
##  Median :153.0           Median :0.800   3: 21   2: 38   7:115
##  Mean   :149.6           Mean   :1.056           3: 20
##  3rd Qu.:166.0           3rd Qu.:1.600
##  Max.   :202.0           Max.   :6.200
```

```
# check the number of diseased/healthy obs
table(df$num)
```

```
##
##   0   1
## 160 137
```

We divide the dataset 50/50 (train-test split)

```
set.seed(42)
sample <- sample.int(n=  nrow(df), size = floor(.5*nrow(df)), replace = F)
train <- df[sample,]
test <- df[-sample,]
```

First we run RF with basic parameters (mtry 3, ntree 500)

```
fit <- randomForest(num ~ ., data = train, importance = T)
fit
```

```
##
## Call:
##  randomForest(formula = num ~ ., data = train, importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 17.57%
## Confusion matrix:
##    0  1 class.error
## 0 58 14   0.1944444
## 1 12 64   0.1578947
```

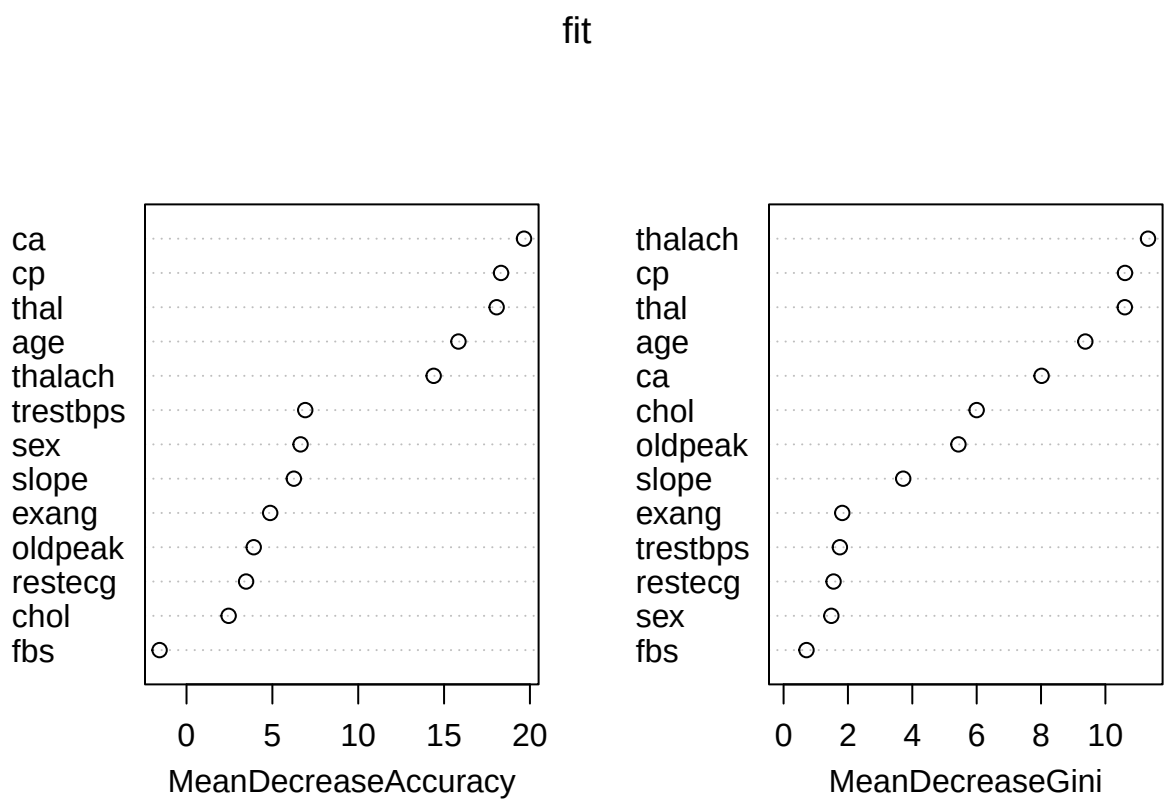
We can check the importance of each feature

