Problem Set 2: Non-Linear Methods

author 1 (matriculation number) author 2 (matriculation number)
author 3 (matriculation number)

2021-11-15 till 2021-12-05

- You may answer this problem set in groups of up to three students. Please state your names and matriculation numbers in the `YAML` header before submission (under the `author` field).
- Please hand in your answers via GitHub until 2021-12-05, 23:59 pm. When you have pushed your final results to GitHub, create a GitHub issue with the title "Hand-in" and mark Marius (GitHub username: *puchalla*) and Simon (GitHub username: *simonschoe*) as assignees or reference us in the issue description. Please ensure that all code chunks can be executed without error before pushing your last commit!
- You are supposed to Git commit your code changes regularly! In particular, each group member is required to contribute commits to the group's final submission to indicate equal participation in the group assignment. In case we observe a highly imbalanced distribution of contributions (indicated by the commits), individual seminar participants may be penalized.
- Make sure to answer **all** questions in this notebook! You may use plain text, R code and LaTeX equations to do so. You must always provide at least one sentence per answer and include the code chunks you used in order to reach a solution. Please comment your code in a transparent manner.
- For several exercises throughout the assignment you may find hints and code snippets that should support you in solving the exercises. You may rely on these to answer the questions, but you don't have to. Any answer that solves the exercise is acceptable.

Before starting, leverage the `pacman` package using the code chunk below. This package is a convenient helper for installing and loading packages at the start of your project. It checks whether a package has already been installed and loads the package if available. Otherwise it installs the package autonomously. Use `pacman::p_load()` to install and load the following packages:

- `tidyverse` (meta-package to load `dplyr`, `ggplot2` and co.)
- `tidymodels` (meta-package to load `rsample`, `parsnip`, `tune`, and co.)
- `ISLR` (contains the data set for *Task 1* and *Task 2*)
- `patchwork` (syntax for combining separate ggplots into the same graphic)
- `rpart` (functions for building classification and regression trees)
- `rpart.plot` (enhanced plotting capabilities for decision trees)
- `randomForest` (implementation of Leo Breiman's Random Forest algorithm)
- `gbm` (functions for performing gradient boosting)
- `kernlab` (kernel-based machine learning methods, incl. support vector machines)

Note: In case you rely on other additional or alternative packages for solving this assignment, please add them to the `pacman` *commands below.*

```r
# check if pacman is installed (install if evaluates to FALSE)
if (!require("pacman")) install.packages("pacman")
# load (or install if pacman cannot find an existing installation) the relevant packages
pacman::p_load(
  tidyverse, tidymodels, ISLR, patchwork,
  rpart, rpart.plot, randomForest, gbm, kernlab
)
```

In case you need any further help with the above mentioned packages and included functions, use the `help()` function build into RStudio (e.g., by running `help(rsample)`).

Task 1: Non-Linear Regression Techniques

For the first exercise of this assignment, you will work with wage data from the `ISLR` package (consult `help(Wage)` for more information on the dataset). You will construct various non-linear regressions using different *basis functions*, such as polynomials, step-wise constants and splines. In particular, you will explore the relationship between `wage` and `age` to gain a solid understanding of the inner workings of the various techniques.

```r
data(Wage, package = "ISLR")

Wage %>%
  tibble::as_tibble() %>%
  tibble::glimpse()
```

```
> Rows: 3,000
> Columns: 11
> $ year       <int> 2006, 2004, 2003, 2003, 2005, 2008, 2009, 2008, 2006, 2004,~
> $ age        <int> 18, 24, 45, 43, 50, 54, 44, 30, 41, 52, 45, 34, 35, 39, 54,~
> $ maritl     <fct> 1. Never Married, 1. Never Married, 2. Married, 2. Married,~
> $ race       <fct> 1. White, 1. White, 1. White, 3. Asian, 1. White, 1. White,~
> $ education  <fct> 1. < HS Grad, 4. College Grad, 3. Some College, 4. College ~
> $ region     <fct> 2. Middle Atlantic, 2. Middle Atlantic, 2. Middle Atlantic,~
> $ jobclass   <fct> 1. Industrial, 2. Information, 1. Industrial, 2. Informatio~
> $ health     <fct> 1. <=Good, 2. >=Very Good, 1. <=Good, 2. >=Very Good, 1. <=~
> $ health_ins <fct> 2. No, 2. No, 1. Yes, 1. Yes, 1. Yes, 1. Yes, 1. Yes, 1. Ye~
> $ logwage    <dbl> 4.318063, 4.255273, 4.875061, 5.041393, 4.318063, 4.845098,~
> $ wage       <dbl> 75.04315, 70.47602, 130.98218, 154.68529, 75.04315, 127.115~
```

Task 1.1

First, explore the relationship between `wage` and `age` by fitting and visualizing a linear, quadratic, cubic, quartic and quintic regression using the code snippet below.

