Exercise 9 – Random Forests

Introduction to Machine Learning

Hint: Useful libraries

R

```r
# Consider the following libraries for this exercise sheet:

library(proxy)
library(mlr3)
library(rpart.plot)
library(mlr3learners)
library(data.table)
library(mlr3verse)
```

Python

```python
# Consider the following libraries for this exercise sheet:

# general
import numpy as np
import pandas as pd
from scipy.spatial.distance import pdist
from scipy.sparse import dok_matrix
# plots
import matplotlib.pyplot as plt
# sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split
```

Exercise 1: Bagging

Only for lecture group A

Learning goals

1. Understand benefit of bagging from a mathematical perspective
2. Solve "show that…"-type exercises
3. Handle expectations over random variables

In this exercise, we briefly revisit why bagging is a useful technique to stabilize predictions.

For a fixed observation $(\mathbf{x}, y)$, show that the expected quadratic loss over individual base learner predictions $b^{[m]}(\mathbf{x})$ is larger than or equal to the quadratic loss of the prediction $f^{[M]}(\mathbf{x})$ of a size-M ensemble.

You can consider all hyperparameters of the base learners and the ensemble fixed.

Hint

Use the law of total expectation ("Verschiebungssatz der Varianz": $\mathsf{Var}(Z) = \mathbb{E}(Z^2) - (\mathbb{E}(Z))^2 \iff \mathbb{E}(Z^2) = \mathsf{Var}(Z) + (\mathbb{E}(Z))^2$, where $\mathsf{Var}(Z) \geq 0$ by definition.) stating $\mathbb{E}(Z^2) \geq (\mathbb{E}(Z))^2$ for a random variable Z.

Solution

- Start with the LHS of the inequality: the *expected quadratic loss over individual base learner predictions* $\rightsquigarrow \mathbb{E}\left(\left(y - b^{[m]}(\mathbf{x})\right)^2\right)$
- Get clear about RHS: the *quadratic loss of the prediction of a size-M ensemble* $\rightsquigarrow \left(y - \left(f^{[M]}(\mathbf{x})\right)\right)^2$
- Before we get busy moving from LHS to RHS, let's think about the expectation for a minute.

Distribution of the loss

Expected value of a RV: (possibly infinite) sum over all values the RV could take, weighted by the probability of observing that value.
What even is the RV here? Since the base learner and ensemble structure is fixed for given

data, the only stochastic part is the bootstrap sample in the m-th tree given training data from the data-generating process. We could write this as $\mathbb{E}_{\mathcal{D}_{\text{train}}^{[m]} \mid \mathcal{D}_{\text{train}} \sim \mathbb{P}_{xy}}$. In this exercise we'll omit the subscript for ease of notation.

- Back to our proof, which will make use of the LOTV.
- By the LOTV, we know that $\mathbb{E}(Z^2) \geq (\mathbb{E}(Z))^2$ for some RV Z.
- Applying that to our LHS, we obtain

$$\mathbb{E}\left(\left(y - b^{[m]}(\mathbf{x})\right)^2\right) \geq \left(\mathbb{E}\left(y - b^{[m]}(\mathbf{x})\right)\right)^2.$$

- Expected values of larger terms can often be simplified so the expectation is only over the actually stochastic parts (using *linearity* of expectation), yielding:

$$\mathbb{E}\left(\left(y - b^{[m]}(\mathbf{x})\right)^2\right) \geq \left(y - \mathbb{E}\left(b^{[m]}(\mathbf{x})\right)\right)^2.$$

- The last missing step is to show that $\mathbb{E}\left(b^{[m]}(\mathbf{x})\right) = f^{[M]}(\mathbf{x})$. To compute the expectation for this discrete random variable (we have a finite ensemble), we sum over all possible realizations, weighted by their probability of occurence. This can be further simplified given that all of the M bootstrap samples were drawn with equal probability:

$$\mathbb{E}\left(b^{[m]}(\mathbf{x})\right) = \sum_{m=1}^{M} b^{[m]}(\mathbf{x}) p\left(\mathcal{D}_{\text{train}}^{[m]}\right) = \frac{1}{M}\sum_{m=1}^{M} b^{[m]}(\mathbf{x}),$$

which is precisely the ensemble prediction $f^{[M]}(\mathbf{x})$.

Putting everything together, we get

$$\mathbb{E}\left(\left(y - b^{[m]}(\mathbf{x})\right)^2\right) \geq \left(y - \left(f^{[M]}(\mathbf{x})\right)\right)^2,$$

showing that the expected quadratic loss over individual base learner predictions is at least as large as the loss of the ensemble prediction.

Exercise 2: Classifying spam

Learning goals

1) Apply RF to data for prediction, OOB error estimation & feature importance computation
2) Understand how 63% probability for observations to end up in a tree comes about

Only for lecture group B

Take a look at the `spam` dataset and shortly describe what kind of classification problem this is. [only for lecture group B]

Hint

R

Access the corresponding task `?mlr3::mlr_tasks_spam`.

