Random Forests

Sebastian Heucke

1/30/2019

Contents

Random Forests

This tutorial is an R markdown version of the video https://www.youtube.com/watch?v=6EXPYzbfLCE&list=PLblh5JKOoLUJJpBNfk8_YadPwDTO2SCbx&index=4&t=0s.

As an example a real dataset will be used.

```
library(ggplot2)
library(cowplot)
```

```
##
## ************************************************************
## Note: As of version 1.0.0, cowplot does not change the
##   default ggplot2 theme anymore. To recover the previous
##   behavior, execute:
##   theme_set(theme_cowplot())
## ************************************************************
```

```
library(randomForest)
```

```
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
url <- "http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data

data <- read.csv(url, header=FALSE)

colnames(data) <- c(
```

```
  "age",
  "sex",# 0 = female, 1 = male
  "cp", # chest pain
          # 1 = typical angina,
          # 2 = atypical angina,
          # 3 = non-anginal pain,
          # 4 = asymptomatic
  "trestbps", # resting blood pressure (in mm Hg)
  "chol", # serum cholestoral in mg/dl
  "fbs",  # fasting blood sugar greater than 120 mg/dl, 1 = TRUE, 0 = FALSE
  "restecg", # resting electrocardiographic results
          # 1 = normal
          # 2 = having ST-T wave abnormality
          # 3 = showing probable or definite left ventricular hypertrophy
  "thalach", # maximum heart rate achieved
  "exang",   # exercise induced angina, 1 = yes, 0 = no
  "oldpeak", # ST depression induced by exercise relative to rest
  "slope", # the slope of the peak exercise ST segment
          # 1 = upsloping
          # 2 = flat
          # 3 = downsloping
  "ca", # number of major vessels (0-3) colored by fluoroscopy
  "thal", # this is short of thalium heart scan
          # 3 = normal (no cold spots)
          # 6 = fixed defect (cold spots during rest and exercise)
          # 7 = reversible defect (when cold spots only appear during exercise)
  "hd" # (the predicted attribute) - diagnosis of heart disease
          # 0 if less than or equal to 50% diameter narrowing
          # 1 if greater than 50% diameter narrowing
  )

head(data) # now we have data and column names
```

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

```
str(data) # this shows that we need to tell R which columns contain factors
```

```
## '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      : Factor w/ 5 levels "?","0.0","1.0",..: 2 5 4 2 2 2 4 2 3 2 ...
##  $ thal    : Factor w/ 4 levels "?","3.0","6.0",..: 3 2 4 2 2 2 2 2 4 4 ...
##  $ hd      : int  0 2 1 0 0 0 3 0 2 1 ...
```

```
          # it also shows us that there are some missing values. There are "?"s
          # in the dataset.

## First, replace "?"s with NAs.
data[data == "?"] <- NA

## Now add factors for variables that are factors and clean up the factors
## that had missing data...
data[data$sex == 0,]$sex <- "F"
data[data$sex == 1,]$sex <- "M"
data$sex <- as.factor(data$sex)

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)

data$ca <- as.integer(data$ca) # since this column had "?"s in it (which
                              # we have since converted to NAs) R thinks that
                              # the levels for the factor are strings, but
                              # we know they are integers, so we'll first
                              # convert the strings to integiers...
data$ca <- as.factor(data$ca)  # ...then convert the integers to factor levels

data$thal <- as.integer(data$thal) # "thal" also had "?"s in it.
data$thal <- as.factor(data$thal)

## This next line replaces 0 and 1 with "Healthy" and "Unhealthy"
data$hd <- ifelse(test=data$hd == 0, yes="Healthy", no="Unhealthy")
data$hd <- as.factor(data$hd) # Now convert to a factor

str(data) ## this shows that the correct columns are factors and we've replaced
```

```
## '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 "2","3","4","5": 1 4 3 1 1 1 3 1 2 1 ...
##  $ thal    : Factor w/ 3 levels "2","3","4": 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 ...
```

```
  ## "?"s with NAs because "?" no longer appears in the list of factors
  ## for "ca" and "thal"
```

