Random Forest Classification in R

Heaven Klair

6/17/2022

Random Forest Classification in R

Importing the libraries

```
## randomForest 4.7-1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

For this Random Forest Classification, we are going to use the dataset from UCI Machine Learning Repository:

Reading the dataset

```
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data
data <- read.csv(url, header=FALSE)
head(data)
```

```
##   V1 V2 V3  V4  V5 V6 V7  V8 V9 V10 V11 V12 V13 V14
## 1 63  1  1 145 233  1  2 150  0 2.3   3 0.0 6.0   0
## 2 67  1  4 160 286  0  2 108  1 1.5   2 3.0 3.0   2
## 3 67  1  4 120 229  0  2 129  1 2.6   2 2.0 7.0   1
## 4 37  1  3 130 250  0  0 187  0 3.5   3 0.0 3.0   0
## 5 41  0  2 130 204  0  2 172  0 1.4   1 0.0 3.0   0
## 6 56  1  2 120 236  0  0 178  0 0.8   1 0.0 3.0   0
```

None of the columns are labeled in this dataset, so first, we need to name them. The information about the columns is given on the webpage.

```
colnames(data) <- c(
  "age",
  "sex",
  "cp",
  "trestbps",
  "chol",
  "fbs",
  "restecg",
  "thalach",
  "exang",
  "oldpeak",
  "slope",
  "ca",
  "thal",
  "hd"
)

head(data)
```

```
##   age sex cp trestbps chol fbs restecg thalach exang oldpeak slope  ca thal hd
## 1  63   1  1      145  233   1       2     150     0     2.3     3 0.0  6.0  0
## 2  67   1  4      160  286   0       2     108     1     1.5     2 3.0  3.0  2
## 3  67   1  4      120  229   0       2     129     1     2.6     2 2.0  7.0  1
## 4  37   1  3      130  250   0       0     187     0     3.5     3 0.0  3.0  0
## 5  41   0  2      130  204   0       2     172     0     1.4     1 0.0  3.0  0
## 6  56   1  2      120  236   0       0     178     0     0.8     1 0.0  3.0  0
```

Now, we have the column names fixed.

Let us describe the structure of the dataset, and see if there is any useful information in there.

```
str(data)
```

```
## '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      : chr  "0.0" "3.0" "2.0" "0.0" ...
##  $ thal    : chr  "6.0" "3.0" "7.0" "3.0" ...
##  $ hd      : int  0 2 1 0 0 0 3 0 2 1 ...
```

We can notice that "sex" column is suppose to represented by a factor, where 0 represents "female" and 1 represents "male". Instead it has a numeric value.

"cp" (aka chest pain) is also suppose to be a factor where levels 1-3 represents different types of pain and 4 represents no chest pain.

"ca" and "thal" are correctly called factors, but one of the levels is "?" when it needs to be NA.

Cleaning the Dataset

```
data[data == "?"] <- NA     # changes the "?" to NA
data[data$sex == 0,]$sex <- "F"       # convert 0 to "F"
data[data$sex == 1,]$sex <- "M"       # convert 1 to "M"
data$sex <- as.factor(data$sex)       # Convert the column "sex" to a factor

data$cp <- as.factor(data$cp)
data$fbs <- as.factor(data$fbs)
data$restecg <- as.factor(data$restecg)
data$exang <- as.factor(data$exang)
data$slope <- as.factor(data$slope)
```

Since the “ca” column originally had “?” in it, rather than NA, R thinks its is a column of strings. We correct this assumption by telling R that its a column of integers.

```
data$ca <- as.integer(data$ca)
data$ca <- as.factor(data$ca)
```

We will do the same thing with “thal”

```
data$thal <- as.integer(data$thal)
data$thal <- as.factor(data$thal)
```

The last thing we need to do to the data is make “hd” (Heart Disease), a factor that is easy on the eyes. Here we are going to use a fancy trick with ifelse() to convert the 0s to “Healthy” and 1s to “Unhealthy”.