Step 1: Define the relevant preprocessing steps using the `recipes` package. Since non-linear regressions rely on simple transformations of the predictors, the different polynoms can be generated via `step_poly()`.

```r
lin_rec   <- recipe(wage ~ age, data = Wage)
quad_rec  <- lin_rec %>% step_poly(age, degree = 2)
cub_rec   <- lin_rec %>% step_poly(age, degree = 3)
quart_rec <- lin_rec %>% step_poly(age, degree = 4)
quint_rec <- lin_rec %>% step_poly(age, degree = 5)
```

Step 2: Specify the desired machine learning model using `parsnip`.

```r
lm_spec <-
  linear_reg() %>%
  set_mode("regression") %>%
  set_engine("lm")
```

Step 3: Bring everything together using `workflows` and train the five models.

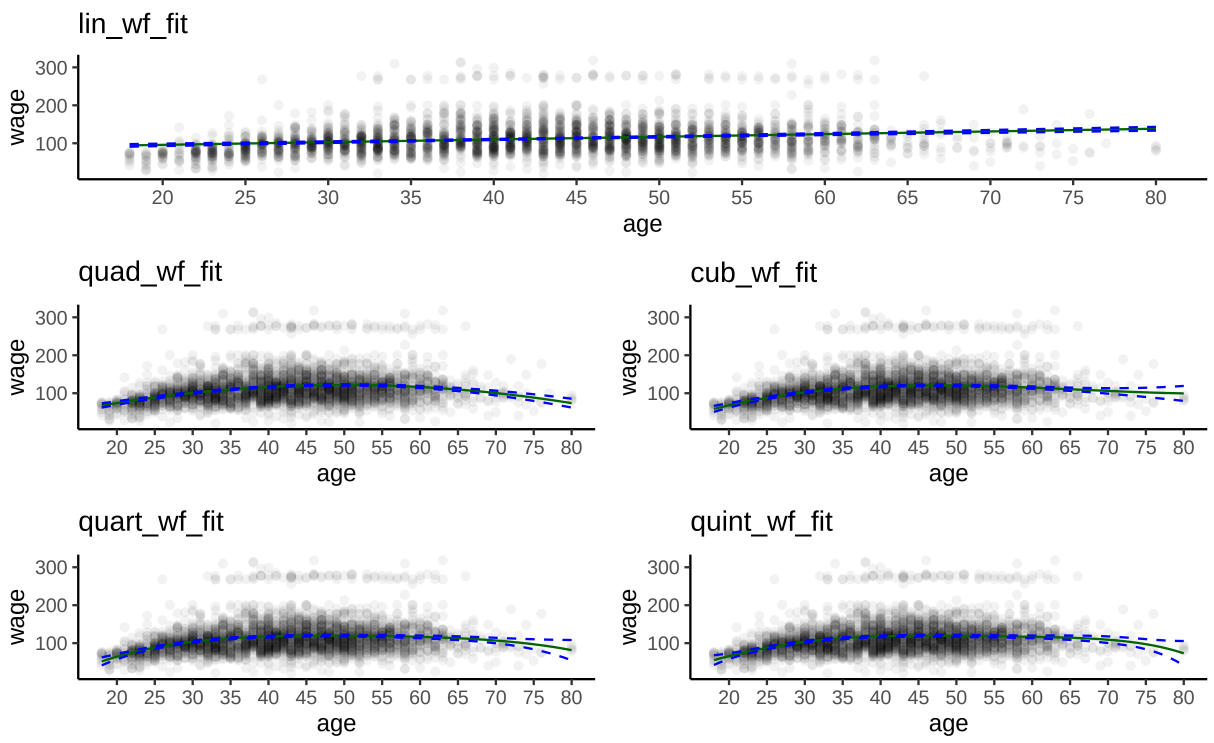
```r
lin_wf_fit    <- workflow(lin_rec, lm_spec) %>% fit(data = Wage)
quad_wf_fit   <- workflow(quad_rec, lm_spec) %>% fit(data = Wage)
cub_wf_fit    <- workflow(cub_rec, lm_spec) %>% fit(data = Wage)
quart_wf_fit  <- workflow(quart_rec, lm_spec) %>% fit(data = Wage)
quint_wf_fit  <- workflow(quint_rec, lm_spec) %>% fit(data = Wage)
```

Step 4: Define a helper function for plotting the results.

```r
plot_model <- function(wf_fit, data) {

  predictions <-
    tibble::tibble(age = seq(min(data$age), max(data$age))) %>%
    dplyr::bind_cols(
      predict(wf_fit, new_data = .),
      predict(wf_fit, new_data = ., type = "conf_int")
    )

  p <- ggplot2::ggplot(aes(age, wage), data = data) +
    geom_point(alpha = 0.05) +
    geom_line(aes(y = .pred),
              data = predictions, color = "darkgreen") +
    geom_line(aes(y = .pred_lower),
              data = predictions, linetype = "dashed", color = "blue") +
    geom_line(aes(y = .pred_upper),
              data = predictions, linetype = "dashed", color = "blue") +
    scale_x_continuous(breaks = seq(20, 80, 5)) +
    labs(title = substitute(wf_fit)) +
    theme_classic()

  return(p)

}
```

Step 5: Visualize the different fits. Note that the fitted curves are generating by obtaining predictions for each possible value for `age` in the underlying data.

```r
              plot_model(lin_wf_fit, Wage) /
(plot_model(quad_wf_fit, Wage) + plot_model(cub_wf_fit, Wage)) /
(plot_model(quart_wf_fit, Wage) + plot_model(quint_wf_fit, Wage))
```



How many model coefficients are learned by each of the five different models (incl. the intercept)?

