Exercise 8 – CART

Introduction to Machine Learning

Hint: Useful libraries

R

```
library(mlr3verse)
library(rattle)
```

Python

```
import numpy as np
```

Exercise 1: Splitting criteria

Learning goals

1) Perform split computation with pen and paper
2) Derive optimal constant predictor for regression under L2 loss

Consider the following dataset:

i	$x^{(i)}$	$y^{(i)}$
1	1.0	1.0
2	2.0	1.0
3	7.0	0.5
4	10.0	10.0
5	20.0	11.0

Compute the first split point the CART algorithm would find for each data set (with pen and paper or in `R`, resp. `Python`). Use mean squared error (MSE) to assess the empirical risk.

Solution

Pen-and-paper solution

Procedure

1. We have only one split variable x. We probe all binary splits, where thresholds are placed equidistant between the observed feature values (think about why this might help generalization).
2. For each threshold:
   - Map observations to child nodes resulting from split at that threshold
   - Compute optimal constant prediction and resulting loss in both child nodes
   - Aggregate to obtain overall loss of split
3. Choose optimal split w.r.t. empirical risk reduction.

Accounting for node size

Two scenarios in aggregating the risk contributions from both candidate child nodes:

- Child node risk is sum of pointwise losses without accounting for number of observations $\rightsquigarrow$ *simple* average
- Child node risk is average of pointwise losses, thus accounting for number of observations $\rightsquigarrow$ *weighted* average

Split points

Split candidates $t \in \{1.5, 4.5, 8.5, 15\}$, leading to the following child nodes:

$$\mathcal{N}_1(t) = \{(x, y) \in \mathcal{N} : x \leq t\}, \quad \mathcal{N}_2(t) = \{(x, y) \in \mathcal{N} : x > t\}.$$

Risk per split point

Calculate $\mathcal{R}(\mathcal{N}, t)$ for each split point:

- $x \leq 1.5$
  - Predictions: $c_1 = \frac{1}{1}\sum_{i=1}^{1} y_i = 1, \quad c_2 = \frac{1}{4}\sum_{i=2}^{5} y_i = 5.625$
  - Risk left: $\rho_{\text{MSE}}(\mathcal{N}_1, 1.5) = 0$
  - Risk right: $\rho_{\text{MSE}}(\mathcal{N}_2, 1.5) = \frac{1}{4}((1-5.625)^2 + (0.5-5.625)^2 + (10-5.625)^2 + (11-5.625)^2)$
  - Aggregate risk: $\mathcal{R}(\mathcal{N}, 1.5) = \frac{|\mathcal{N}_1|}{|\mathcal{N}|}\rho_{\text{MSE}}(\mathcal{N}_1, 1.5) + \frac{|\mathcal{N}_2|}{|\mathcal{N}|}\rho_{\text{MSE}}(\mathcal{N}_2, 1.5) = \frac{1}{5}\cdot 0 + \frac{4}{5}\cdot 23.925 = 19.14$
- $x \leq 4.5 \rightsquigarrow \mathcal{R}(\mathcal{N}, 4.5) = 13.43$
- $x \leq 8.5 \rightsquigarrow \mathcal{R}(\mathcal{N}, 8.5) = 0.13 \rightsquigarrow$ **optimal**
- $x \leq 15 \rightsquigarrow \mathcal{R}(\mathcal{N}, 15) = 12.64$

R

Write function to perform MSE-based splits:

```r
x <- c(1, 2, 7, 10, 20)
y <- c(1, 1, 0.5, 10, 11)

compute_mse <- function (y) mean((y - mean(y))**2)

compute_total_mse <- function (yleft, yright) {
  num_left <- length(yleft)
  num_right <- length(yright)
  w_mse_left <- num_left / (num_left + num_right) * compute_mse(yleft)
  w_mse_right <- num_right / (num_left + num_right) * compute_mse(yright)
  w_mse_left + w_mse_right
}

perform_split <- function(x, y) {
  # try out all unique points as potential split points and ...
  unique_sorted_x <- sort(unique(x))
  split_points <- head(unique_sorted_x, length(unique_sorted_x) - 1) +
    0.5 * diff(unique_sorted_x)

  node_mses <- lapply(
```

```
    split_points,
    function(i) {
      y_left <- y[x <= i]
      y_right <- y[x > i]
      # ... compute SS in both groups
      mse_split <- compute_total_mse(y_left, y_right)
      print(sprintf("split at %.2f: empirical risk = %.2f", i, mse_split))
      mse_split
    }
  )

  # select the split point yielding the maximum impurity reduction
  split_points[which.min(node_mses)]
}
```