```
importance(fit)
```

```
##                   0         1 MeanDecreaseAccuracy MeanDecreaseGini
## age      14.8089810  9.574592            15.838240        9.3758077
## sex       7.8435348  1.298506             6.642163        1.4835642
## cp       14.4619841 14.099231            18.318179       10.6095090
```

```
## trestbps  7.0115333  2.556693            6.914512        1.7498884
## chol      1.9941211  1.904328            2.452581        5.9997134
## fbs      -0.2719562 -2.273344           -1.569159        0.7116747
## restecg   1.8196997  3.476304            3.469078        1.5527951
## thalach   8.0886946 13.266171           14.394343       11.3245792
## exang     0.3261426  6.028456            4.873386        1.8234367
## oldpeak   1.1362873  4.348030            3.913843        5.4338251
## slope     3.8699134  5.035299            6.251089        3.7171380
## ca       16.1770447 14.303537           19.655888        8.0175218
## thal     14.3260000 13.861635           18.061538       10.6032061
```

```r
varImpPlot(fit)
```



Let's predict values in test dataset

```r
pred <- predict(fit, newdata = test[-14])
confusionMatrix(pred, test$num)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 70 13
##          1 18 48
##
```

```
##                Accuracy : 0.7919
##                  95% CI : (0.7179, 0.854)
##     No Information Rate : 0.5906
##     P-Value [Acc > NIR] : 1.525e-07
##
##                   Kappa : 0.5751
##
##  Mcnemar's Test P-Value : 0.4725
##
##             Sensitivity : 0.7955
##             Specificity : 0.7869
##          Pos Pred Value : 0.8434
##          Neg Pred Value : 0.7273
##              Prevalence : 0.5906
##          Detection Rate : 0.4698
##    Detection Prevalence : 0.5570
##       Balanced Accuracy : 0.7912
##
##        'Positive' Class : 0
##
```

We can tune mtry hyperparameter - number of variables randomly sampled as candidates of each split. We also can tune number of trees (manually).

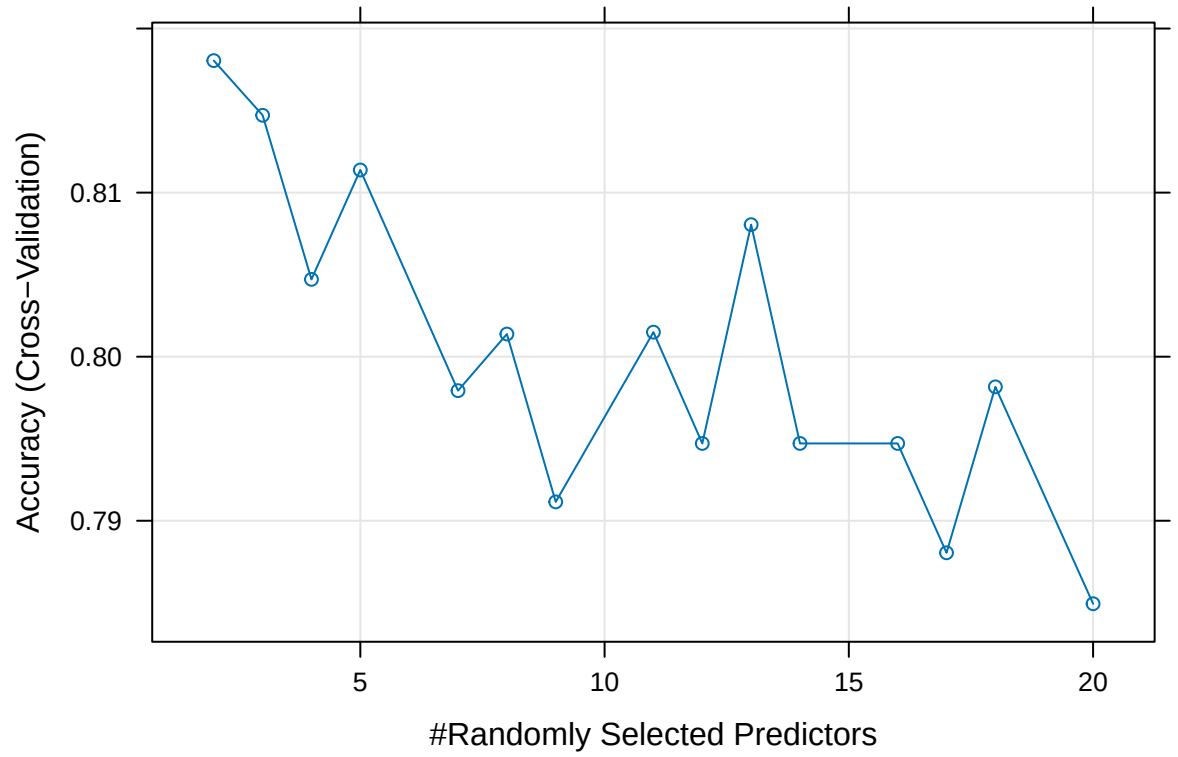
We will tune mtry using cross-validation and caret package