Python

Read spam.csv.

Solution

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

R

```r
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,
    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
```

Python

Load data

```
data_spam = pd.read_csv("../data/spam.csv")
data_spam.drop(data_spam.columns[[0]], axis=1, inplace=True)

X_spam = data_spam.copy() # note without copy() X_spam is not a variable but a pointer
y_spam = X_spam.pop("type")

print(y_spam.value_counts())
```

```
type
nonspam    2788
spam       1813
Name: count, dtype: int64
```

Inspect

```
print(X_spam.describe().head())
```

```
          address    addresses          all     business   capitalAve  \
count  4601.000000  4601.000000  4601.000000  4601.000000  4601.000000   
mean      0.213015     0.049205     0.280656     0.142586     5.191515   
std       1.290575     0.258843     0.504143     0.444055    31.729449   
min       0.000000     0.000000     0.000000     0.000000     1.000000   
25%       0.000000     0.000000     0.000000     0.000000     1.588000   

       capitalLong  capitalTotal   charDollar  charExclamation     charHash  \
count  4601.000000   4601.000000  4601.000000      4601.000000  4601.000000   
mean     52.172789    283.289285     0.075811         0.269071     0.044238   
std     194.891310    606.347851     0.245882         0.815672     0.429342   
min       1.000000      1.000000     0.000000         0.000000     0.000000   
25%       6.000000     35.000000     0.000000         0.000000     0.000000   

       ...           re      receive       remove       report        table  \
count  ...  4601.000000  4601.000000  4601.000000  4601.000000  4601.000000   
mean   ...     0.301224     0.059824     0.114208     0.058626     0.005444   
std    ...     1.011687     0.201545     0.391441     0.335184     0.076274   
min    ...     0.000000     0.000000     0.000000     0.000000     0.000000   
25%    ...     0.000000     0.000000     0.000000     0.000000     0.000000   
```

```
       technology       telnet         will          you         your
count  4601.000000  4601.000000  4601.000000  4601.000000  4601.000000
mean      0.097477     0.064753     0.541702     1.662100     0.809761
std       0.402623     0.403393     0.861698     1.775481     1.200810
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000     0.000000

[5 rows x 57 columns]
```

Only for lecture group B

Use a decision tree to predict `spam`. Re-fit the tree using two random subsets of the data (each comprising 60% of observations). How stable are the trees?

Hint

R

Use `rpart.plot()` from the package `rpart.plot` to visualize the trees.

Python

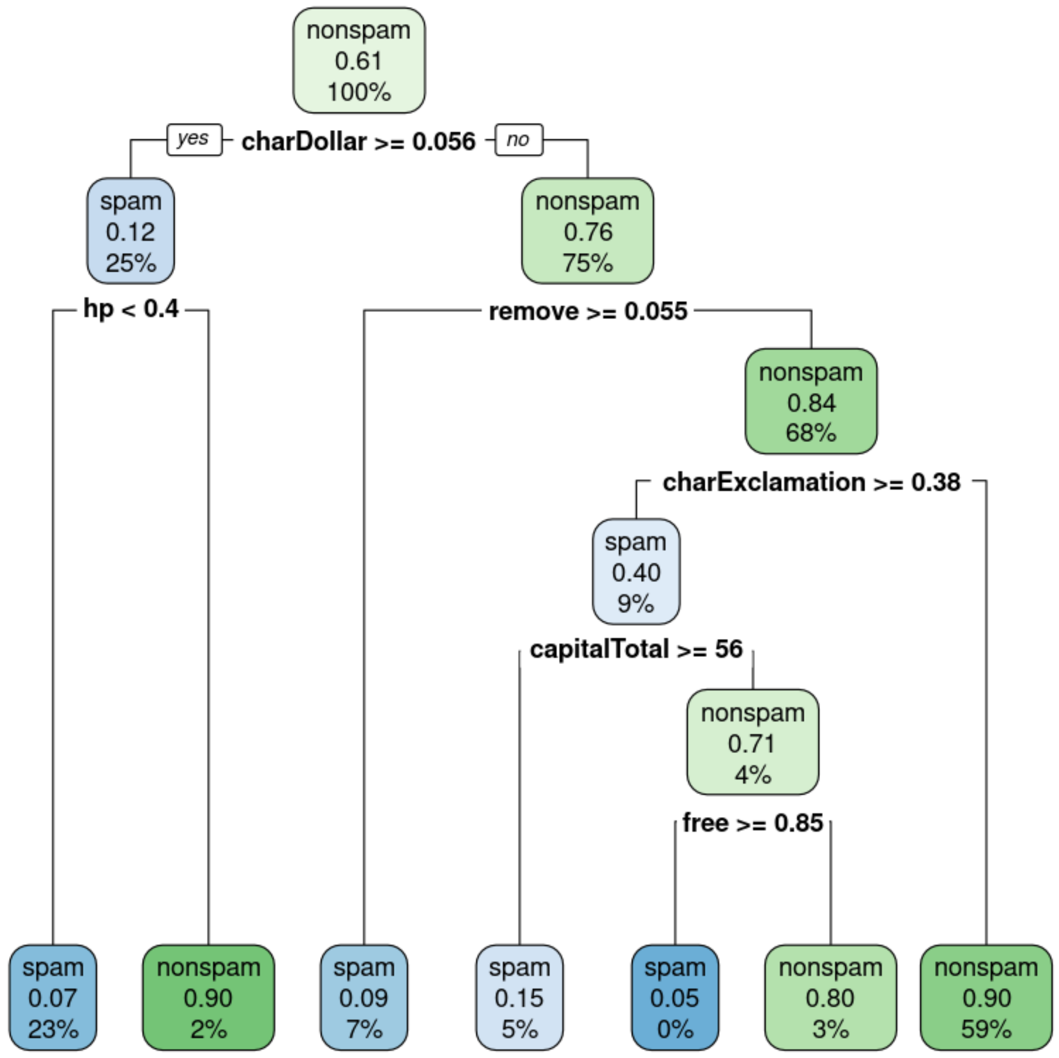
Use `from sklearn.tree import plot_tree` to visualize the trees.