Apply to dataset:

```
perform_split(x, y) # 3rd obs is best split point
```

```
[1] "split at 1.50: empirical risk = 19.14"
[1] "split at 4.50: empirical risk = 13.43"
[1] "split at 8.50: empirical risk = 0.13"
[1] "split at 15.00: empirical risk = 12.64"
```

8.5

Python

Imports

```
import numpy as np
```

Write function to perform MSE-based splits:

```
def find_best_split(x_train,y_train):
    best_threshold = None
    min_risk = np.inf
    unique_sorted_x = np.unique(x_train)
    unique_sorted_x.sort()
    thresholds = (unique_sorted_x[1:] + unique_sorted_x[:-1]) / 2
    mse = lambda y: np.mean((y - y.mean())**2)
```

```
    for t in thresholds:
        y_left, y_right = y_train[x_train <= t], y_train[x_train > t]
        weight_left = len(y_left) / len(y_train)
        t_mse = weight_left * mse(y_left) + (1 - weight_left) * mse(y_right)
        print("split at %.2f: empirical risk = %.2f" % (t,t_mse))
        if t_mse < min_risk: # save best split
            min_risk = t_mse
            best_threshold = t

    print("best split at ", best_threshold)
    return {'threshold': best_threshold, 'empirical_risk': min_risk}
```

Apply to dataset:

```
x = np.array([1, 2, 7, 10, 20])
y = np.array([1, 1, 0.5, 10, 11])
find_best_split(x, y)
```

```
split at 1.50: empirical risk = 19.14
split at 4.50: empirical risk = 13.43
split at 8.50: empirical risk = 0.13
split at 15.00: empirical risk = 12.64
best split at  8.5

{'threshold': 8.5, 'empirical_risk': 0.13333333333333333}
```

Derive the optimal constant predictor for a node $\mathcal{N}$ when minimizing the empirical risk under $L2$ loss and explain why this is equivalent to minimizing variance impurity.

Solution

Constant $L2$ risk minimizer for a node $\mathcal{N}$:

$$\bar{y} = \arg\min_c \frac{1}{|\mathcal{N}|} \sum_{i=1}^{|\mathcal{N}|} (y^{(i)} - c)^2,$$

because

$$
\begin{aligned}
\min_{c\in\mathbb{R}} \frac{1}{|\mathcal{N}|}\sum_{i=1}^{|\mathcal{N}|}(y^{(i)}-c)^2 &\iff \frac{\partial}{\partial c}\left(\frac{1}{|\mathcal{N}|}\sum_{i=1}^{|\mathcal{N}|}(y^{(i)}-c)^2\right) = 0 \\
&\iff \frac{1}{|\mathcal{N}|}\frac{\partial}{\partial c}\left(\sum_{i=1}^{|\mathcal{N}|}\left({y^{(i)}}^2 - 2y^{(i)}c + c^2\right)\right) = 0 \\
&\iff \left(\sum_{i=1}^{|\mathcal{N}|}\left(-2y^{(i)} + 2c\right)\right) = 0 \\
&\iff |\mathcal{N}| \cdot c = \sum_{i=1}^{|\mathcal{N}|} y^{(i)} \\
&\implies \hat{c} = \frac{1}{|\mathcal{N}|}\sum_{i=1}^{|\mathcal{N}|} y^{(i)} = \bar{y}.
\end{aligned}
$$

It's easy to see that the mean prediction minimizes *variance impurity*.