```
control <- trainControl(method = "cv",
                        number = 10)
rf_tune <- train(num ~ ., data = df, method = "rf", metric= 'Accuracy',
                 tuneLength=15, trControl=control)
print(rf_tune)
```

```
## Random Forest
##
## 297 samples
##  13 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 267, 267, 267, 268, 267, 267, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8180460  0.6322535
##    3    0.8147126  0.6267946
##    4    0.8047126  0.6058139
##    5    0.8113793  0.6194346
##    7    0.7979310  0.5923026
##    8    0.8013793  0.5994486
##    9    0.7911494  0.5780919
##   11    0.8014943  0.5993356
##   12    0.7947126  0.5852041
##   13    0.8080460  0.6121179
```

```
##    14    0.7947126  0.5850412
##    16    0.7947126  0.5852780
##    17    0.7880460  0.5712888
##    18    0.7981609  0.5920731
##    20    0.7849425  0.5660351
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```r
plot(rf_tune)
```



The best mtry = 2

```r
store_maxtrees <- list()

for (ntree in c(50, 100, 150, 200, 250, 300, 350, 400, 450, 500)) {
    rf_maxtrees <- train(num~.,
        data = df,
        method = "rf",
        metric = "Accuracy",
        trControl = control,
        importance = TRUE,
        ntree = ntree)
    key <- toString(ntree)
    store_maxtrees[[key]] <- rf_maxtrees
}
```

```r
results_tree <- resamples(store_maxtrees)
summary(results_tree)
```

```
##
## Call:
## summary.resamples(object = results_tree)
##
## Models: 50, 100, 150, 200, 250, 300, 350, 400, 450, 500
## Number of resamples: 10
##
## Accuracy
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 50  0.7333333 0.7948276 0.8500000 0.8354023 0.8666667 0.9310345    0
## 100 0.6333333 0.8000000 0.8275862 0.8185057 0.8583333 0.9000000    0
## 150 0.6666667 0.7666667 0.8333333 0.8256322 0.8620690 0.9666667    0
## 200 0.6206897 0.7732759 0.8304598 0.8074713 0.8666667 0.9000000    0
## 250 0.6896552 0.7396552 0.8166667 0.8078161 0.8583333 0.9333333    0
## 300 0.6666667 0.7948276 0.8275862 0.8181609 0.8583333 0.9333333    0
## 350 0.6896552 0.7500000 0.8333333 0.8114943 0.8655172 0.9000000    0
## 400 0.7000000 0.7732759 0.8310345 0.8150575 0.8655172 0.9000000    0
## 450 0.7000000 0.7732759 0.8333333 0.8287356 0.8548851 0.9655172    0
## 500 0.7333333 0.7750000 0.8000000 0.8051724 0.8534483 0.8666667    0
##
## Kappa
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 50  0.4594595 0.5849467 0.6969697 0.6669335 0.7297297 0.8605769    0
## 100 0.2533937 0.5991032 0.6436572 0.6322708 0.7175516 0.8000000    0
## 150 0.3243243 0.5216327 0.6651185 0.6464165 0.7231072 0.9327354    0
## 200 0.1924051 0.5390979 0.6603023 0.6073662 0.7309253 0.8000000    0
## 250 0.3587224 0.4806482 0.6260644 0.6099320 0.7113738 0.8660714    0
## 300 0.3421053 0.5812609 0.6489104 0.6328942 0.7116809 0.8648649    0
## 350 0.3680387 0.4932432 0.6636501 0.6184242 0.7275858 0.8000000    0
## 400 0.3835616 0.5422937 0.6555428 0.6251747 0.7283754 0.7982063    0
## 450 0.4000000 0.5343687 0.6666667 0.6530951 0.7112397 0.9297821    0
## 500 0.4545455 0.5445883 0.5945617 0.6040421 0.7017990 0.7321429    0
```

