Solution 1: Bagging

Let's take a look at the average loss of individual base learner predictions. Since we are in the (theoretical) case of an infinitely large ensemble, the average contains an infinite sum, which can equally be viewed as the expectation over $\mathcal{M}$:

$$\begin{aligned}\mathbb{E}_{\mathcal{M}}\left(\left(y - b^{[m]}(\mathbf{x})\right)^2\right) &= \mathbb{E}_{\mathcal{M}}\left(y^2 - 2yb^{[m]}(\mathbf{x}) + \left(b^{[m]}(\mathbf{x})\right)^2\right) \\ &= y^2 - 2y\mathbb{E}_{\mathcal{M}}\left(b^{[m]}(\mathbf{x})\right) + \mathbb{E}_{\mathcal{M}}\left(\left(b^{[m]}(\mathbf{x})\right)^2\right),\end{aligned}$$

where we use the linearity of the expectation.

Note that the average base learner prediction is simply the prediction of the ensemble: $\mathbb{E}_{\mathcal{M}}(b^{[m]}(\mathbf{x})) = f^{[M]}(\mathbf{x})$. Plugging this into the above equation and using the definition of the variance of a random variable Z ("Verschiebungssatz"[1]), which tells us that $\mathbb{E}(Z^2) \geq (\mathbb{E}(Z))^2$, we obtain:

$$\begin{aligned}\mathbb{E}_{\mathcal{M}}\left(\left(y - b^{[m]}(\mathbf{x})\right)^2\right) &\geq y^2 - 2y\mathbb{E}_{\mathcal{M}}\left(b^{[m]}(\mathbf{x})\right) + \left(\mathbb{E}_{\mathcal{M}}\left(b^{[m]}(\mathbf{x})\right)\right)^2 \\ &= \left(y - \mathbb{E}_{\mathcal{M}}\left(b^{[m]}(\mathbf{x})\right)\right)^2 \\ &= \left(y - f^{[M]}(\mathbf{x})\right)^2.\end{aligned}$$

The ensemble loss is thus less than or equal to the average loss of individual base learners. How "sharp" this inequality is depends on how unequal both sides of

$$\mathbb{E}_{\mathcal{M}}\left(\left(b^{[m]}(\mathbf{x})\right)^2\right) \geq \left(\mathbb{E}_{\mathcal{M}}\left(b^{[m]}(\mathbf{x})\right)\right)^2$$

are, determined via the definition of variance by $\mathsf{Var}\left(b^{[m]}(\mathbf{x})\right)$. In other words: the more instable the base learners (high variance), the more beneficial the ensembling procedure.

Solution 2: Classifying `spam`

a) The `spam` data is a binary classification task where the aim is to classify an e-mail as spam or non-spam.

```
library(mlr3)
tsk("spam")

## <TaskClassif:spam> (4601 x 58): HP Spam Detection
## * Target: type
## * Properties: twoclass
## * Features (57):
##   - dbl (57): address, addresses, all, business, capitalAve,
##     capitalLong, capitalTotal, charDollar, charExclamation, charHash,
##     charRoundbracket, charSemicolon, charSquarebracket, conference,
##     credit, cs, data, direct, edu, email, font, free, george, hp, hpl,
```

[1] $\mathsf{Var}(Z) = \mathbb{E}(Z^2) - (\mathbb{E}(Z))^2 \iff \mathbb{E}(Z^2) = \mathsf{Var}(Z) + (\mathbb{E}(Z))^2$, where $\mathsf{Var}(Z) \geq 0$ by definition.