- We just found that
$$\bar{y} = \arg\min_{c\in\mathbb{R}} \frac{1}{|\mathcal{N}|}\sum_{i=1}^{|\mathcal{N}|}(y^{(i)}-c)^2.$$
- Looking closely at this minimization objective reveals: it's simply the (biased) sample variance for sample mean c.
- Therefore, predicting the sample mean minimizes both $L2$ risk and variance impurity.

Thinking a little further

- The mean of a sample is precisely the value for which the sum of squared distances to all points is minimal (the "center" of the data).
- Constant mean prediction is equivalent to an intercept LM (trained with $L2$ loss) $\rightsquigarrow$ regression trees with $L2$ loss perform piecewise constant linear regression.

Explain why performing further splits can never result in a higher overall risk with $L2$ loss as a splitting criterion.

Solution

The variance of a subset of the observations in a node cannot be higher than the variance of the entire node, so it's not possible to make the tree worse (w.r.t. training error) by introducing a further split.

Exercise 2: Splitting criteria

> Learning goals
>
> Understand the effect of CART hyperparameters

In this exercise, we will have a look at two of the most important CART hyperparameters, i.e., design choices exogenous to training. Both `minsplit` and `maxdepth` influence the number of input space partitions the CART will perform.

How do you expect the number of splits to affect the model fit and generalization performance?

Solution

Allowing for more splits will make the model more complex, thus—all else being equal—achieving a better data fit but also increasing the risk of overfitting.

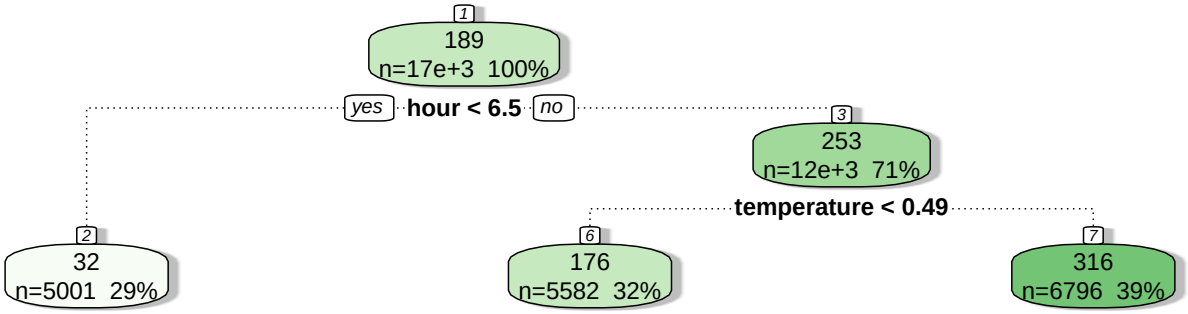
Using `mlr3`, fit a regression tree learner (`regr.rpart`) to the `bike_sharing` task (omitting the `date` feature) for

- `maxdepth` $\in \{2, 4, 8\}$ with `minsplit` $= 2$
- `minsplit` $\in \{5, 1000, 10000\}$ with `maxdepth` $= 20$

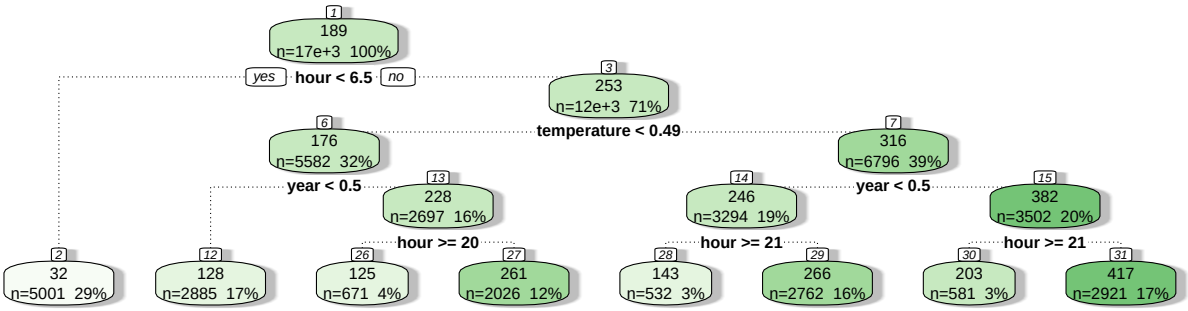
What do you observe?