```r
fit2 <- randomForest(num ~ ., data = train,
                     mtry = 2, importance = T)
fit2
```

```
##
## Call:
##  randomForest(formula = num ~ ., data = train, mtry = 2, importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 15.54%
## Confusion matrix:
##    0  1 class.error
## 0 59 13   0.1805556
## 1 10 66   0.1315789
```

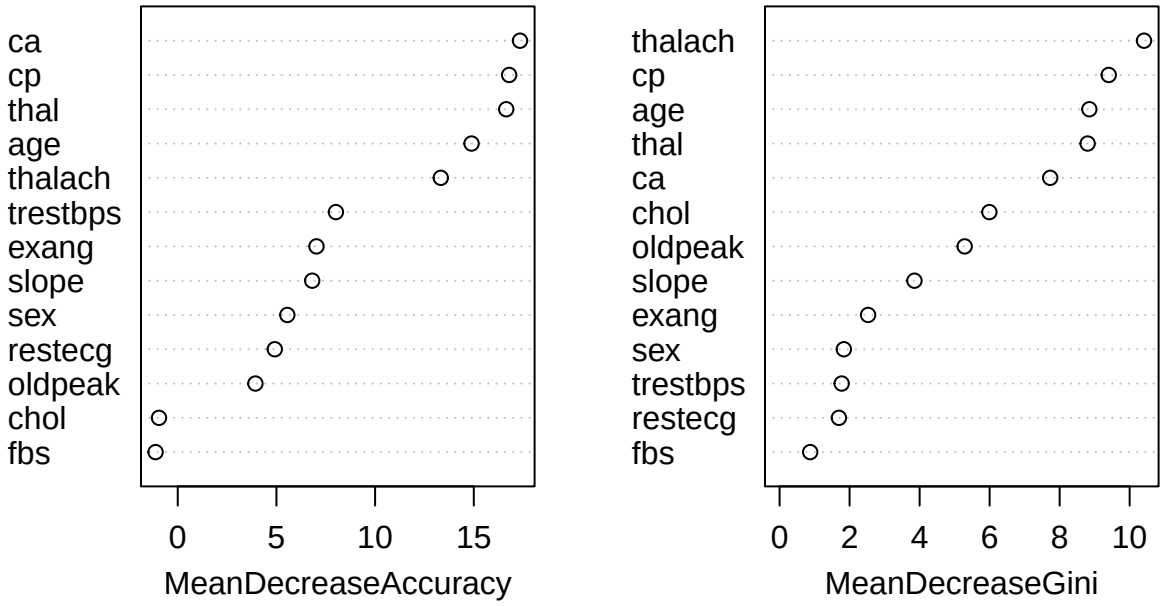
```r
pred2 <- predict(fit2, newdata = test[-14])
confusionMatrix(pred2, test$num)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 70 14
##          1 18 47
##
##                Accuracy : 0.7852
##                  95% CI : (0.7106, 0.8482)
##     No Information Rate : 0.5906
##     P-Value [Acc > NIR] : 3.992e-07
##
##                   Kappa : 0.5603
##
##  Mcnemar's Test P-Value : 0.5959
##
##             Sensitivity : 0.7955
##             Specificity : 0.7705
##          Pos Pred Value : 0.8333
##          Neg Pred Value : 0.7231
##              Prevalence : 0.5906
##          Detection Rate : 0.4698
##    Detection Prevalence : 0.5638
##       Balanced Accuracy : 0.7830
##
##        'Positive' Class : 0
##
```