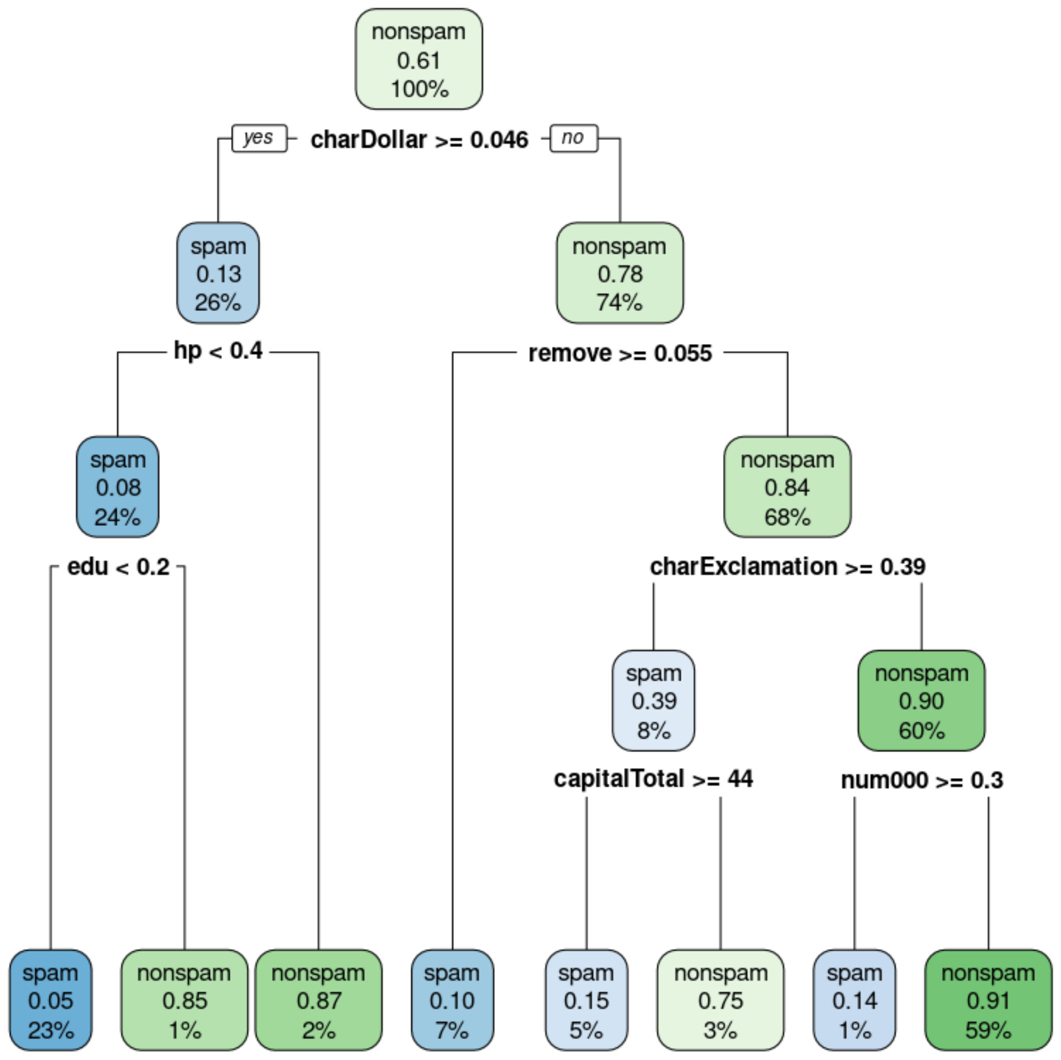
Solution

R

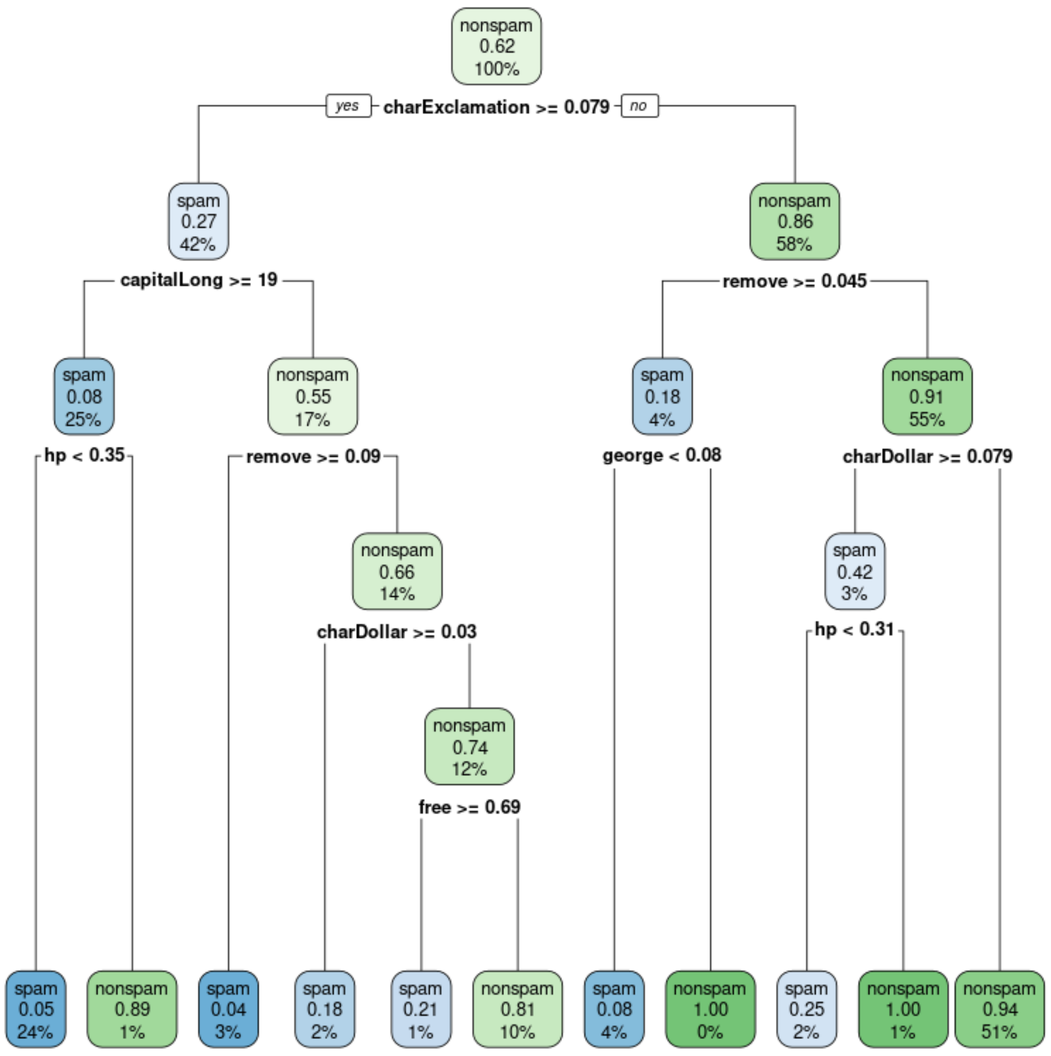
```r
task_spam <- tsk("spam")

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

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

set.seed(456)
```

```r
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)
}
```





nonspam 0.61 100%
yes
charDollar >= 0.046
no
spam 0.13 26%
nonspam 0.78 74%
hp < 0.4
remove >= 0.055
spam 0.08 24%
nonspam 0.84 68%
edu < 0.2
charExclamation >= 0.39
spam 0.39 8%
nonspam 0.90 60%
capitalTotal >= 44
num000 >= 0.3
spam 0.05 23%
nonspam 0.85 1%
nonspam 0.87 2%
spam 0.10 7%
spam 0.15 5%
nonspam 0.75 3%
spam 0.14 1%
nonspam 0.91 59%