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

```
set.seed(42)
```

Now we impute values for the **NAs** in the dataset with **rfImput()**. The first argument to **rfImoute()** is **hd**~ **.**, and that means we want the **hd** (aka **h**eart **d**isease) column to be predicted by the data in all of the other columns. The data is specified by the data=data argument and then you specify how many random forests **rfImpute()** should build to estimate the missing values.

```
## NOTE: For most machine learning methods, you need to divide the data
## manually into a "training" set and a "test" set. This allows you to train
## the method using the training data, and then test it on data it was not
## originally trained on.
##
## In contrast, Random Forests split the data into "training" and "test" sets
## for you. This is because Random Forests use bootstrapped
## data, and thus, not every sample is used to build every tree. The
## "training" dataset is the bootstrapped data and the "test" dataset is
## the remaining samples. The remaining samples are called
## the "Out-Of-Bag" (OOB) data.

## impute any missing values in the training set using proximities
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%
```

```
## NOTE: iter = the number of iterations to run. Breiman says 4 to 6 iterations
## is usually good enough. With this dataset, when we set iter=6, OOB-error
## bounces around between 17% and 18%. When we set iter=20,
# set.seed(42)
# data.imputed <- rfImpute(hd ~ ., data = data, iter=20)
## we get values a little better and a little worse, so doing more
## iterations doesn't improve the situation.
##
## NOTE: If you really want to micromanage how rfImpute(),
## you can change the number of trees it makes (the default is 300) and the
```

```
## number of variables that it will consider at each step.

## Now we are ready to build a random forest.

## NOTE: If the thing we're trying to predict (in this case it is
## whether or not someone has heart disease) is a continuous number
## (i.e. "weight" or "height"), then by default, randomForest() will set
## "mtry", the number of variables to consider at each step,
## to the total number of variables divided by 3 (rounded down), or to 1
## (if the division results in a value less than 1).
## If the thing we're trying to predict is a "factor" (i.e. either "yes/no"
## or "ranked"), then randomForest() will set mtry to
## the square root of the number of variables (rounded down to the next
## integer value).

## In this example, "hd", the thing we are trying to predict, is a factor and
## there are 13 variables. So by default, randomForest() will set
## mtry = sqrt(13) = 3.6 rounded down = 3
## Also, by default random forest generates 500 trees (NOTE: rfImpute() only
## generates 300 tress by default)
```

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.

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

## RandomForest returns all kinds of things
model # gives us an overview of the call, along with...
```

```
##
## 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
```

```
      # 1) The OOB error rate for the forest with ntree trees.
      #    In this case ntree=500 by default
      # 2) The confusion matrix for the forest with ntree trees.
      #    The confusion matrix is laid out like this:
#
#               Healthy                       Unhealthy
#          ------------------------------------------------------------
# Healthy  | Number of healthy people   | Number of healthy people      |
#          | correctly called "healthy" | incorectly called "unhealthy" |
#          | by the forest.             | by the forest                 |
#          ------------------------------------------------------------
# Unhealthy| Number of unhealthy people | Number of unhealthy peole     |
```

```
#           | incorrectly called          | correctly called "unhealthy"  |
#           | "healthy" by the forest     | by the forest                 |
#           ------------------------------------------------------------