```r
importance(fit2)
```

```
##                   0          1 MeanDecreaseAccuracy MeanDecreaseGini
## age      13.5780068  8.3185803           14.8867822        8.8487385
## sex       6.6697148  0.8899492            5.5516850        1.8363852
## cp       12.6467875 13.2461856           16.7908458        9.4021195
## trestbps  7.5192283  4.2228810            8.0134255        1.7746337
## chol      1.5231067 -2.5927451           -0.9503575        5.9931420
## fbs      -0.2066262 -1.3506021           -1.1231162        0.8743553
## restecg   2.2033101  4.2201209            4.9161070        1.6944428
## thalach   8.1037223 11.8305116           13.3278857       10.4105022
## exang     4.0189661  6.4167543            7.0296791        2.5283016
## oldpeak   2.4773911  3.0334468            3.9345549        5.2871829
## slope     4.2719206  5.8361600            6.8128590        3.8527299
## ca       14.3729393 13.4741544           17.3417426        7.7301143
## thal     13.3791725 11.9622913           16.6434965        8.7995670
```

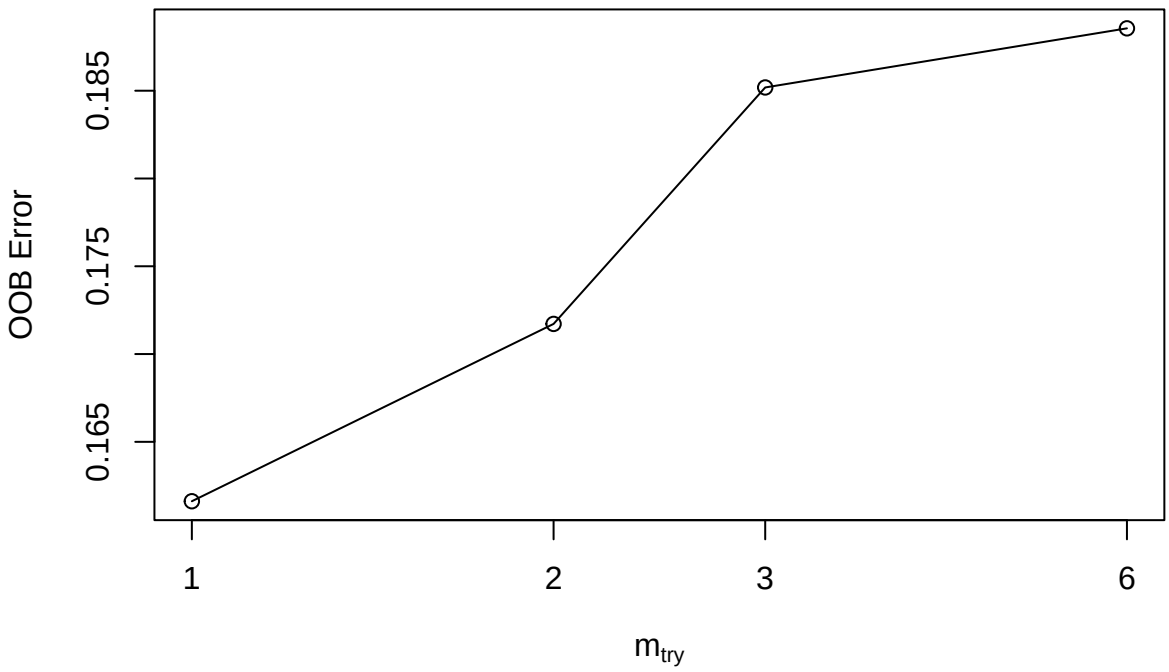
```r
varImpPlot(fit2)
```



We also can tune mtry with basic randomForest function tuneRF, but the resultis the same