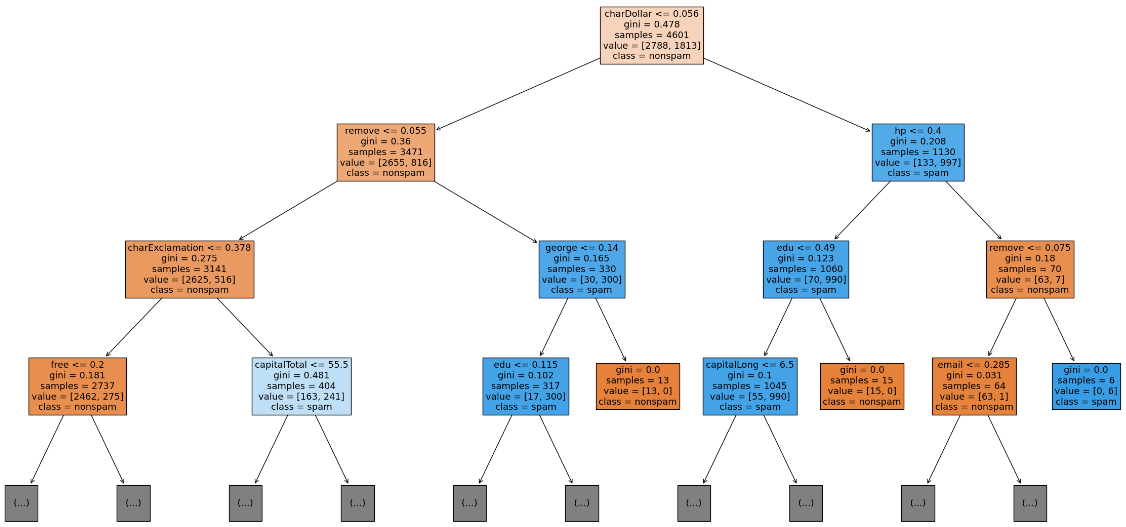


Python

Full dataset

```
# train on full data set
random_state = 43
tree_class_full = DecisionTreeClassifier()
```

```
tree_class_full.fit(X_spam, y_spam)
class_names=data_spam.type.unique().tolist()
class_names.sort()
# use plot_tree to visualize the decision tree
plt.figure(figsize=(30,15))
plot_tree(
    tree_class_full,
    max_depth=3,
    feature_names=X_spam.columns,
    class_names = class_names,
    filled=True,
    fontsize=13
)
plt.show()
```