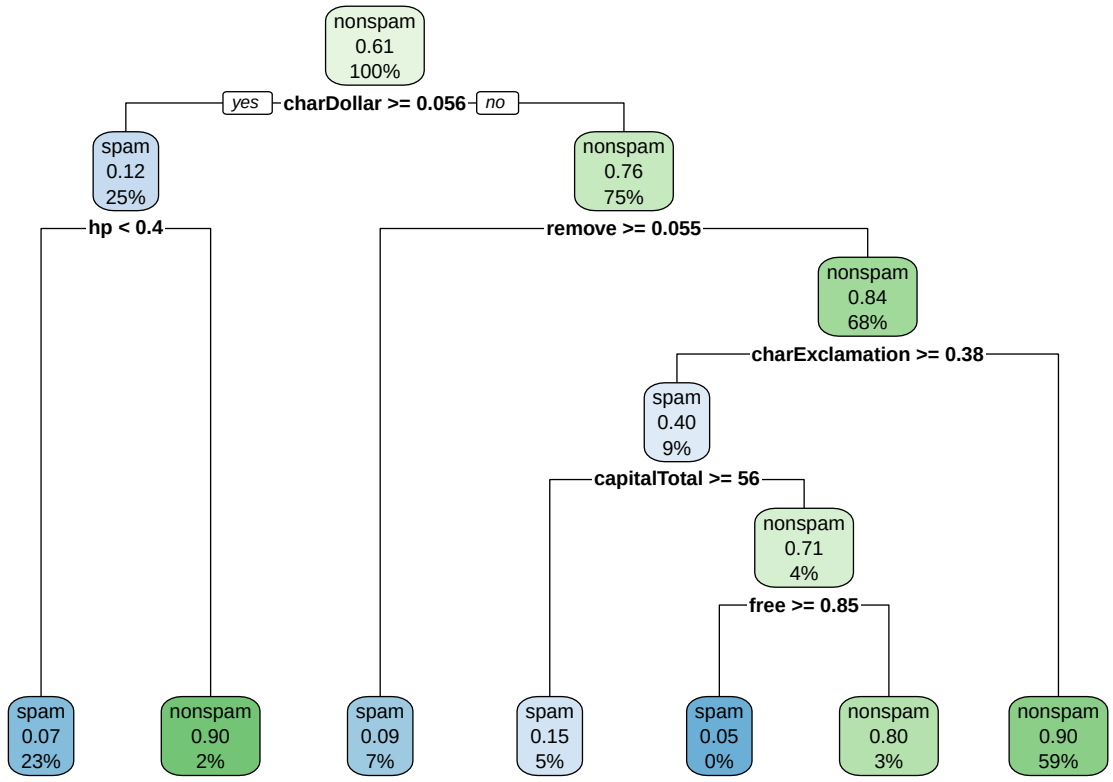
```
##     internet, lab, labs, mail, make, meeting, money, num000, num1999,
##     num3d, num415, num650, num85, num857, order, original, our, over,
##     parts, people, pm, project, re, receive, remove, report, table,
##     technology, telnet, will, you, your
```

b)