Solution

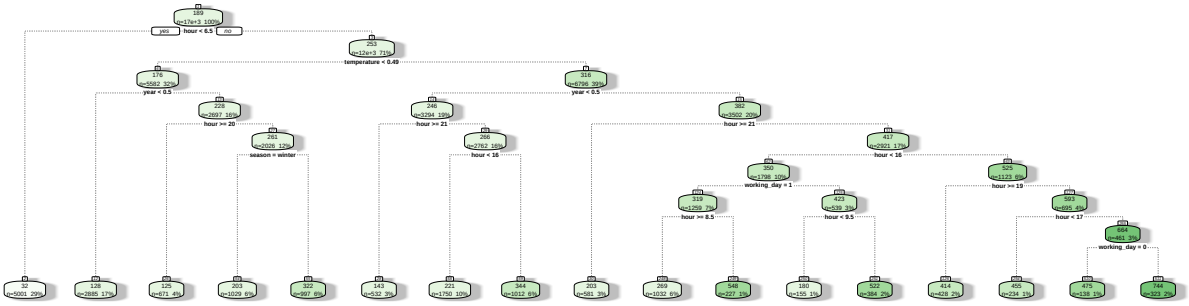
```
library(mlr3verse)
library(rattle)
task <- tsk("bike_sharing")
task$select(task$feature_names[task$feature_names != "date"])

for (i in c(2, 4, 8)) {
  learner <- lrn("regr.rpart", minsplit = 2, maxdepth = i, minbucket = 1)
  learner$train(task)
  fancyRpartPlot(learner$model, caption = sprintf("maxdepth: %i", i))
}
```