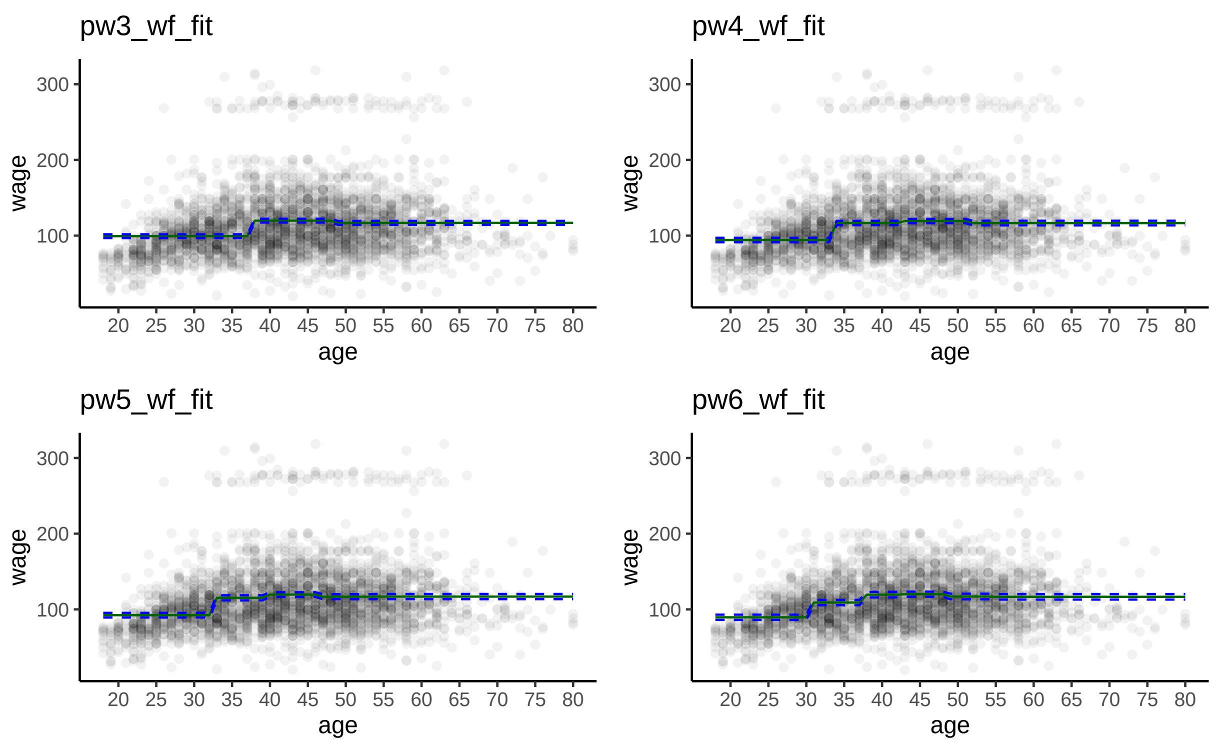
Task 1.2

Second, explore the relationship between `wage` and `age` by fitting and visualizing a set of piece-wise linear functions (i.e. *piecewise regression*). This can be achieved by simply adjusting the transformation step in the modeling `recipe` to `step_discretize()` which splits the continuous `age` predictors into `num_breaks` quantiles of equal size.

```r
pw3_rec <- lin_rec %>% step_discretize(age, num_breaks = 3, min_unique = 5)
pw4_rec <- lin_rec %>% step_discretize(age, num_breaks = 4, min_unique = 5)
pw5_rec <- lin_rec %>% step_discretize(age, num_breaks = 5, min_unique = 5)
pw6_rec <- lin_rec %>% step_discretize(age, num_breaks = 6, min_unique = 5)
```

```r
pw3_wf_fit <- workflow(pw3_rec, lm_spec) %>% fit(data = Wage)
pw4_wf_fit <- workflow(pw4_rec, lm_spec) %>% fit(data = Wage)
pw5_wf_fit <- workflow(pw5_rec, lm_spec) %>% fit(data = Wage)
pw6_wf_fit <- workflow(pw6_rec, lm_spec) %>% fit(data = Wage)
```

```r
(plot_model(pw3_wf_fit, Wage) + plot_model(pw4_wf_fit, Wage)) /
(plot_model(pw5_wf_fit, Wage) + plot_model(pw6_wf_fit, Wage))
```



What is the predicted income of a person with `age` equal to 37 for each piecewise model? Considering the flexible `pw6_wf_fit` model configuration and in particular the six breakpoints above: Exceeding which (approximate) value of `age` correlates strongest with `wage`? For which break-point do you observe barely any influence? Explain your answer.

Hint: If you find the plots to be badly rendered on your system, you may click the `Show in New Window` button in the top right corner of your chunk output to scale the plot size.