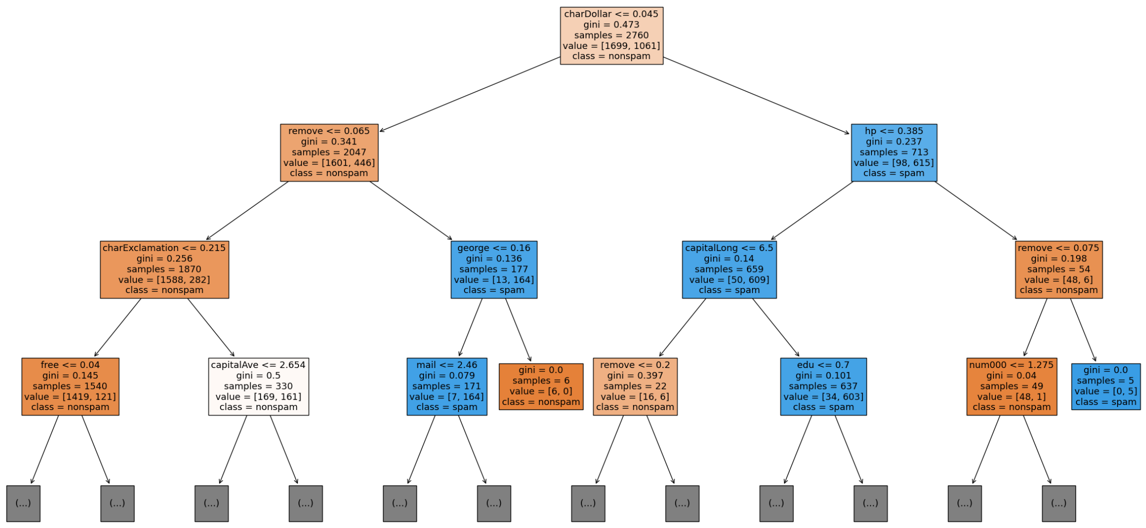


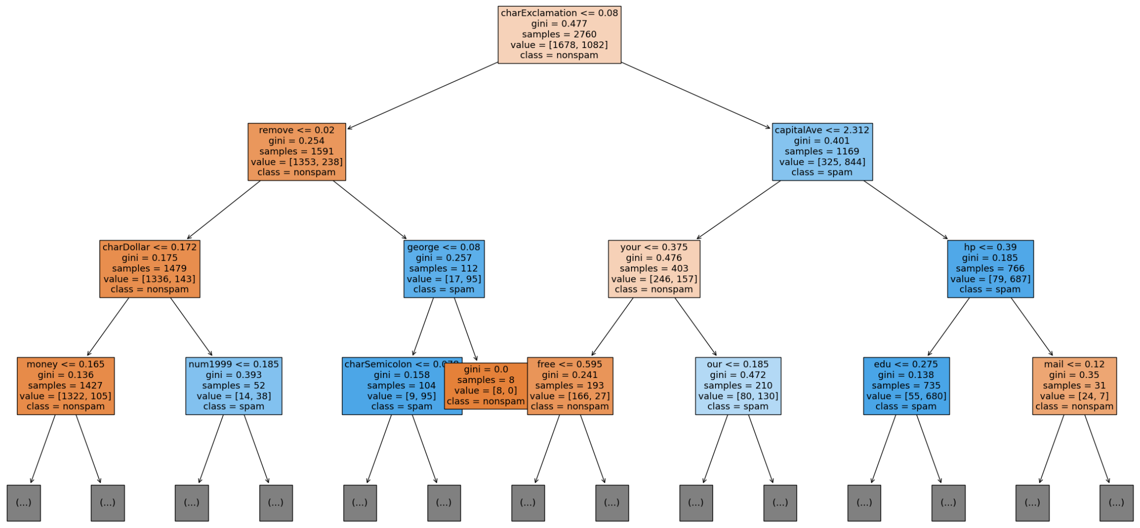
charDollar <= 0.056 gini = 0.478 samples = 4601 value = [2788, 1813] class = nonspam
remove <= 0.055 gini = 0.36 samples = 3471 value = [2655, 816] class = nonspam
hp <= 0.4 gini = 0.208 samples = 1130 value = [133, 997] class = spam
charExclamation <= 0.378 gini = 0.275 samples = 3141 value = [2625, 516] class = nonspam
george <= 0.14 gini = 0.165 samples = 330 value = [30, 300] class = spam
edu <= 0.49 gini = 0.123 samples = 1060 value = [70, 990] class = spam
remove <= 0.075 gini = 0.18 samples = 70 value = [63, 7] class = nonspam
free <= 0.2 gini = 0.181 samples = 2737 value = [2462, 275] class = nonspam
capitalTotal <= 55.5 gini = 0.481 samples = 404 value = [163, 241] class = spam
edu <= 0.115 gini = 0.102 samples = 317 value = [17, 300] class = spam
gini = 0.0 samples = 13 value = [13, 0] class = nonspam
capitalLong <= 6.5 gini = 0.1 samples = 1045 value = [55, 990] class = spam
gini = 0.0 samples = 15 value = [15, 0] class = nonspam
email <= 0.285 gini = 0.031 samples = 64 value = [63, 1] class = nonspam
gini = 0.0 samples = 6 value = [0, 6] class = spam
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

Data subsets

```
# train on random 60% splits

for random_state in [42, 321]:
    X_train, X_test, y_train, y_test = train_test_split(
        X_spam, y_spam, train_size = 0.6, random_state=random_state
    )
```

```
tree_class_sub1 = DecisionTreeClassifier()
tree_class_sub1.fit(X_train, y_train)
# use plot_tree to visualize the decision tree
plt.figure(figsize=(30,15))
plot_tree(
    tree_class_sub1,
    max_depth=3,
    feature_names=X_train.columns,
    class_names = class_names,
    filled=True,
    fontsize=13
)
plt.show()
```




charExclamation <= 0.08 gini = 0.477 samples = 2760 value = [1678, 1082] class = nonspam
remove <= 0.02 gini = 0.254 samples = 1591 value = [1353, 238] class = nonspam
capitalAve <= 2.312 gini = 0.401 samples = 1169 value = [325, 844] class = spam