## Now check to see if the random forest is actually big enough...
## Up to a point, the more trees in the forest, the better. You can tell when
## you've made enough when the OOB no longer improves.
```

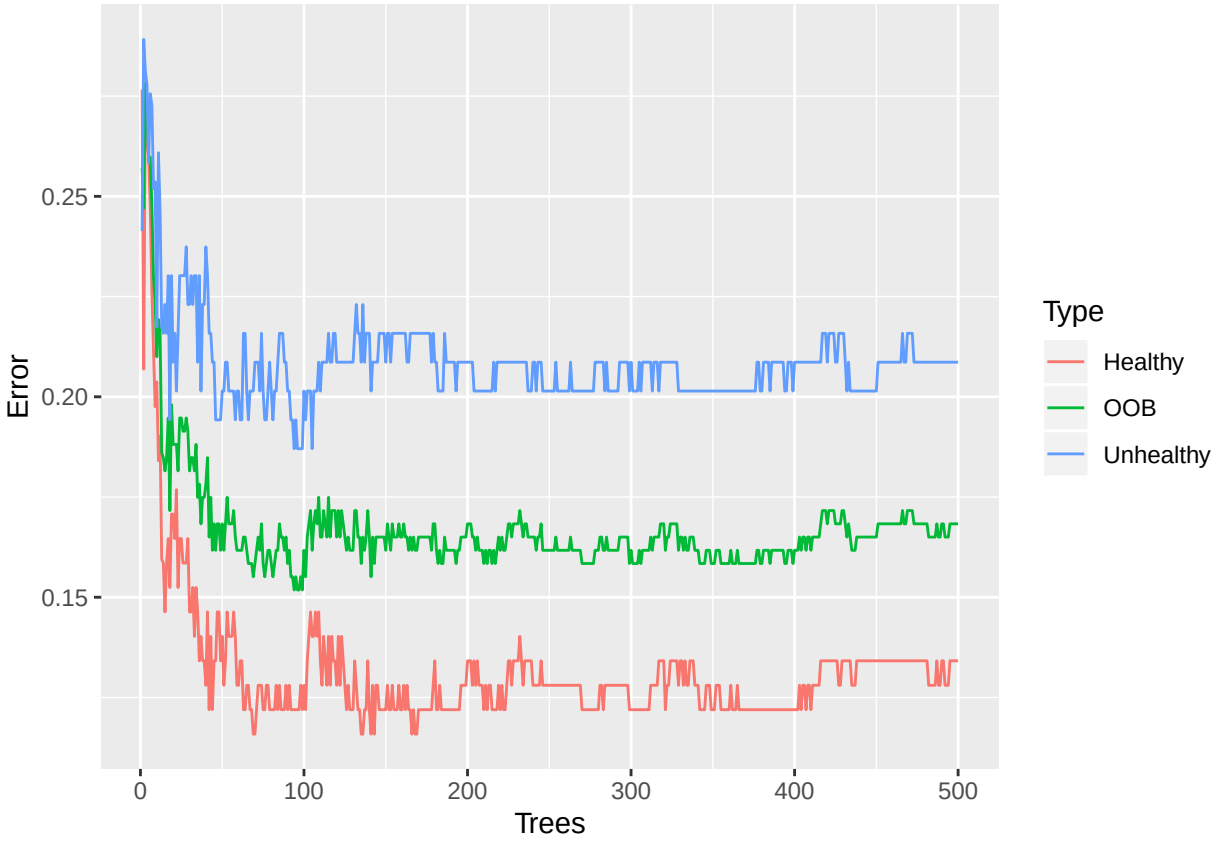
Just like when we imputed values for the **NAs**, we want to predict **hd** (aka **h**eart **d**isease) using all of the other columns in the dataset. However, this time we specify **data.imputed** as the dataset. We also want **randomForest()** to return the proximity matrix. We'll use this to cluster the samples at the end.

The first line in the output is the original call to **randomForest()**. Next we see that the random forest was built to classify samples. Then is tells us how many trees are in the random forest. The default value is 500. Next it tells us how many 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. Then there is the **O**ut-**o**f-**B**ag (**OOB**) error estimate. This means that 83.5% of the OOB samples were correctly classified by the random forest and then you have the confusion matrix. There wee 141 healthy patients that were correctly labeled "Healthy". There were 27 unhealthy patients that were incorrectly classified "Healthy". There were 23 healthy patients that were incorrectly classified "Unhealthy" and lastly, there were 112 unhealthy patients that were correctly classified "Unhealthy".

To see if 500 trees is enough for optimal classification, we can plot the error rates. First you create a dataframe that formats the error rate information so that ggplot2 will be happy. For the most part this is all based on a matrix withing **model** called **err.rate**. What we create is one column for the number of trees, one column for the type of error and one column for the actual error value.

```
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"]))

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



The *blue* line shows the error rate when classifying unhealthy patients. The *green* line shows the overall OOB error rate and the *red* line shows the error rate when classifying healthy patients. In general, we see the error rates decrease when our random forest has more trees.

Now we need to make sure we are considering the optimal number of variables at each internal node in the tree. We 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. Each time we go through the loop, **"i"** increases by 1. It starts at 1 and ends after 10. In every iteration we are building a random forest using "i" to determine the number of variables to try at each step. Specifically, we are setting **mtry=i**, and **"i"** equals values between 1 and 10. The **OOB** error rate after we build each random forest gets stored in a variable.

```
##f we want to compare this random forest to others with different values for
## mtry (to control how many variables are considered 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]
}
oob.values
```

```
##  [1] 0.1716172 0.1716172 0.1584158 0.1815182 0.1749175 0.1881188 0.1881188
##  [8] 0.1914191 0.1947195 0.1947195
```

The **3rd** value, corresponding to **mtry=3**, which is the default in this case, has the lowest OOB error rate. So the default value was optimal, but we wouldn't have known that unless we tried other values.

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. We start by using **dist()** to make a distance matrix from 1-proximity matrix. Then we run **cmdscale()** on the distance matrix. Then we calculate the percentage of variation in the distance matrix

that the X and Y axes account for. Next we format the data for **ggplot()** and draw the data with ggplot.