```
mtry_best = tuneRF(df[-14], df$num)
```

```
## mtry = 3  OOB error = 18.52%
## Searching left ...
## mtry = 2     OOB error = 17.17%
## 0.07272727 0.05
## mtry = 1     OOB error = 16.16%
## 0.05882353 0.05
## Searching right ...
## mtry = 6     OOB error = 18.86%
## -0.1666667 0.05
```



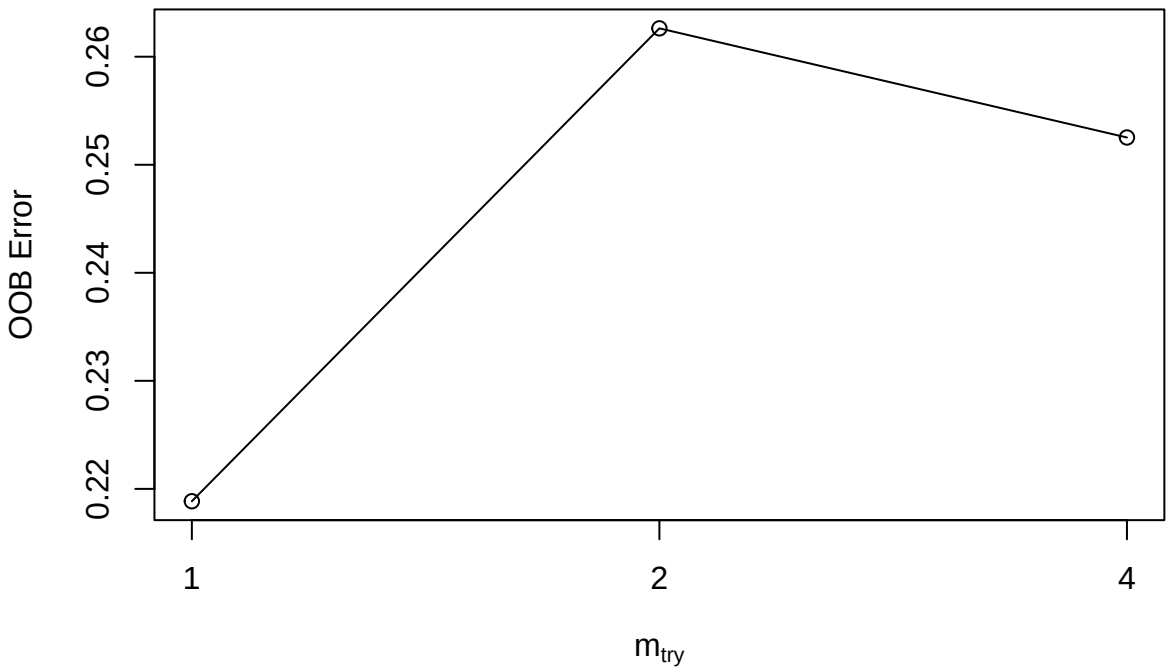

```
print(mtry_best)
```

```
##       mtry  OOBError
## 1.OOB    1 0.1616162
## 2.OOB    2 0.1717172
## 3.OOB    3 0.1851852
## 6.OOB    6 0.1885522
```

Now we will try to use feture selection and tune mtry for feature selected df

```
mtry_best = tuneRF(df[, c("cp", "thal", "age", "thalach", "oldpeak")], df$num)
```

```
## mtry = 2  OOB error = 26.26%
## Searching left ...
## mtry = 1     OOB error = 21.89%
## 0.1666667 0.05
## Searching right ...
## mtry = 4     OOB error = 25.25%
## -0.1538462 0.05
```