```
data$hd <- ifelse(test = data$hd == 0, yes="Healthy", no="Unhealthy")
data$hd <- as.factor(data$hd)
```

Now we are done fixing the dataset. Let’s see the structure of the dataset.

```
str(data)
```

```
## 'data.frame':    303 obs. of  14 variables:
##  $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
##  $ sex     : Factor w/ 2 levels "F","M": 2 2 2 2 1 2 1 1 2 2 ...
##  $ cp      : Factor w/ 4 levels "1","2","3","4": 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     : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 2 ...
##  $ restecg : Factor w/ 3 levels "0","1","2": 3 3 3 1 3 1 3 1 3 3 ...
##  $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
##  $ exang   : Factor w/ 2 levels "0","1": 1 2 2 1 1 1 1 2 1 2 ...
##  $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
##  $ slope   : Factor w/ 3 levels "1","2","3": 3 2 2 3 1 1 3 1 2 3 ...
##  $ ca      : Factor w/ 4 levels "0","1","2","3": 1 4 3 1 1 1 3 1 2 1 ...
##  $ thal    : Factor w/ 3 levels "3","6","7": 2 1 3 1 1 1 1 1 3 3 ...
##  $ hd      : Factor w/ 2 levels "Healthy","Unhealthy": 1 2 2 1 1 1 2 1 2 2 ...
```

Since we are going to randomly sampling things, let’s set the seed for the random number generator so that we can reproduce our results.

After that, we are going to impute values for the NA in the dataset with **rfImput()**. It is used to impute the missing values in predictor data using proximity from randomForest.

```
set.seed(42)
data.imputed <- rfImpute(hd ~ ., data = data, iter=6)
```

```
## ntree      OOB      1      2
##   300:  17.49% 12.80% 23.02%
## ntree      OOB      1      2
##   300:  16.83% 14.02% 20.14%
## ntree      OOB      1      2
##   300:  17.82% 13.41% 23.02%
## ntree      OOB      1      2
##   300:  17.49% 14.02% 21.58%
## ntree      OOB      1      2
##   300:  17.16% 12.80% 22.30%
## ntree      OOB      1      2
##   300:  18.15% 14.63% 22.30%
```

The first argument to rfImpute is **hd**. That means we want to the hd column to be predicted by the data in all of the other column. The third argument **iter** is the number of random Forests rfImpute should build to estimate the missing values. In theory, 4 to 6 iterations is enough. Lastly, we save the results, the dataset with imputed values instead of NAs as data.imputed.

After each iteration, **rfImpute()** prints out the Out-of-Bag (OOB) error rate. This should get smaller if the estimates are improving. Since it doesn't, we can conclude our estimates are as good as they are going to get with this method.

Buildling Random Forest

Now, we will build the **randomForest()**.

```r
model <- randomForest(hd ~ ., data = data.imputed, proximity = TRUE)
```

We want **randomForest()** to return the proximity matrix. We will use the *proximity = TRUE* to cluster the samples at the end.

