Assignment 3

Group 2

2023-10-29

Task 1

Task 2

Task 3

Task 4

Task 5

Task 6

Task 7

Task 8

The data generating process is given by $y = x + \epsilon$, where $x \sim N(0, 1), \epsilon \sim N(0, 0.1)$. We first draw 100 samples of 100 observations for y and store it in a list.

```
# get 100 training data sets of length 100
training_data <- lapply(1:100, function(i) {
  x <- rnorm(100)
  noise <- rnorm(100, mean = 0, sd = sqrt(0.1))
  y <- x + noise
  data.frame(y = y, x = x)
})
```

To approximate the prediction error we draw 100 test data sets from the data generating process but of length 20 each.

```
# get 100 test data sets
test_data <- lapply(1:100, function(i) {
  x <- rnorm(20)
  noise <- rnorm(20, mean = 0, sd = sqrt(0.1))
  y <- x + noise
  data.frame(y = y, x = x)
})
```

We then first estimate 100 linear regressions of the form $y = \beta_0 + \beta_1 x + \epsilon$.

```r
# get OLS fits
linear_fits <- lapply(training_data, lm, formula = y ~ x)

lm_pred_err <- lapply(linear_fits, function(l) {
  # get mean squared loss for each test data set
  mean_squared_loss <- lapply(test_data, function(d) {
    pred <- predict(l, newdata = d)
    mean((d$y - pred)^2)
  })
  # prediction error
  mean(unlist(mean_squared_loss))
})
```

Next, we fit regression trees for each of the training data sets.

```r
library(caret)
```

```
## Lade nötiges Paket: lattice

##
## Attache Paket: 'caret'

## Das folgende Objekt ist maskiert 'package:purrr':
##
##     lift
```

```r
library(rpart)

# fit trees
tree_fits <- lapply(training_data, function(d) {
  # use caret for convenience, cost complexity
  # pruning is conducted automatically and best
  # model is by default chosen by RMSE
  train(y ~ x,
        data = d,
        method = "rpart")
})

# determine size as the number of nodes
tree_sizes <- lapply(tree_fits, function(t) {
  # get frame and count non leaf entries
  sum(t$finalModel$frame$var != "<leaf>")
})

# get the prediction error by monte carlo simulation,
# i.e. by calculating the mean squared error for 1000 test
# data sets and averaging those again
tree_pred_err <- lapply(tree_fits, function(t) {
  # get mean squared loss for each test data set
  mean_squared_loss <- lapply(test_data, function(d) {
    pred <- predict(t, newdata = d)
    mean((d$y - pred)^2)
```

```
  })
  # prediction error
  mean(unlist(mean_squared_loss))
})
```