```
train_best <- train[,c("cp", "thal", "age", "thalach", "oldpeak", "num")]
fit_best <- randomForest(num ~ ., data=train_best,
                   mtry = 2, importance = T, ntry=500)
fit_best
```

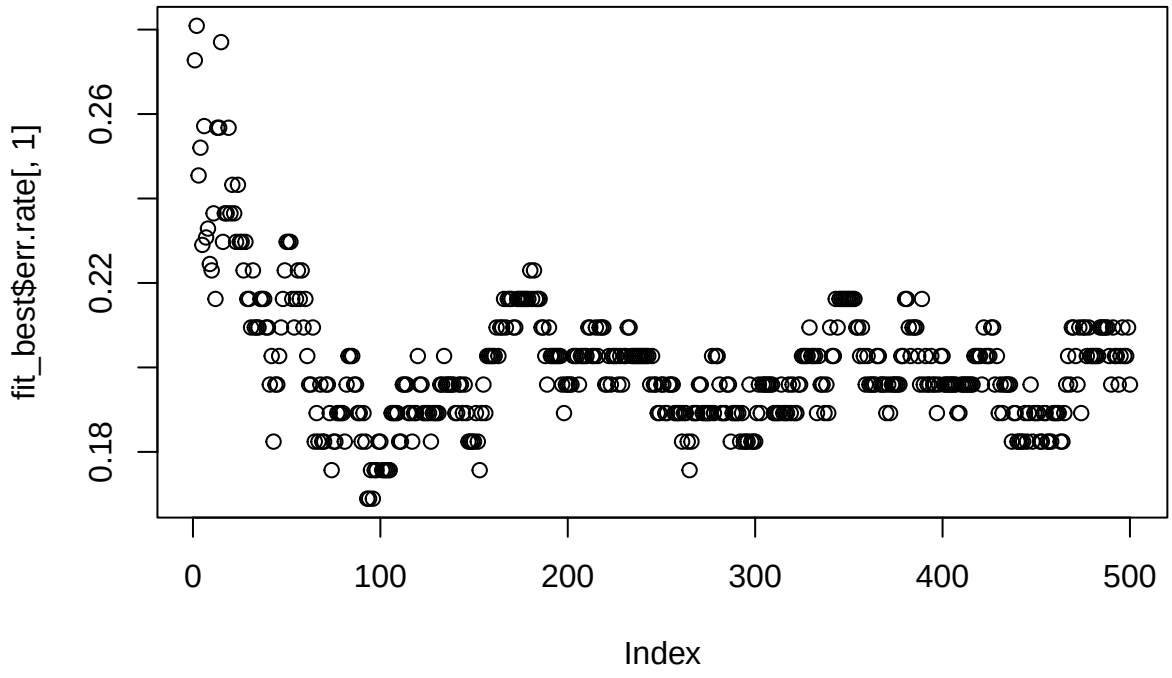
```
##
## Call:
##  randomForest(formula = num ~ ., data = train_best, mtry = 2,      importance = T, ntry = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 19.59%
## Confusion matrix:
##    0  1 class.error
## 0 58 14   0.1944444
## 1 15 61   0.1973684
```

```
pred_best <- predict(fit_best, newdata = test[-14])
confusionMatrix(pred_best, test$num)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
```

```
##          0 61 14
##          1 27 47
##
##                Accuracy : 0.7248
##                  95% CI : (0.6457, 0.7947)
##     No Information Rate : 0.5906
##     P-Value [Acc > NIR] : 0.0004517
##
##                   Kappa : 0.449
##
##  Mcnemar's Test P-Value : 0.0609187
##
##             Sensitivity : 0.6932
##             Specificity : 0.7705
##          Pos Pred Value : 0.8133
##          Neg Pred Value : 0.6351
##              Prevalence : 0.5906
##          Detection Rate : 0.4094
##    Detection Prevalence : 0.5034
##       Balanced Accuracy : 0.7318
##
##        'Positive' Class : 0
##
```

```
plot(fit_best$err.rate[,1])
```



```r
df_best=df[,c("cp", "thal", "age", "thalach", "oldpeak", "num")]
store_maxtrees <- list()

for (ntree in c(50, 100, 150, 200, 250, 300, 350, 400, 450, 500)) {
    rf_maxtrees <- train(num~.,
        data = df_best,
        method = "rf",
        metric = "Accuracy",
        trControl = control,
        importance = TRUE,
        ntree = ntree)
    key <- toString(ntree)
    store_maxtrees[[key]] <- rf_maxtrees
}
results_tree <- resamples(store_maxtrees)
summary(results_tree)
```