```r
model
```

```
##
## Call:
##  randomForest(formula = hd ~ ., data = data.imputed, proximity = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 16.83%
## Confusion matrix:
##           Healthy Unhealthy class.error
## Healthy       142        22   0.1341463
## Unhealthy      29       110   0.2086331
```

"No. of variables tried at each split" tells us how variables (or columns of data) were considered at each internal node.

Classification trees have a default setting of the square root of the number of variables while Regression trees have a default setting of the number of variables divided by 3.

We do not know if 3 is the best value for us, so we can fiddle with this parameter.

"OOB estimate of error rate" tells us that $100 - 16.5 = 83.5$ of the OOB samples were correctly classified by the random forest.

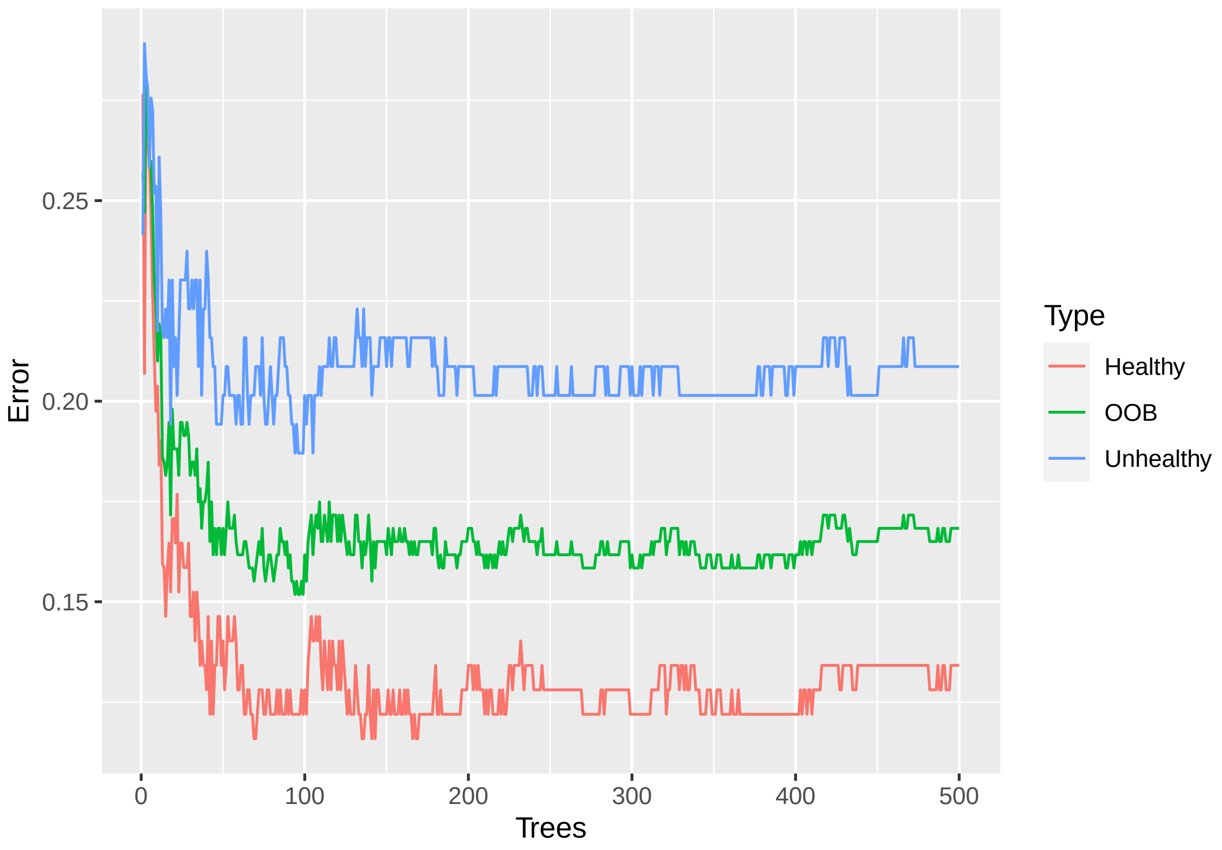
Tuning Parameters (ntrees with respect visualizing error rates)

To see if 500 trees is enough for optimal classification, we can plot the error rates.