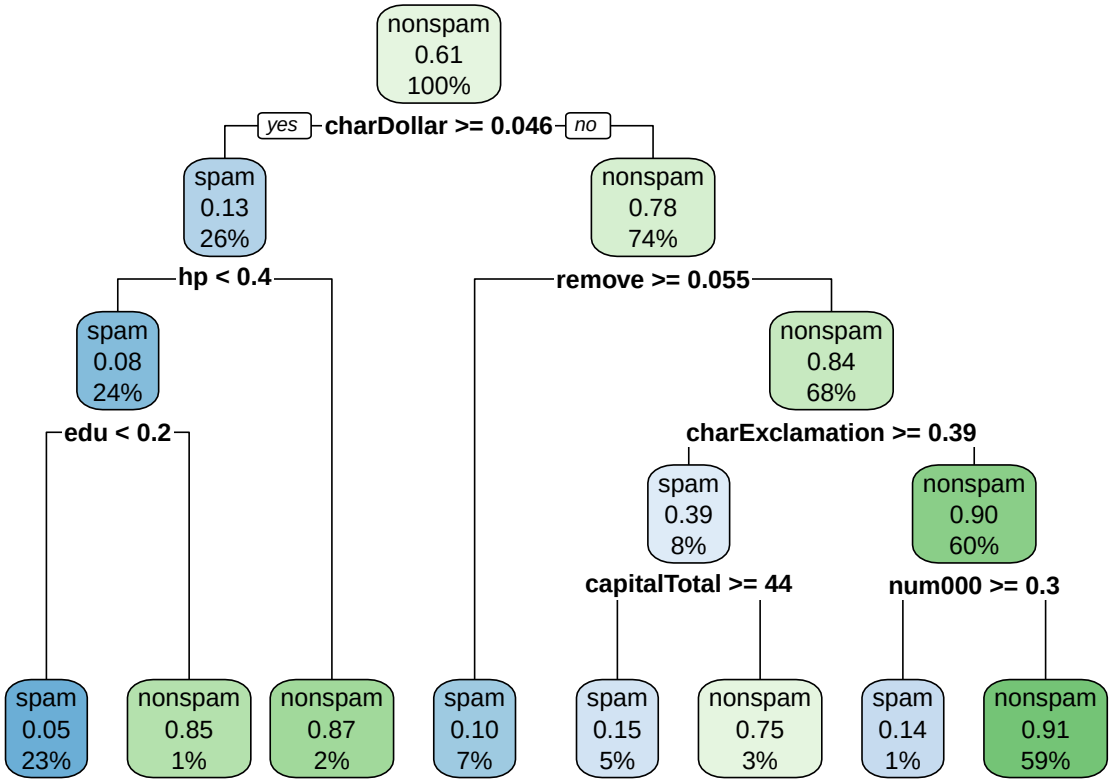
```
library(rpart.plot)
```

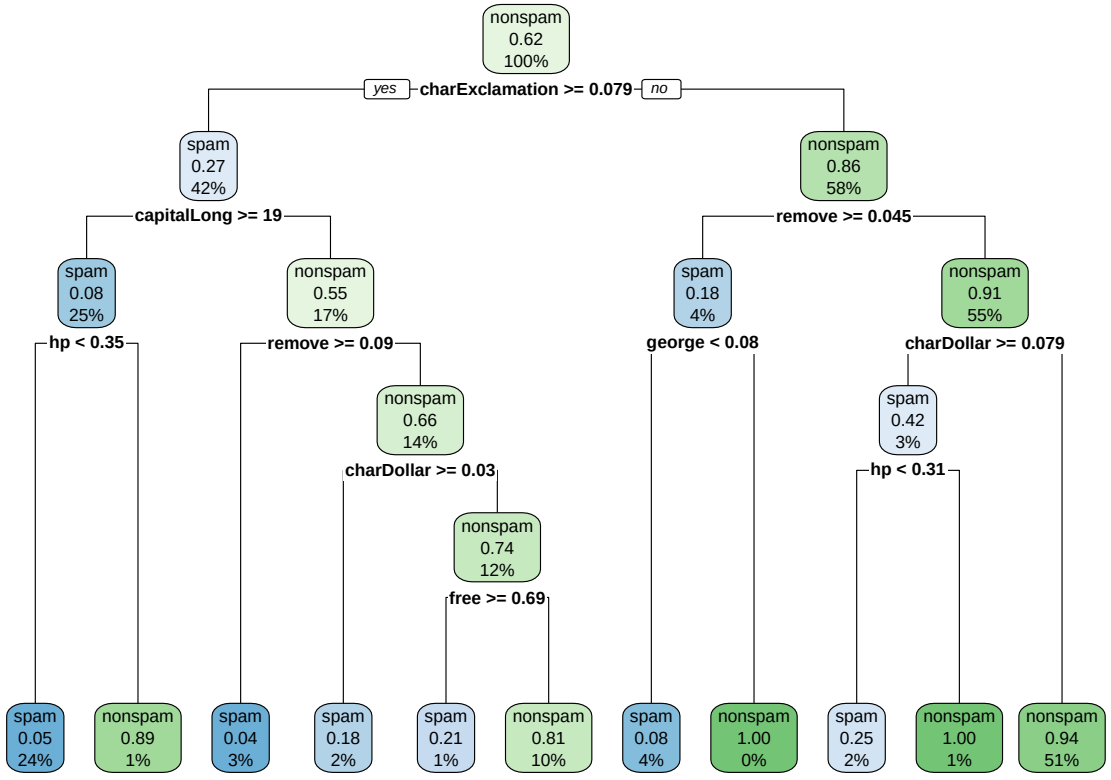
```
## Loading required package:  rpart
```

```
task_spam <- tsk("spam")

learner <- lrn("classif.rpart")
learner$train(task_spam)

set.seed(123)
rpart.plot(learner$model, roundint = FALSE)
```




```
set.seed(456)
subset_1 <- sample.int(task_spam$nrow, size = 0.6 * task_spam$nrow)
set.seed(789)
subset_2 <- sample.int(task_spam$nrow, size = 0.6 * task_spam$nrow)

for (i in list(subset_1, subset_2)) {
  learner$train(task_spam, row_ids = i)
  rpart.plot(learner$model, roundint = FALSE)
}
```