charDollar <= 0.172 gini = 0.175 samples = 1479 value = [1336, 143] class = nonspam
george <= 0.08 gini = 0.257 samples = 112 value = [17, 95] class = spam
your <= 0.375 gini = 0.476 samples = 403 value = [246, 157] class = nonspam
hp <= 0.39 gini = 0.185 samples = 766 value = [79, 687] class = spam
money <= 0.165 gini = 0.136 samples = 1427 value = [1322, 105] class = nonspam
num1999 <= 0.185 gini = 0.393 samples = 52 value = [14, 38] class = spam
charSemicolon <= 0.0 gini = 0.158 samples = 104 value = [9, 95] class = spam
gini = 0.0 samples = 8 value = [8, 0] class = nonspam
free <= 0.595 gini = 0.241 samples = 193 value = [166, 27] class = nonspam
our <= 0.185 gini = 0.472 samples = 210 value = [80, 130] class = spam
edu <= 0.275 gini = 0.138 samples = 735 value = [55, 680] class = spam
mail <= 0.12 gini = 0.35 samples = 31 value = [24, 7] class = nonspam
(...)

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).

Forests come with a built-in estimate of their generalization ability via the out-of-bag (OOB) error.

i. Show that the probability for an observation to be OOB in an arbitrary bootstrap sample converges to $\frac{1}{e}$.
ii. Use the random forest learner (R: `classif.ranger`, Python: `RandomForestClassifier()` to fit the model and state the out-of-bag (OOB) error.

Solution

i. This requires a little trick.

- First, think about the probability of an observation to be OOB in a tree. Imagine the tree's bootstrap sample has n free spots, to be filled from the training observations.
- In each place, the probability of being drawn for the observation is $\frac{1}{n}$ (all observations are equally likely to be selected). Conversely, the probability of *not* being drawn is $1 - \frac{1}{n}$.
- We draw with replacement, meaning the events of filling a place in the bootstrap sample are all independent. The probability of not being drawn at all for any of the free spots – i.e., not ending up in the tree's bootstrap sample and thus being OOB – is thus $\left(1 - \frac{1}{n}\right)^n$.