```r
oob.error.data <- data.frame(
  Trees = rep(1:nrow(model$err.rate), times=3),
  Type = rep(c("OOB", "Healthy", "Unhealthy"), each = nrow(model$err.rate)),
  Error = c(model$err.rate[, "OOB"],
            model$err.rate[, "Healthy"],
            model$err.rate[, "Unhealthy"])
)

# The above code creates a dataframe that formats the error rate information so that ggplot2 will be ha

ggplot(data = oob.error.data, aes(x=Trees, y=Error)) +
  geom_line(aes(color=Type))
```



Walkthrough of the above code block: For the most part, this is all based on a matrix within model called **err.rate**. This is what the **err.rate** matrix looks like

```
head(model$err.rate)
```

```
##              OOB   Healthy Unhealthy
## [1,] 0.2571429 0.2765957 0.2413793
## [2,] 0.2470588 0.2068966 0.2891566
## [3,] 0.2783019 0.2758621 0.2812500
## [4,] 0.2704918 0.2647059 0.2777778
## [5,] 0.2592593 0.2600000 0.2583333
## [6,] 0.2597865 0.2467532 0.2755906
```

There is first column for the OOB error rate, one column for the Healthy error rate (i.e. how frequently healthy patients are misclassified), one column for Unhealthy error rate (i.e. how frequently unhealthy patients are misclassified). Each row reflects the error rate at different stages of creating the random forest. The first row contains the error rate after making the first tree. The second row contains the error rate after making the first two trees. The last row contains the error rate after making the 500th tree.

In general, we see that the error rate decreases when our random forest has more trees. If we added more trees, will the error go down further?

To test this hypothesis, we can change the number of trees in the model and plot the error rates again.

```
model <- randomForest(hd ~ ., data = data.imputed, ntree=1000, proximity = TRUE)
# Random Forest with 1000 trees

model
```

```
##
```

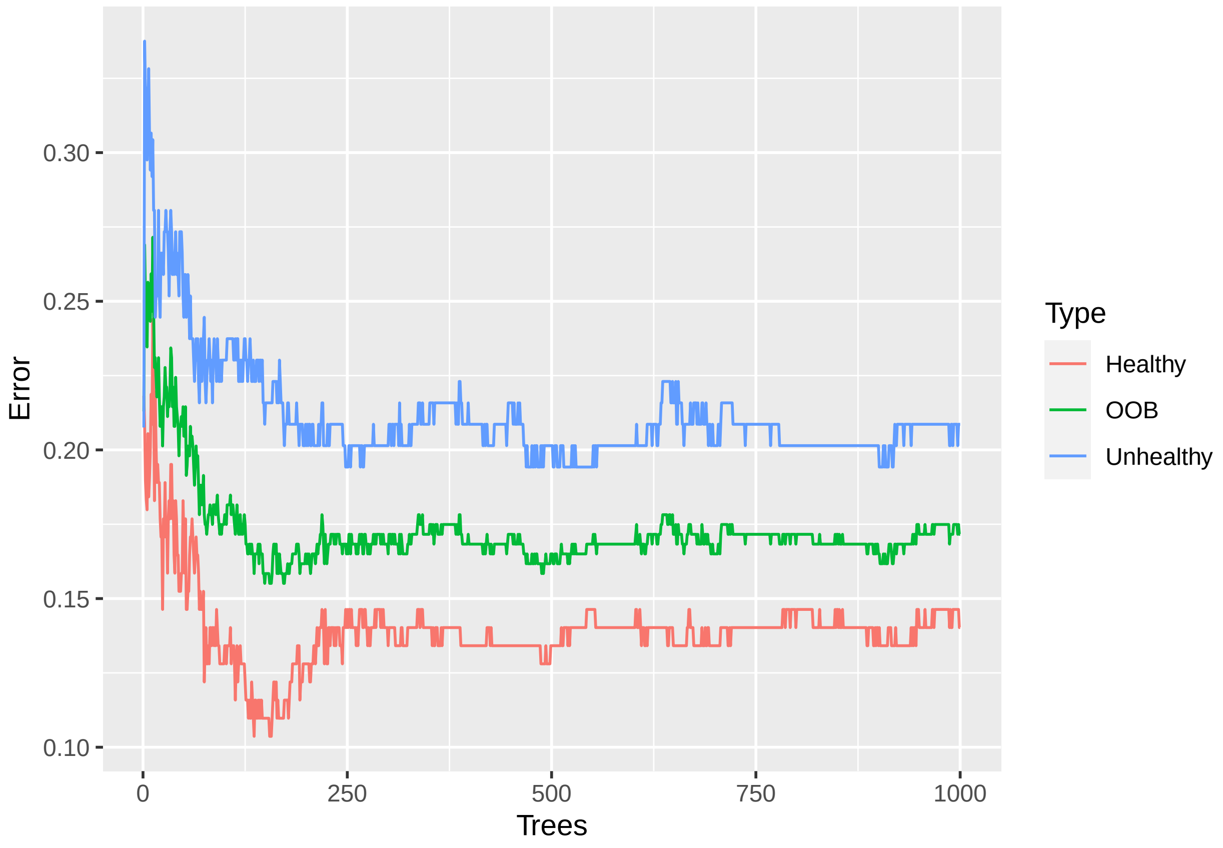
```
## Call:
##  randomForest(formula = hd ~ ., data = data.imputed, ntree = 1000,      proximity = TRUE)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 17.16%
## Confusion matrix:
##           Healthy Unhealthy class.error
## Healthy       141        23   0.1402439
## Unhealthy      29       110   0.2086331
```

We see that the OOB estimate of error rate is still 16.5 which means that adding more numbe of trees does not change the error rates. We can still plot the error rate just like before.