Observation: trees trained on different samples differ considerably in their structure, regarding split variables as well as thresholds (recall, though, that the split candidates are a further source of randomness).

c) i) This is actually quite easy when we recall that the exponential function at an arbitrary input x can be characterized via

$$e^x = \lim_{n\to\infty} \left(1 + \frac{x}{n}\right)^n,$$

which already resembles the limit expression we are looking for. Setting x to -1 yields:

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = e^{-1} = \frac{1}{e}.$$

ii)
```
library(mlr3learners)

learner <- lrn("classif.ranger", "oob.error" = TRUE)
learner$train(tsk("spam"))
learner$model$prediction.error

## [1] 0.04542491
```

d) Variable importance in general measures the contributions of features to a model. One way of computing the variable importance of the j-th variable is based on permuting it for the OOB observations and calculating the mean increase in OOB error this permutation entails.

In order to determine the with the biggest influence on prediction quality, we can choose the k variables with the highest importance score, e.g., for $k = 5$:

```
library(mlr3filters)

learner <- lrn("classif.ranger", importance = "permutation", "oob.error" = TRUE)
filter <- flt("importance", learner = learner)
filter$calculate(tsk("spam"))
head(as.data.table(filter), 5)

##            feature      score
## 1:     capitalLong 0.04523183
## 2:              hp 0.04099699
## 3: charExclamation 0.04018370
## 4:          remove 0.03975776
## 5:      capitalAve 0.03412908
```

Solution 3: Proximities

a) Using the `treeInfo()` output, we can follow the path of each sample through each tree.

The following table prints for each observation (rows) their terminal nodes as assigned by trees 1-3. For example, consider observation 1 in tree 1 (first cell): the observation has `phenols` > 1.94, putting it in node 2 (`rightChild` of node 0), from there in node 6 (because it has `alcohol` > 13.04).

```
# end node each observation is placed in across trees
end_nodes

##    tree_1 tree_2 tree_3
## 1:      6      6      6
## 2:      6      5      5
## 3:      6      6      6
```

b) For the proximities, we consider each pair of observations and compute the relative frequency of trees assigning them to the same terminal node.

- Observations 1 and 2: only tree 1 assigns them to the same node, so the proximity is $\frac{1}{3}$.
- Observations 1 and 3: all trees assign them to the same node, so the proximity is 1.
- Observations 2 and 3: only tree 1 assigns them to the same node, so the proximity is $\frac{1}{3}$.

c) We can put this information into a similarity matrix (as such matrices become large quite quickly for more data, it is common to store only the lower diagonal – the rest is non-informative/redundant):

```
library(proxy)
compute_prox <- function(i, j) sum(i == j) / length(i)
round(proxy::dist(end_nodes, method = compute_prox), 2L)

##      1    2
## 2 0.33
## 3 1.00 0.33
```