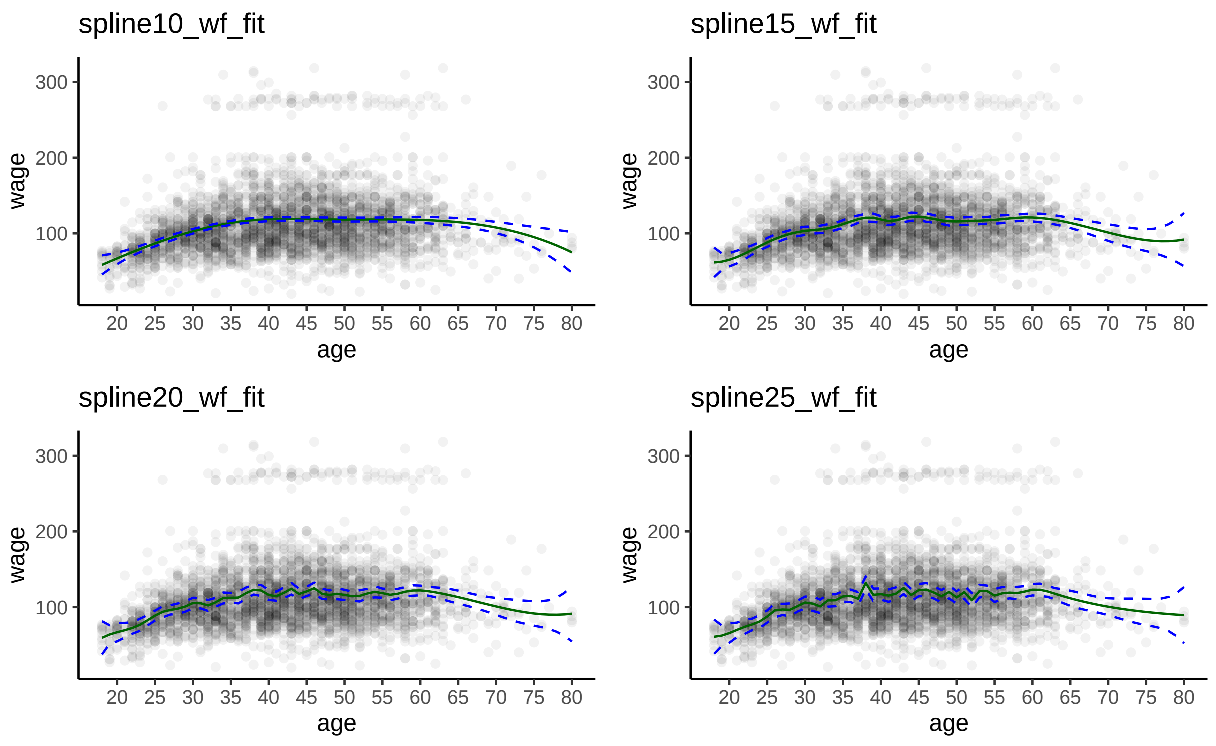
Task 1.3

Third, explore the relationship between `wage` and `age` using cubic regression splines. Again, a simple modification of the preprocessing recipe allows us to transform the `age` predictor accordingly.

```
spline10_rec <- lin_rec %>% step_bs(age, degree = 3, deg_free = 5)
spline15_rec <- lin_rec %>% step_bs(age, degree = 3, deg_free = 15)
spline20_rec <- lin_rec %>% step_bs(age, degree = 3, deg_free = 25)
spline25_rec <- lin_rec %>% step_bs(age, degree = 3, deg_free = 35)
```

```
spline10_wf_fit <- workflow(spline10_rec, lm_spec) %>% fit(data = Wage)
spline15_wf_fit <- workflow(spline15_rec, lm_spec) %>% fit(data = Wage)
spline20_wf_fit <- workflow(spline20_rec, lm_spec) %>% fit(data = Wage)
spline25_wf_fit <- workflow(spline25_rec, lm_spec) %>% fit(data = Wage)
```

```
(plot_model(spline10_wf_fit, Wage) + plot_model(spline15_wf_fit, Wage)) /
(plot_model(spline20_wf_fit, Wage) + plot_model(spline25_wf_fit, Wage))
```



How many knots are produced by the four different spline regressions? To answer this question, reflect upon the number of coefficients (i.e. degrees of freedom) of a single cubic function vis-à-vis the degrees of freedom of the specified spline regressions. Finally, construct and plot a polynomial regression with $d = 15$ (by recycling the relevant code snippets above) and explain how a cubic spline regression with 15 degrees of freedom differs from a polynomial regression with $d = 15$.

Hint: You may find it helpful to check out ch. 7.4 of the ISL textbook before answering the questions.

Task 1.4

Which of the following statements are true, which are false? Explain your answer in case you select "false".

1. A natural cubic spline has fewer degrees of freedom than a cubic spline.
2. Smoothing splines attempt to reduce the variance of a model fit by penalizing its second-order derivative.
3. A regression spline with polynomials of degree $M - 1$ has continuous derivatives up to order $M - 2$, but not at the knots.
4. A regression spline of order 3 with 4 knots has 8 basis functions (not counting the intercept).