```r
oob.error.data <- data.frame(
  Trees = rep(1:nrow(model$err.rate), times=3),
  Type = rep(c("OOB", "Healthy", "Unhealthy"), each = nrow(model$err.rate)),
  Error = c(model$err.rate[, "OOB"],
            model$err.rate[, "Healthy"],
            model$err.rate[, "Unhealthy"])
)

# The above code creates a dataframe that formats the error rate information so that ggplot2 will be ha

ggplot(data = oob.error.data, aes(x=Trees, y=Error)) +
  geom_line(aes(color=Type))
```



Thus, we can see that the error rates after 500 trees stays the same.

Tuning Parameters (No of variables tried at each split)

Now we need to make sure we are considering the optimal number of variables at each split at each internal node in the tree. We will start by making an empty vector that can hold 10 values. Then we create a loop that tests different numbers of variables at each step.

```
oob.values <- vector(length = 10)

for(i in 1:10){
  temp.model <- randomForest(hd ~., data = data.imputed, mtry=i, ntree = 1000)
  oob.values[i] <- temp.model$err.rate[nrow(temp.model$err.rate), 1]
}
# In the first line, we are building a random forest using "i" to determine the number of # variables t

# The second line is where we store the OOB error rate after we build each random Forest # that uses a

# temp.model contains a matrix called err.rate (just like before), and we want to access the value in t

oob.values
```

```
##  [1] 0.1749175 0.1782178 0.1716172 0.1782178 0.1683168 0.1881188 0.1815182
##  [8] 0.1980198 0.1782178 0.1848185
```

The default value for mtry is 3, and that is when we see that error rate is least in the obb.values vector. Thus, the default value is the optimal value to be used here.

MDS Plot with samples

Lastly, we want to use the random forest to draw an MDS plot with samples. This will show us how they are related to each other.