maxdepth: 2

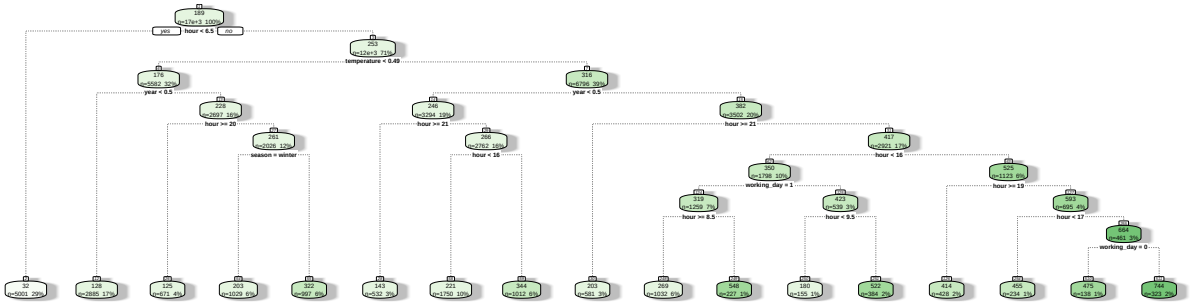


maxdepth: 4

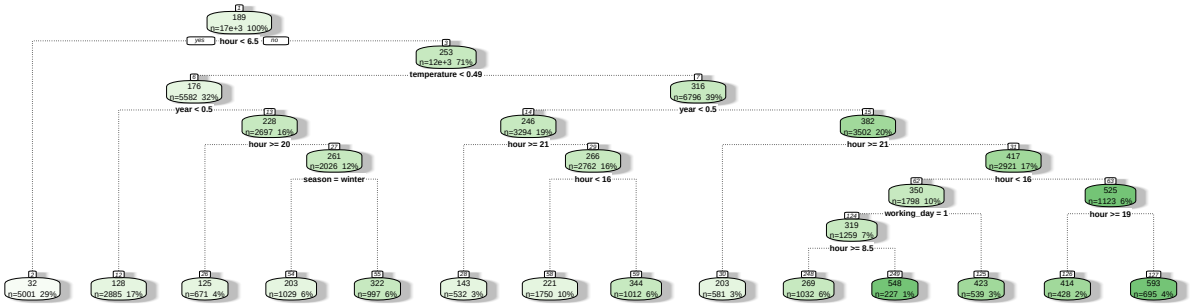


maxdepth: 8

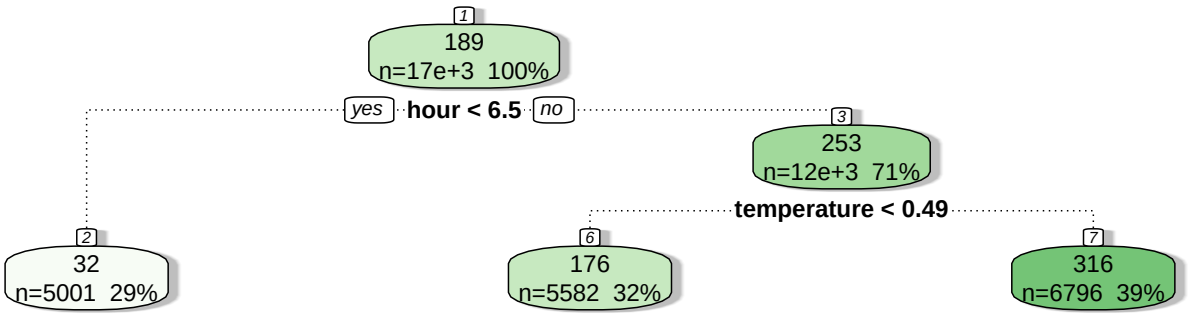
```
for (i in c(5, 1000, 10000)) {
  learner <- lrn("regr.rpart", minsplit = i, maxdepth = 20, minbucket = 1)
  learner$train(task)
  fancyRpartPlot(learner$model, caption = sprintf("minsplit: %i", i))
}
```



minsplit: 5



minsplit: 1000



minsplit: 10000

Higher values of `maxdepth` and lower values of `minsplit`, respectively, produce more complex trees.

Which of the two options should we use to control tree appearance?

Solution

Both hyperparameters can be used to control tree depth, and their effect depends on the properties of the data-generating process (e.g., at least 100 observations to split further can mean very pure or very impure nodes). Sometimes requirements like interpretability might steer our decision (e.g., a tree of depth 15 is probably hard to interpret). Usually, however, we will employ **hyperparameter tuning** to determine the values for both (and other) hyperparameters, deferring the responsibility from the practitioner to the data.

Exercise 3: Impurity reduction

Only for lecture group A

Learning goals

1. Develop intuition for use of Brier score in classification trees
2. Reason about distribution and expectations of random variables
3. Handle computations involving expectations

TLDR;

This exercise is rather involved and requires some knowledge of probability theory.
Main take-away (besides training proof-type questions): *In constructing CART with minimal Gini impurity, we minimize the expected rate of misclassification across the training data.*

We will now build some intuition for the Brier score / Gini impurity as a splitting criterion by showing that it is equal to the expected MCE of the resulting node.

The fractions of the classes $k = 1, \dots, g$ in node $\mathcal{N}$ of a decision tree are $\pi_1^{(\mathcal{N})}, \dots, \pi_g^{(\mathcal{N})}$, where

$$\pi_k^{(\mathcal{N})} = \frac{1}{|\mathcal{N}|} \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{N}} [y^{(i)} = k].$$

For an expression that holds in expectation over arbitrary data, we need to introduce stochasticity. Assume we replace the (deterministic) classification rule in node $\mathcal{N}$

$$\hat{k} \mid \mathcal{N} = \arg\max_k \pi_k^{(\mathcal{N})}$$

by a randomizing rule

$$\hat{k} \sim \text{Cat}\left(\pi_1^{(\mathcal{N})}, \dots, \pi_g^{(\mathcal{N})}\right),$$

in which we draw the classes from the categorical distribution of their estimated probabilities (i.e., class k is predicted with probability $\pi_k^{(\mathcal{N})}$).

Explain the difference between the deterministic and the randomized classification rule.