- You can imagine that this probability is lower if we only have a few observations. It will converge to a fixed value for larger datasets.
- Since we're interested in a general statement, we look for this stable value, taking n to the limit: $\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n$.
- Now comes the trick: If you're well-versed in analysis you might recognize this expression as a way to characterize the exponential function.
- For an arbitrary input x, we have $e^x = \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n$.
- We see that our above probability is equivalent to the exponential function at input value -1, resulting in

$$\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = e^{-1} = \frac{1}{e} \approx 0.37.$$

ii. The OOB error can be computed by:

R

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

0.0454249076287764

Python

```
crf_full = RandomForestClassifier(random_state=43, oob_score = True)
crf_full.fit(X_spam, y_spam)

print("OOB-error: ", 1 - crf_full.oob_score_)
```

```
OOB-error:  0.04564225168441638
```

You are interested in which variables have the greatest influence on the prediction quality. Explain how to determine this in a permutation-based approach and compute the importance scorses for the `spam` data.

Hint

R

Use an adequate variable importance filter as described here.

Python

Choose an adequate importance measure as described here.

Solution

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 $k = 5$:

R

```r
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)
```

A data.table: 5 × 2

feature <chr>	score <dbl>
capitalLong	0.04523183
hp	0.04099699
charExclamation	0.04018370
remove	0.03975776
capitalAve	0.03412908

Python

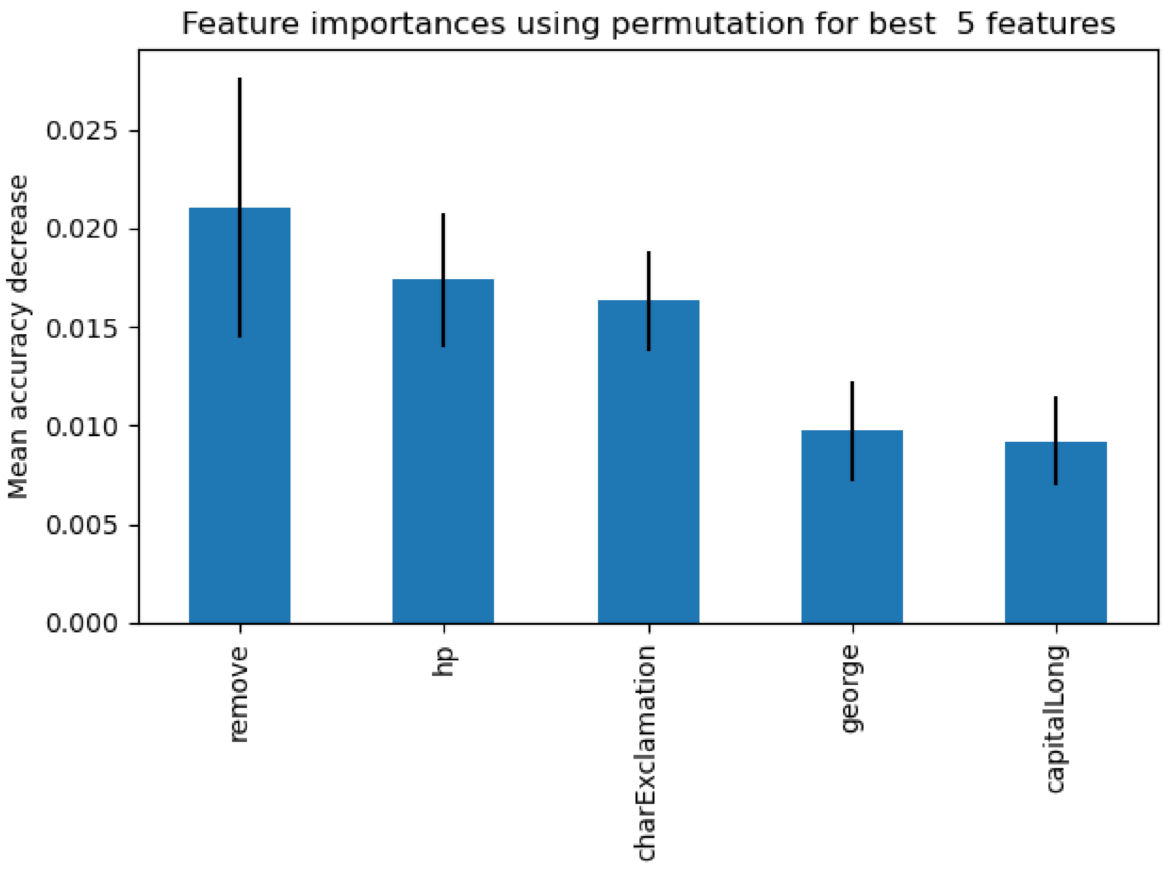
Numerical scores

```
random_state = 321
k = 5
# create a hold-out set and fit another Random Forest Classifier
X_train, X_test, y_train, y_test = train_test_split(
    X_spam, y_spam, test_size=0.25,
    random_state=random_state
)
crf_perm = RandomForestClassifier(random_state=random_state)
crf_perm.fit(X_train, y_train)

result = permutation_importance(
    crf_perm, X_test, y_test, random_state=random_state
)
forest_importances = pd.Series(result.importances_mean, index=X_test.columns)
sorted_idx = forest_importances.argsort()
sorted_idx = sorted_idx[::-1] #reverse order
sorted_idx_selected = sorted_idx[:k]

print(forest_importances[sorted_idx_selected])
```

```
remove              0.021025
hp                  0.017376
charExclamation     0.016334
george              0.009731
capitalLong         0.009209
dtype: float64
```