```
distance.matrix <- dist(1 - model$proximity) # we start by using the dist() to make a distance matrix f
mds.stuff <- cmdscale(distance.matrix, eig = TRUE, x.ret = TRUE) # Then we run cmdscale() on the distan
```

cmdscale stands for classical multi-dimensional matrix.

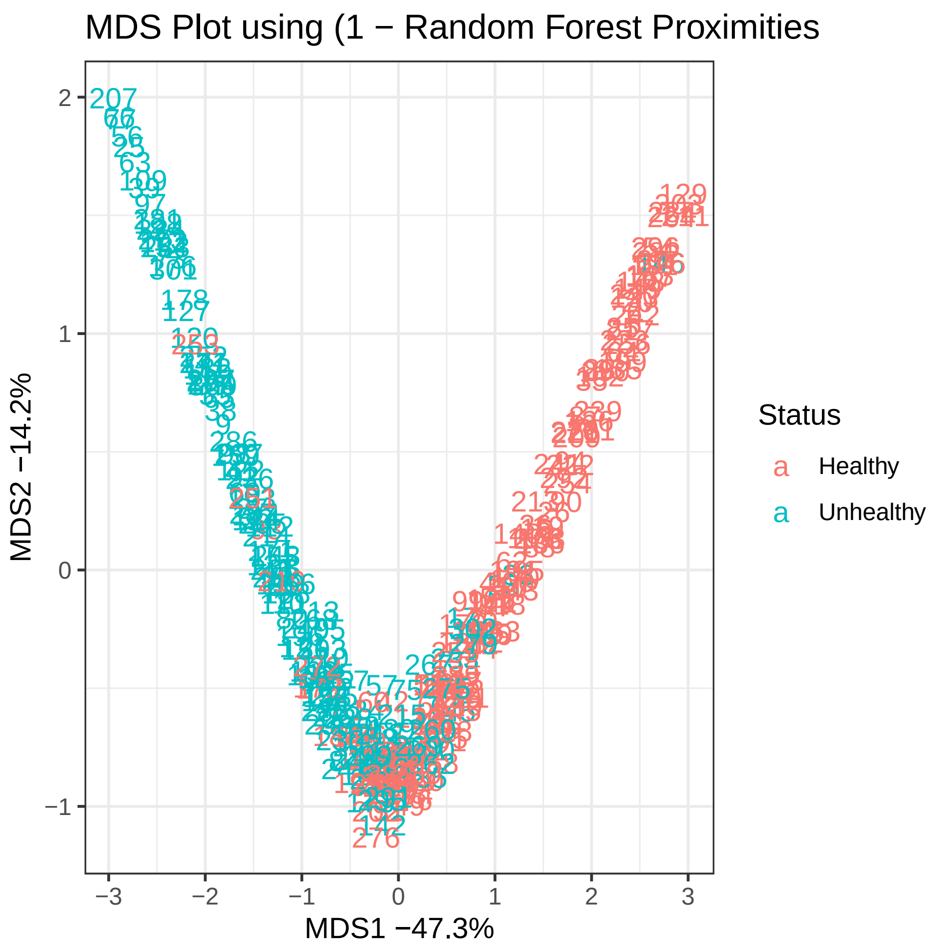
Then we calculate the percentage of variation in the distance matrix that the X and Y axes account for.

```
mds.var.per <- round(mds.stuff$eig/sum(mds.stuff$eig)*100, 1)
```

Then we format the data for ggplot() and draw the plot

```
mds.values <- mds.stuff$points
mds.data <- data.frame(Sample = rownames(mds.values),
                       X = mds.values[,1],
                       Y = mds.values[,2],
                       Status = data.imputed$hd)

ggplot(data = mds.data, aes(x = X, y = Y, label = Sample)) +
  geom_text(aes(color = Status)) +
  theme_bw() +
  xlab(paste("MDS1 -", mds.var.per[1], "%", sep = ""))+
  ylab(paste("MDS2 -", mds.var.per[2], "%", sep = ""))+
  ggtitle("MDS Plot using (1 - Random Forest Proximities")
```



Unhealthy samples are on the left side and healthy samples are on the right side.

There is a "253" number in red of the left side which raises the questions whether the patient 253 was misdiagnosed and actually has heart disease?

NOTE: The X-axis accounts for 47 of the variation in the distance matrix. The Y-axis only accounts for 14 of the variation in the distance matrix. This means that the big differences are along with X-axis.

Lastly, if we got a new patient and didn't know if they had heart disease and they clustered down on the left side, then we'd pretty confident that they had heart disease.