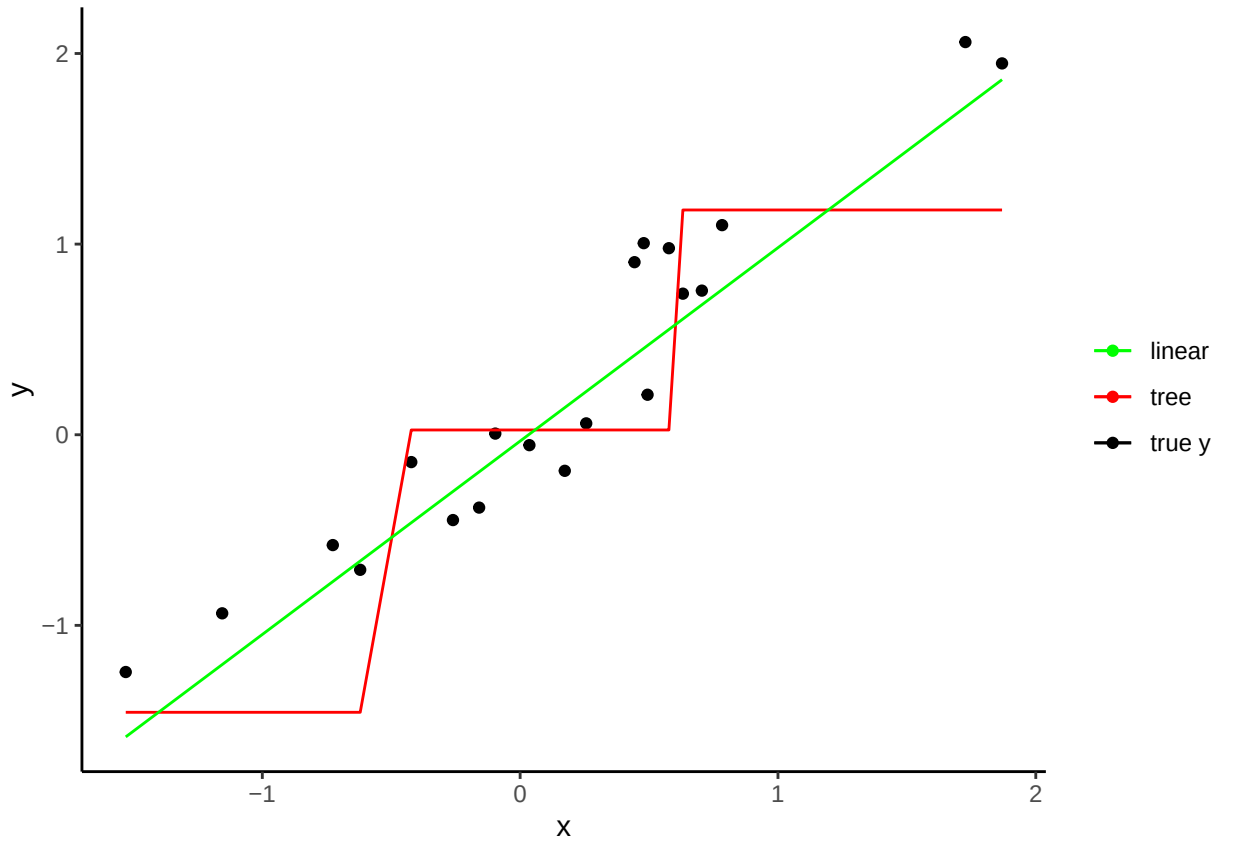
As instructed we now pick a test data seit, in our case the very first one and look at the actual values vs the predicted values for both the linear model as well as the regression tree.

```
pred_l <- predict(linear_fits[[1]], newdata = test_data[[1]])

pred_t <- predict(tree_fits[[1]], newdata = test_data[[1]])

data <- test_data[[1]]
data$l <- pred_l
data$t <- pred_t

data %>%
  ggplot(aes(x = x)) +
  geom_point(aes(y = y, color = "true y")) +
  geom_line(aes(y = t, color = "tree")) +
  geom_line(aes(y = l, color = "linear")) +
  scale_color_manual(values = c("true y" = "black", "tree" = "red", "linear" = "green")) +
  theme_classic() +
  theme(legend.title = element_blank())
```