Task 2: Tree-Based Methods

In chapter *"8.3.1 Fitting Classification Trees"* of the ISLR text book, a classification tree is fitted on the `Carseats` data to predict `sales` after encoding it as a categorical response variable. Now your goal is to predict car seat sales using regression trees and related ensemble-approaches (i.e. bagging, random forests and boosting), treating the response as a numerical variable.

```
data(Carseats, package = "ISLR")

Carseats %>%
  tibble::as_tibble()
```

```
> # A tibble: 400 x 11
>    Sales CompPrice Income Advertising Population Price ShelveLoc   Age Education
>    <dbl>     <dbl>  <dbl>       <dbl>      <dbl> <dbl> <fct>     <dbl>     <dbl>
>  1  9.5        138     73          11        276   120 Bad          42        17
>  2 11.2        111     48          16        260    83 Good         65        10
>  3 10.1        113     35          10        269    80 Medium       59        12
>  4  7.4        117    100           4        466    97 Medium       55        14
>  5  4.15       141     64           3        340   128 Bad          38        13
>  6 10.8        124    113          13        501    72 Bad          78        16
>  7  6.63       115    105           0         45   108 Medium       71        15
>  8 11.8        136     81          15        425   120 Good         67        10
>  9  6.54       132    110           0        108   124 Medium       76        10
> 10  4.69       132    113           0        131   124 Medium       76        17
> # ... with 390 more rows, and 2 more variables: Urban <fct>, US <fct>
```

Data dictionary:

- `Sales`: Unit sales (in thousands) at each location
- `CompPrice`: Price charged by competitor at each location
- `Income`: Community income level (in thousands of dollars)
- `Advertising`: Local advertising budget for company at each location (in thousands of dollars)
- `Population`: Population size in region (in thousands)
- `Price`: Price company charges for car seats at each site
- `ShelveLoc`: A factor with levels `Bad`, `Good` and `Medium` indicating the quality of the shelving location
- `Age`: Average age of the local population
- `Education`: Education level at each location
- `Urban`: A factor with levels `No` and `Yes` to indicate whether the store is in an urban or rural location
- `US`: A factor with levels `No` and `Yes` to indicate whether the store is in the US or not

Task 2.1

Split the data set into a training set and a test set using `set.seed(2021)`, the `rsample` package and a train-test-ratio of 3:1. Extract the training and test samples and write them to a variable called `train_set_cs` and `test_set_cs`, respectively.

Task 2.2

Consider the following model formula:

`Sales ~ .`

Fit a regression tree to your training set using the `rpart` package. Use the following code snippet for this purpose (replace `___` with your training set and set the code chunk option to `eval=T` in order to execute the chunk):

```r
set.seed(2021)

rpart::rpart(
  Sales ~ ., data = ___,
  method = "anova",
  control = rpart.control(cp = 0.015)
)
```

Alternatively, use the following `tidymodels` pipeline to achieve the same result:

```r
set.seed(2021)

tree_rec <-
  recipe(formula = Sales ~ ., data = ___)

tree_spec <-
  decision_tree() %>%
  set_mode("regression") %>%
  set_engine("rpart") %>%
  set_args(cost_complexity = 0.015)

tree_wf <-
  workflow(tree_rec, tree_spec)

tree_wf_fit <- tree_wf %>%
  fit(___)
```

Generate a `summary()` of the fitted tree and plot the tree (using the `rpart.plot` package). For the purpose of plotting, use the following code snippet (replace `___` with the fitted tree object):

```r
rpart.plot::rpart.plot(___, type = 0, tweak = 1.1, extra = 1, uniform = FALSE)
```

Note: When using the `tidymodels` approach, you have to extract the fitted model from the trained `workflow` object using `extract_fit_engine()`. Refer to the documentation of the `rpart.plot` package in order to learn more about the aesthetic arguments of the `rpart.plot` function (e.g., `tweak` or `uniform`).

Finally, answer the following questions:

i. What is the response variable and what are the predictors in this regression problem?
ii. What is the purpose of the `cp` / `cost_complexity` hyperparameter?
iii. What is the test RMSE and how would you interpret it relative to the mean value for `Sales` in the test set?
iv. What is the *tree size* (number of terminal nodes) and *tree depth* (longest path from root to leaf, i.e. maximum number of split nodes)? Which predictors are used for constructing the tree?
v. Pick one of the terminal nodes. How would you interpret an observation that falls into this node? Provide an interpretation that considers the path along the tree in order to reach the node.
vi. Give an interpretation of the branch lengths and the terminal node colors in the decision tree plot.
vii. Which result does the regression tree predict for a car seat that is of good quality, costs 111$ and where the local advertising budget amounts to 8,000$?

Note: In contrast to the text book we rely on the more powerful `rpart` *instead of the* `tree` *package. The decision tree implementation in the* `rpart` *package mainly differs in its way of handling surrogate splits (by referring to the second-best predictor for splitting in the presence of a missing value).*

Task 2.3

Under the hood, `rpart` can perform cross-validation (CV) for you. Refit the tree on the training set using `cp = 0.001`, `xval = 5` for 5-fold CV and `set.seed(2021)`. Alternatively, you may also implement a `tidymodels` pipeline with the following hyperparameter grid for `cost_complexity` (note that the hyperparameter tuning operation may take some time, depending on your hardware):

```
dials::grid_regular(
  cost_complexity() %>% range_set(c(-3, -0.5)),
  levels = 25
)
```

```
> # A tibble: 25 x 1
>    cost_complexity
>              <dbl>
>  1         0.001
>  2         0.00127
>  3         0.00162
>  4         0.00205
>  5         0.00261
>  6         0.00332
>  7         0.00422
>  8         0.00536
>  9         0.00681
> 10         0.00866
> # ... with 15 more rows
```

Why can it generally make sense to prune a decision tree? What is the optimal level of tree complexity in this case? Which level of tree complexity would you suggest if your goal is to generate a parsimonious tree using the one-standard error rule? How does this parsimonious tree compare to the optimal solution in terms of tree size?