```
##
## Call:
## summary.resamples(object = results_tree)
##
## Models: 50, 100, 150, 200, 250, 300, 350, 400, 450, 500
## Number of resamples: 10
##
## Accuracy
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 50  0.6666667 0.7347701 0.7798851 0.7914943 0.8500000 0.9310345    0
## 100 0.5862069 0.7666667 0.7798851 0.7606897 0.8000000 0.8275862    0
## 150 0.6333333 0.7396552 0.7666667 0.7777011 0.8206897 0.9000000    0
## 200 0.7000000 0.7333333 0.7500000 0.7679310 0.7864943 0.8666667    0
## 250 0.7241379 0.7416667 0.7666667 0.7914943 0.8318966 0.8965517    0
## 300 0.7333333 0.7666667 0.8000000 0.7916092 0.8206897 0.8275862    0
## 350 0.6666667 0.7333333 0.8000000 0.7886207 0.8620690 0.9000000    0
## 400 0.6000000 0.7416667 0.7931034 0.7880460 0.8206897 0.9333333    0
## 450 0.7000000 0.7333333 0.7793103 0.7878161 0.8333333 0.9000000    0
## 500 0.7000000 0.7732759 0.8000000 0.7949425 0.8206897 0.8620690    0
##
## Kappa
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 50  0.3243243 0.4688994 0.5565273 0.5783101 0.6977558 0.8585366    0
## 100 0.1386139 0.5301943 0.5575321 0.5141613 0.5945946 0.6489104    0
## 150 0.2533937 0.4770926 0.5270174 0.5487789 0.6334718 0.7982063    0
## 200 0.3891403 0.4507923 0.4945863 0.5289385 0.5685851 0.7368421    0
## 250 0.4502370 0.4758163 0.5311659 0.5785667 0.6589596 0.7923628    0
## 300 0.4495413 0.5312222 0.5945946 0.5773536 0.6332435 0.6489104    0
## 350 0.3119266 0.4630156 0.5909091 0.5716971 0.7201337 0.7963801    0
## 400 0.1964286 0.4768816 0.5756098 0.5698321 0.6353315 0.8672566    0
## 450 0.3946188 0.4520208 0.5532111 0.5699136 0.6666667 0.7963801    0
## 500 0.3891403 0.5437804 0.5982143 0.5845752 0.6371253 0.7128713    0
```

We also can adjust ntree.

```r
fit_n <- randomForest(num ~ ., data = train_best,
                      mtry = 2, ntree = 250, importance = T)
fit_n
```

```
##
## Call:
##  randomForest(formula = num ~ ., data = train_best, mtry = 2,      ntree = 250, importance = T)
##                Type of random forest: classification
##                      Number of trees: 250
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 21.62%
## Confusion matrix:
##    0  1 class.error
## 0 57 15   0.2083333
## 1 17 59   0.2236842
```

```r
pred_n <- predict(fit_n, newdata = test[-14])
confusionMatrix(pred_n, test$num)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 63 14
##          1 25 47
##
##                Accuracy : 0.7383
##                  95% CI : (0.66, 0.8068)
##     No Information Rate : 0.5906
##     P-Value [Acc > NIR] : 0.0001202
##
##                   Kappa : 0.4733
##
##  Mcnemar's Test P-Value : 0.1093146
##
##             Sensitivity : 0.7159
##             Specificity : 0.7705
##          Pos Pred Value : 0.8182
##          Neg Pred Value : 0.6528
##              Prevalence : 0.5906
##          Detection Rate : 0.4228
##    Detection Prevalence : 0.5168
##       Balanced Accuracy : 0.7432
##
##        'Positive' Class : 0
##
```

```r
fit_n <- randomForest(num ~ ., data = train_best,
                      mtry = 2, ntree = 100, importance = T)
fit_n
```

```
##
## Call:
##  randomForest(formula = num ~ ., data = train_best, mtry = 2,      ntree = 100, importance = T)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 20.95%
## Confusion matrix:
##    0  1 class.error
## 0 58 14   0.1944444
## 1 17 59   0.2236842
```

```
pred_n <- predict(fit_n, newdata = test[-14])
confusionMatrix(pred_n, test$num)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 62 14
##          1 26 47
##
##                Accuracy : 0.7315
##                  95% CI : (0.6529, 0.8008)
##     No Information Rate : 0.5906
##     P-Value [Acc > NIR] : 0.0002368
##
##                   Kappa : 0.4611
##
##  Mcnemar's Test P-Value : 0.0819903
##
##             Sensitivity : 0.7045
##             Specificity : 0.7705
##          Pos Pred Value : 0.8158
##          Neg Pred Value : 0.6438
##              Prevalence : 0.5906
##          Detection Rate : 0.4161
##    Detection Prevalence : 0.5101
##       Balanced Accuracy : 0.7375
##
##        'Positive' Class : 0
##
```