Random Forest

```r
library(tree)
library(tidyverse)
```

```
## Registered S3 method overwritten by 'cli':
##   method     from
##   print.tree tree

## -- Attaching packages ----------

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts -------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Tree

We only illustrate it via a classification tree. Much of the followings are also true for regression tree.

```r
library(kernlab)  # for the data spam
```

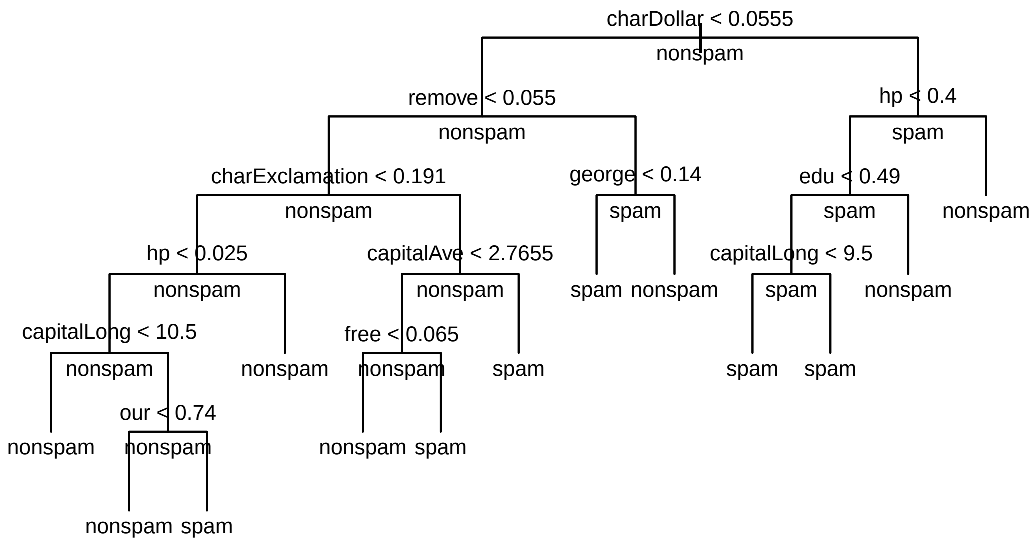
```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:purrr':
##
##     cross

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

```r
data(spam)
```

```r
tree_spam <- tree(type ~ ., spam)
plot(tree_spam, type = "uniform")
text(tree_spam, pretty = 1, all = TRUE, cex = 0.7)
```



Random forest

In R, there are packages like `randomForest` and `ranger` to perform random forest. However, we want to implement it from scratch.

The main idea bebind random forest is to resample the dataset by sampling the row with replacement and selecting the columns randomly.

For the `spam` data, suppose we want to predict the class of a random observation. (We randomly draw an observation from the original data and pretend that we do not know its class)

```r
set.seed(141)
new_data <- spam %>% sample_n(1)
```

With the whole tree, we could do

```r
predict(tree_spam, new_data)
```

```
##    nonspam     spam
## 1 0.812709 0.187291
```

that gives the probabiliy of nonspam about 0.81.

However it is known that a single tree is not predictive. We want to use bootstrap to increase the predictivity.

```r
r <- 1000 # in practise, we need a larger value, say 10000
m <- 8
n <- nrow(spam)
all_col_names <- names(spam)[1:57]  # skip "type"

probs <- map_dbl(seq_len(r), function(i) {
  col_names <- c("type", sample(all_col_names, m))
  spam_boot <- spam[sample(n, n, replace = TRUE), col_names]
  tree_spam_boot <- tree(type ~ ., spam_boot)
  # we only need the probabliy of spam, because the sum of the two values is always 1
  predict(tree_spam_boot, new_data)[2]
})
```

There are two ways to yield the final predicted class, either by consensus or by averaging probablies. Either way, we need a baseline to compare with - using the prior proportion as the baseline is a simplest way (though may not be the best way). One may also use CV to select the baseline.

```r
(baseline <- mean(spam$type == "spam"))
```

```
## [1] 0.3940448
```

Consensus

```r
mean(probs > baseline)
```

```
## [1] 0.513
```

Since more than 50% of the trees predicted `spam`, by consensus, the predicted class for the new data is spam.

By averaging probablies

```r
mean(probs)
```

```
## [1] 0.4475824
```

The average probabliy acress all trees is 0.45 > baseline so the predicted class is "spam". For this new data, we have the same prediction using average probability.

In general, it is more stable to use average probability rather than consensus.

Confidence interval

To construst a CI, you may be thinking of

```r
quantile(probs, c(0.025, 0.975))
```

```
##       2.5%      97.5%
## 0.07925649 0.93187178
```

This confidence interval is essentially the bootstrap percentile interval for a tree model which randomly selects m predictors. It is not very reliable because the sampling of the columns introduces extra variability.

A correct way is to make use of the Jackknife, see https://arxiv.org/pdf/1311.4555.pdf

Well, it is too hard!? Use the package `ranger`!