Hint: In order to implement CV using `rpart`*, check out the* `control` *argument of* `rpart`*. Note that the output columns* `rel error` *and* `xerror/xstd` *refer to* $1 - R^2$ *on the train and test set, respectively.*

Task 2.4

The following code snippet extracts *variable importance weights* from a fitted decision tree object (again replace `___` with your fitted tree object from *Task 2.3*) and plots them using `ggplot2`:

```
vip_scores <- ___$variable.importance

vip_scores

vip_scores %>%
  dplyr::bind_rows() %>%
  tidyr::pivot_longer(
    everything(),
    names_to = "var", values_to = "var_importance"
  ) %>%
  ggplot2::ggplot(aes(
    x = var_importance / sum(var_importance) * 100, # rescale importance weights
    y = forcats::fct_reorder(var, var_importance, .desc = T) # reorder columns
  )) +
  geom_col(fill = "lightblue", color = "black") +
  geom_text(
    aes(label = scales::percent(var_importance / sum(var_importance)), hjust = -.5)) +
  scale_x_continuous(limits = c(0, 50)) +
  labs(x = "Variable Importance") +
  theme_classic() +
  theme(axis.title.y = element_blank())
```

Use the code to generate a variable importance ranking for the tree constructed in the previous task. What do the importance weights indicate? What are the three most important predictors and how do they compare to each other in terms of variable importance?

Task 2.5

Next, you are supposed to use the `randomForest` package to fit various ensemble models for predicting car seat sales. First, use bagging with 1,000 bootstrapped samples and all available predictors to fit a bagged ensemble on the training set, estimate the test RMSE and identify the five most important variables using the `randomForest::importance()` function. Set the random seed to `2021`.

Alternatively, use `tidymodels` to obtain an almost identical result:

```
bag_spec <-
  rand_forest() %>%
  set_args(trees = 1000, mtry = .cols()) %>%
  set_mode("regression") %>%
  set_engine("randomForest")
```

Task 2.6

Second, fit a random forest with 1,000 trees using `tidymodels`. Vary the number of predictors (`mtry`) and try out all possible values for `mtry` to find the model specification with the lowest validation set RMSE using 5-fold CV. What is the test RMSE for the optimal `mtry` configuration? What are the five most important predictors for the optimal `mtry` configuration? Does the variable importance ranking differ from the ranking in *Task 2.5*? Set the random seed to `2021`.

Using `tidymodels` is probably the easiest way to tackle this task (unless you like to implement numerous for-loops). You may leverage the code snippet below to kickstart your analysis (replace `___` according to the exercise description):

```
set.seed(2021)

folds_cs <-
  vfold_cv(train_set_cs, v = ____)

rf_rec <-
  recipe(formula = ___ ~ ., data = train_set_cs)
```

```r
rf_spec <-
  rand_forest() %>%
  set_args(trees = ___, mtry = ___) %>%
  set_mode("regression") %>%
  set_engine("randomForest")

rf_wf <-
  workflow(rf_rec, rf_spec)

rf_grid <-
  grid_regular(
    finalize(mtry(), train_set_cs %>% select(-Sales)),
    levels = ___
  )

rf_wf_fit <-
  tune_grid(
    rf_wf, folds_cs,
    grid = grid, metrics = metric_set(rmse),
    control = control_grid(verbose = T)
  )

rf_wf_fit %>%
  show_best(metric = "rmse", n = 10)
```

```r
rf_wf_final <- rf_wf %>%
  finalize_workflow(select_best(rf_wf_fit, metric = "rmse"))

rf_wf_final %>%
  last_fit(split = split_cs, metrics = metric_set(___)) %>%
  collect_metrics()

rf_wf_final %>%
  last_fit(split = split_cs, metrics = metric_set(___)) %>%
  extract_fit_engine() %>%
  randomForest::importance()
```

Task 2.7

Third, leverage the `gbm` package and train a boosted regression tree with 100 trees while setting `interaction.depth = 3` (maximum number of split nodes). Note that this time we are not interested in tuning the model, hence, the whole training set serves the model training. How does the model perform in terms of test set $RMSE$ compared to the previous ensembles (i.e. the bagging model and the optimal random forest model)? Produce a partial dependence plot for `Price`, `ShelveLoc` and `c("Price", "ShelveLoc")` using the `plot.gbm()` function. What can you say about the marginal effect of these two predictors on car seat sales? Set the random seed to 2021.

Task 2.8

Please give a **short** explanation of Bayesian Additive Regression Trees (BART). Explain the method and how it deviates from Random Forests and “regular” decision trees. How could the BART results differ from the previous results?