Then we summarise different tree sizes

```
table(unlist(tree_sizes))
```

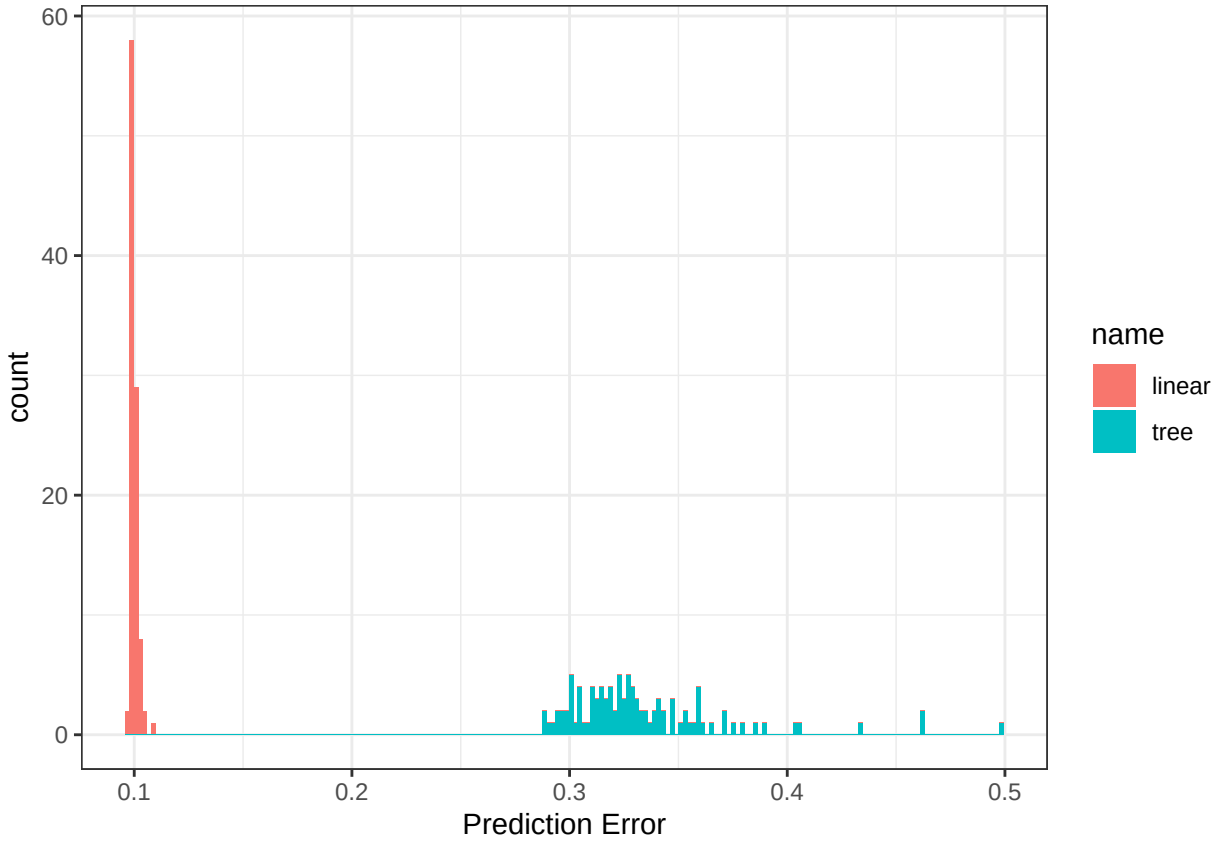
```
##
##  1  2
##  3 97
```

and observe that there are always 2 nodes in the trees.

Next we summarise the estimated prediction errors. We use a two variable histogram to do so.

```
prediction_errors <- data.frame(linear = unlist(lm_pred_err),
                                tree = unlist(tree_pred_err))

prediction_errors %>%
  pivot_longer(cols = everything()) %>%
  ggplot(aes(x = value, fill = name)) +
  geom_histogram(bins = 200) +
  theme(legend.title = element_blank()) +
  theme_bw() +
  labs(x = "Prediction Error")
```



What we observe is that the tree indeed has problems to capture the clearly linear relationship between y and x. This is because the tree takes the originally continuous space and discretizes it, such that predictions do not correspond to a certain value of x but rather a range within the support of x. One can see that quite clearly in the plot that visualizes the predictions vs the true value of y in the test data set. The tree predicts in a stepwise manner while the linear regression perfectly captures the linear relationship. Also,

we can see that the prediction errors are much lower for linear regression compared to the regression trees which is no surprise looking at the difficulties regression trees have with linear relationships.

Task 9

Task 10

We draw 10 observations from the data generating process $y \sim N(0, 3)$.

```
set.seed(123)
# draw sample s
n = 10
y <- rnorm(n = n, mean = 0, sd = sqrt(3))

# calculate mean
mean_sample <- mean(y)

# bootstrap: draw 10 times with replacement from the sample,
# do this 1000 times
B = 1000
bootstrap_samples <- lapply(1:B, function(i) {
  sample(x = y, size = n, replace = T)
})

# calculate the means for each bootstrap sample
means_b <- do.call("c", lapply(bootstrap_samples, mean))

# calculate the mean over the means
mean_sample
```

```
## [1] 0.1292554
```