Visual representation

```
fig, ax = plt.subplots()
forest_importances[sorted_idx_selected].plot.bar(
    yerr=result.importances_std[sorted_idx_selected], ax=ax
)
ax.set_title("Feature importances using permutation for best %2d features" % k)
ax.set_ylabel("Mean accuracy decrease")
fig.tight_layout()
plt.show()
```



Exercise 3: Proximities

Learning goals

1) Be able to make predictions from code output for RF
2) Compute proximities

You solve the `wine` task, predicting the `type` of a wine – with 3 classes – from a number of covariates. After training, you wish to determine how similar your observations are in terms of proximities.

The model information was created with `ranger::treeInfo()`, which assigns observations with values larger than `splitval` to the right child node in each split.

observation	alcalinity	alcohol	flavanoids	hue	malic	phenols
1	11.4	14.75	3.69	1.25	1.73	3.10

observation	alcalinity	alcohol	flavanoids	hue	malic	phenols
2	25.0	13.40	0.96	0.67	4.60	1.98
3	17.4	13.94	3.54	1.12	1.73	2.88

```
[1] "Tree 1:"
```

nodeID	leftChild	rightChild	splitvarID	splitvarName	splitval	terminal	prediction
0	1	2	5	phenols	1.94	FALSE	NA
1	3	4	1	alcohol	12.43	FALSE	NA
2	5	6	1	alcohol	13.04	FALSE	NA
3	NA	NA	NA	NA	NA	TRUE	2
4	NA	NA	NA	NA	NA	TRUE	3
5	NA	NA	NA	NA	NA	TRUE	2
6	NA	NA	NA	NA	NA	TRUE	1

```
[1] "Tree 2:"
```

nodeID	leftChild	rightChild	splitvarID	splitvarName	splitval	terminal	prediction
0	1	2	1	alcohol	12.78	FALSE	NA
1	3	4	3	hue	0.68	FALSE	NA
2	5	6	2	flavanoids	2.18	FALSE	NA
3	NA	NA	NA	NA	NA	TRUE	3
4	NA	NA	NA	NA	NA	TRUE	2
5	NA	NA	NA	NA	NA	TRUE	3
6	NA	NA	NA	NA	NA	TRUE	1

```
[1] "Tree 3:"
```

nodeID	leftChild	rightChild	splitvarID	splitvarName	splitval	terminal	prediction
0	1	2	1	alcohol	12.79	FALSE	NA
1	3	4	5	phenols	2.01	FALSE	NA
2	5	6	5	phenols	2.28	FALSE	NA
3	NA	NA	NA	NA	NA	TRUE	2
4	NA	NA	NA	NA	NA	TRUE	2
5	NA	NA	NA	NA	NA	TRUE	3
6	NA	NA	NA	NA	NA	TRUE	1

nodeID	leftChild	rightChild	splitvarID	splitvarName	splitval	terminal	prediction

For the following subset of the training data and the random forest model given above,

---

find the terminal node of each tree the observations are placed in,

Solution

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).

```
   tree_1  tree_2  tree_3
0       6       6       6
1       6       5       5
2       6       6       6
```

---

compute the observations' pairwise proximities, and

Solution

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 $\frac{1}{3}$.

---

construct a similarity matrix from these proximities in `R` resp. `Python`.

Solution

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):

R

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

```
     1    2    3
1 0.00
2 0.33 0.00
3 1.00 0.33 0.00
```

Python

```python
# Compute the pairwise distances between the rows of X
distances = 1 - pdist(end_nodes, metric='hamming')
# Compute the size of the matrix
n = end_nodes.shape[0]
size = (n * (n - 1)) // 2
# Create a sparse matrix to store the pairwise distances.
# Sparse matrices only need memory space for values unequal to zero.
distance_matrix = dok_matrix((n, n), dtype=np.float32)
# Populate the lower triangle of the matrix with the distances
i, j = np.tril_indices(n, k=-1)
distance_matrix[i, j] = distances[:size]
# Print the matrix
print(distance_matrix.toarray())
```

```
[[0.         0.         0.        ]
 [0.33333334 0.         0.        ]
 [1.         0.33333334 0.        ]]
```