Task 3: Support Vector Machines

In this task, you will work with synthetic data from the Elements of Statistical Learning text book. An important feature of this dataset is, that since it is a simulated dataset, we know the true Bayes decision boundary. First, download the data-file given the command below:

```
load(url("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda"))
tibble::glimpse(ESL.mixture)
```

```
> List of 8
>  $ x       : num [1:200, 1:2] 2.5261 0.367 0.7682 0.6934 -0.0198 ...
>  $ y       : num [1:200] 0 0 0 0 0 0 0 0 0 0 ...
>  $ xnew    : 'matrix' num [1:6831, 1:2] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -1.9 -1.8 -1.7 ...
>   ..- attr(*, "dimnames")=List of 2
>   .. ..$ : chr [1:6831] "1" "2" "3" "4" ...
>   .. ..$ : chr [1:2] "x1" "x2"
>  $ prob    : num [1:6831] 3.55e-05 3.05e-05 2.63e-05 2.27e-05 1.96e-05 ...
>   ..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
>  $ marginal: num [1:6831] 6.65e-15 2.31e-14 7.62e-14 2.39e-13 7.15e-13 ...
>   ..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
>  $ px1     : num [1:69] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -1.9 -1.8 -1.7 ...
>  $ px2     : num [1:99] -2 -1.95 -1.9 -1.85 -1.8 -1.75 -1.7 -1.65 -1.6 -1.55 ...
>  $ means   : num [1:20, 1:2] -0.2534 0.2667 2.0965 -0.0613 2.7035 ...
```

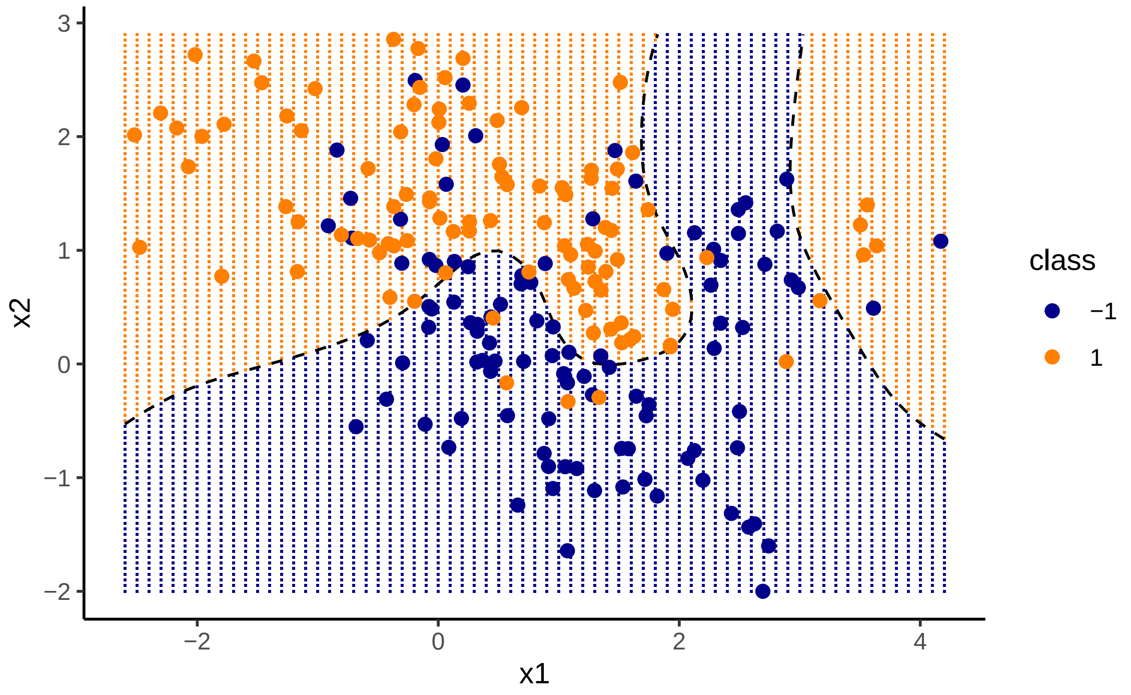
Data dictionary:

- `x`: 200 x 2 matrix of training predictors `x1` and `x2`
- `y`: response (`lgl` vector with a class distribution of 1:1)
- `px1`: 69 grid coordinates for `x1`
- `px2`: 99 grid coordinates for `x2`
- `prob`: vector of 6831 (= 69 * 99) probabilities (of class TRUE) at each grid coordinate

The following code plots the Bayes decision boundary along with the training data as well as small grid points which visualize the class membership:

```
mixture_data <-
  tibble::tibble(
    x1 = ESL.mixture$x[,1],
    x2 = ESL.mixture$x[,2],
    y = dplyr::if_else(ESL.mixture$y == 1, 1, -1) %>% as.factor
  )

contour_data_bayes <-
  tidyr::expand_grid(
    x2 = ESL.mixture$px2,
    x1 = ESL.mixture$px1
  ) %>%
  dplyr::mutate(
    proba = ESL.mixture$prob,
    class = dplyr::if_else(proba > .5, 1, -1) %>% as.factor
  )
```

```r
p <- ggplot2::ggplot(data = contour_data_bayes, aes(x = x1, y = x2)) +
  geom_point(aes(z = NULL, color = class), shape = ".") +
  geom_contour(aes(z = proba), color = "black", size = .5, bins = 2, lty = "dashed") +
  geom_point(data = mixture_data, aes(x = x1, y = x2, color = y), size = 2) +
  scale_color_manual(values = c("blue4", "darkorange1")) +
  theme_classic()

p
```



Task 3.1

Briefly explain the terms *Bayes classifier*, *Bayes decision boundary* and *Bayes error rate* in your own words and use them to explain the graph above. Why is the *Bayes error rate* positive in this example?