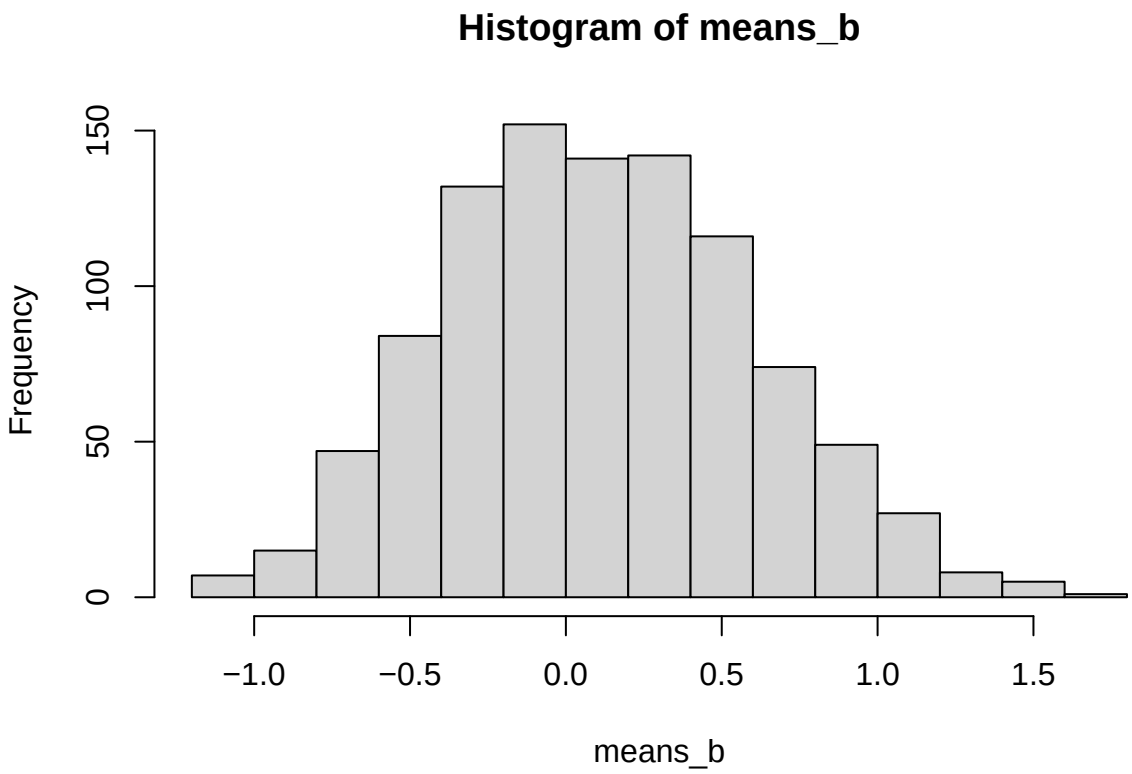
```
mean(means_b)
```

```
## [1] 0.1079687
```

```
sd(means_b)
```

```
## [1] 0.4960259
```

```
hist(means_b)
```



In the above we relied on the empirical distribution of y_s and estimated the mean by Monte-Carlo simulation by creating a distribution of means and taking the mean of this distribution as our bootstrapping estimate for the popopulation mean. We see that the sample mean and the bootstrapping estimate are very similar but quite far from the actual mean of the DGP. This observation is not surprising as as the initial sample has very limited information given $n = 10$ while the support of y is $(-\infty, \infty)$. Thus we know that this just can be an approximation of the true bootstrapping distribution.

The true bootstrapping distribution of the mean of the initially drawn sample would be the set of means drawn from all possible combinations of the values in the initial sample with replacement and without ordering. The number of these possible combinations in this case is $\binom{n+k-1}{k} = \binom{19}{10} = 92378$ if the order does not matter. If the order matters it would be just $n^n = 10^{10}$. However determining both sets is computationally demanding if not even infeasible. The bootstrapping estimate for the mean from the true bootstrapping distribution would be the population mean, so the bootstrapping procedure we used is just an approximation.