Solution

The deterministic rule tells us to pick the class with the highest empirical frequency. With the randomizing rule, we draw from a categorical distribution that is parameterized with these same empirical frequencies, meaning we draw the most frequent class with probability $\gamma = \max_k \pi_k^{(\mathcal{N})}$. If we repeat this process many times, we will predict this class γ % of the time. Therefore, in expectation, we will also predict the most frequent class in most cases, but the rule is more nuanced as the magnitude of γ makes a difference (the closer to 1, the more similar both rules).

Using the randomized rule, compute the expected MCE in node $\mathcal{N}$ that contains n random training samples. What do you notice?

Solution

In order to compute the expected MCE, we need some random variables (RV) because we want to make a statement that holds for arbitrary training data drawn from the data-generating process. More precisely, we define $n \in \mathbb{N}$ i.i.d. RV $Y^{(1)}, \dots, Y^{(n)}$ that are distributed according to the categorical distribution induced by the observed class frequencies:

$$\mathbb{P}(Y^{(i)} = k|\mathcal{N}) = \pi_k^{(\mathcal{N})} \quad \forall i \in \{1, \dots, n\}, \quad k \in \mathcal{Y}.$$

The label $y^{(i)}$ of the i-th training observation is thus a realization of the corresponding RV $Y^{(i)}$.

Since our new randomization rule is stochastic, the predictions for the training observations will also be realizations of RV that we denote by $\hat{Y}^{(1)}, ..., \hat{Y}^{(n)}$. By design, they follow the same categorical distribution:

$$\mathbb{P}(\hat{Y}^{(i)} = k|\mathcal{N}) = \pi_k^{(\mathcal{N})} \quad \forall i \in \{1, \dots, n\}, \quad k \in \mathcal{Y}.$$

Then, we can define the MCE for a node with $n = |\mathcal{N}|$ when the randomizing rule is used:

$$\rho_{\text{MCE}}(\mathcal{N}) = \frac{1}{n} \sum_{i=1}^{n} \left[Y^{(i)} \neq \hat{Y}^{(i)} \right].$$

Taking the expectation of this MCE leads to a statement about a node with arbitrary training data:

$$
\begin{aligned}
\mathbb{E}_{Y^{(1)},\ldots,Y^{(n)},\hat{Y}^{(1)},\ldots,\hat{Y}^{(n)}}\left(\rho_{\mathrm{MCE}}(\mathcal{N})\right) &= \mathbb{E}_{Y^{(1)},\ldots,Y^{(n)},\hat{Y}^{(1)},\ldots,\hat{Y}^{(n)}}\left(\frac{1}{n}\sum_{i=1}^{n}\left[Y^{(i)} \neq \hat{Y}^{(i)}\right]\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{Y^{(i)},\hat{Y}^{(i)}}\left(\left[Y^{(i)} \neq \hat{Y}^{(i)}\right]\right) \quad \text{i.i.d. assumption + linearity} \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{Y^{(i)}}\left(\mathbb{E}_{\hat{Y}^{(i)}}\left(\left[Y^{(i)} \neq \hat{Y}^{(i)}\right]\right)\right) \quad \text{Fubini's theorem} \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{Y^{(i)}}\left(\sum_{k\in\mathcal{Y}}\pi_k^{(\mathcal{N})}\cdot\left[Y^{(i)} \neq k\right]\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{Y^{(i)}}\left(1-\pi_{k=Y^{(i)}}^{(\mathcal{N})}\right) \\
&= \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{g}\pi_k^{(\mathcal{N})}\cdot\left(1-\pi_k^{(\mathcal{N})}\right) \\
&= n\cdot\frac{1}{n}\sum_{k=1}^{g}\pi_k^{(\mathcal{N})}\cdot\left(1-\pi_k^{(\mathcal{N})}\right) \\
&= \sum_{k=1}^{g}\pi_k^{(\mathcal{N})}\cdot\left(1-\pi_k^{(\mathcal{N})}\right).
\end{aligned}
$$

This is precisely the Gini index CART use for splitting with Brier score. Gini impurity can thus be viewed as the expected frequency with which the training samples will be misclassified in a given node $\mathcal{N}$.

In other words, in constructing CART with minimal Gini impurity, we minimize the expected rate of misclassification across the training data.