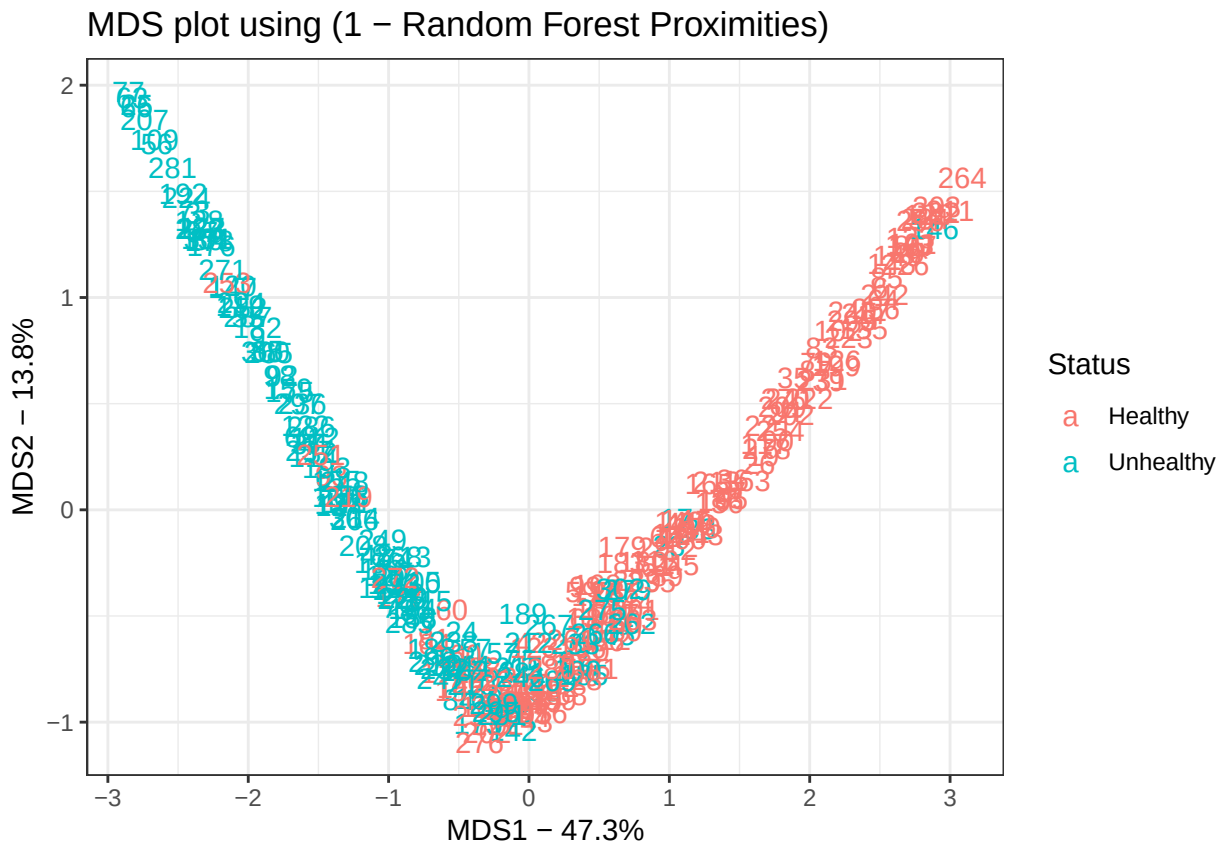
```
## Now let's create an MDS-plot to show how the samples are related to each
## other.
##
## Start by converting the proximity matrix into a distance matrix.
distance.matrix <- dist(1-model$proximity)

mds.stuff <- cmdscale(distance.matrix, eig=TRUE, x.ret=TRUE)

## calculate the percentage of variation that each MDS axis accounts for...
mds.var.per <- round(mds.stuff$eig/sum(mds.stuff$eig)*100, 1)

## now make a fancy looking plot that shows the MDS axes and the variation:
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 one the left side, healthy samples are one the right side.

Session information

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] randomForest_4.6-14 cowplot_1.0.0       ggplot2_3.2.0
## [4] knitr_1.23          devtools_2.1.0      usethis_1.5.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.1        pillar_1.4.2      compiler_3.6.1
##  [4] prettyunits_1.0.2 remotes_2.1.0     tools_3.6.1
##  [7] testthat_2.1.1    digest_0.6.20     pkgbuild_1.0.5
## [10] pkgload_1.0.2     tibble_2.1.3      evaluate_0.14
## [13] memoise_1.1.0     gtable_0.3.0      pkgconfig_2.0.2
## [16] rlang_0.4.0       cli_1.1.0         yaml_2.2.0
## [19] xfun_0.8          dplyr_0.8.3       withr_2.1.2
## [22] stringr_1.4.0     desc_1.2.0        fs_1.3.1
## [25] tidyselect_0.2.5  rprojroot_1.3-2   grid_3.6.1
## [28] glue_1.3.1        R6_2.4.0          processx_3.4.1
## [31] rmarkdown_1.14    sessioninfo_1.1.1 purrr_0.3.2
## [34] callr_3.3.1       magrittr_1.5      backports_1.1.4
## [37] scales_1.0.0      ps_1.3.0          htmltools_0.3.6
## [40] assertthat_0.2.1  colorspace_1.4-1  labeling_0.3
## [43] stringi_1.4.3     lazyeval_0.2.2    munsell_0.5.0
## [46] crayon_1.3.4
```

The document was processed on 2019-10-01.