Hint: Refer to pp. 37-39 of the ISLR text book in order to answer this question.

Task 3.2

In this task we know the true Bayes decision boundary since the data was simulated. Hence, we have complete information about the data generating process and know all the distributional parameters. When this is the case, we call the data *synthetic*. In reality, however, we are rarely informed about the true data generating process and have no access to the true class probabilities of each sample - rather we have to estimate them. Why can it still be insightful to work with synthetic data in machine learning, even though we already know the Bayes decision boundary?

Task 3.3

The following code fits two different support vector machine (SVM) models, `svm_lin_spec` and `svm_rbf_spec`, using the `kernlab` package as underlying engine for `parnsip`. The optimale hyperparameters are determined via 5-fold CV.

`svm_lin_spec`:

- The predictors are scaled to zero mean and unit variance (*z-normalization*).
- The model implements a linear kernel function.
- The `cost` parameter is tuned over a sequence of 10 values on the log-10 scale using grid search (when grid search boils down to finding a single optimal hyperparameter, we call it *line search*).

`svm_rbf_spec`:

- The predictors are scaled to zero mean and unit variance (*z-normalization*).
- The model implements a non-linear, radial basis kernel function.
- The `cost` and `gamma` parameters are tuned over a grid of 100 hyperparameter candidates using grid search.

```
set.seed(2021)

mixture_folds <-
  vfold_cv(mixture_data, v = 5)

svm_rec <-
  recipe(formula = y ~ ., data = mixture_data) %>%
  step_normalize(all_numeric_predictors())
```

```
svm_lin_spec <-
  svm_linear() %>%
  set_mode("classification") %>%
  set_engine("kernlab") %>%
  set_args(cost = tune())

wf_svm_lin <-
  workflow(svm_rec, svm_lin_spec)

grid_svm_lin <-
  grid_regular(
    cost(c(-5, 4), trans = log10_trans()),
    levels = 10
  )

wf_lin_fit <-
  tune_grid(
    wf_svm_lin, mixture_folds,
    grid = grid_svm_lin, metrics = metric_set(accuracy),
    control = control_grid(verbose = T, event_level = "second")
  )

wf_lin_final <-
  wf_svm_lin %>%
  finalize_workflow(select_best(wf_lin_fit)) %>%
  fit(mixture_data)
```

```
>  Setting default kernel parameters
```

```r
wf_lin_final
```

```
> == Workflow [trained] ==========================================================
> Preprocessor: Recipe
> Model: svm_linear()
>
> -- Preprocessor ----------------------------------------------------------------
> 1 Recipe Step
>
> * step_normalize()
>
> -- Model -----------------------------------------------------------------------
> Support Vector Machine object of class "ksvm"
>
> SV type: C-svc  (classification)
>  parameter : cost C = 0.01
>
> Linear (vanilla) kernel function.
>
> Number of Support Vectors : 172
>
> Objective Function Value : -1.5435
> Training error : 0.265
> Probability model included.
```

```r
svm_rbf_spec <-
  svm_rbf() %>%
  set_mode("classification") %>%
  set_engine("kernlab") %>%
  set_args(cost = tune(), rbf_sigma = tune())

wf_svm_rbf <-
  workflow(svm_rec, svm_rbf_spec)

grid_svm_rbf <-
  grid_regular(
    cost(c(-5, 4), trans = log10_trans()),
    rbf_sigma(c(-5, 4), trans = log10_trans()),
    levels = c(10, 10)
  )

wf_rbf_fit <-
  tune_grid(
    wf_svm_rbf, mixture_folds,
    grid = grid_svm_rbf, metrics = metric_set(accuracy),
    control = control_grid(verbose = T, event_level = "second")
  )

wf_rbf_final <-
  wf_svm_rbf %>%
  finalize_workflow(select_best(wf_rbf_fit)) %>%
  fit(mixture_data)
```

```r
wf_rbf_final
```

```
> == Workflow [trained] ==========================================================
> Preprocessor: Recipe
> Model: svm_rbf()
>
> -- Preprocessor ----------------------------------------------------------------
> 1 Recipe Step
>
> * step_normalize()
>
> -- Model -----------------------------------------------------------------------
> Support Vector Machine object of class "ksvm"
>
> SV type: C-svc  (classification)
>  parameter : cost C = 1
>
> Gaussian Radial Basis kernel function.
>  Hyperparameter : sigma =  10
>
> Number of Support Vectors : 145
>
> Objective Function Value : -77.2608
> Training error : 0.11
> Probability model included.
```

Reflect on the previous code chunks and output and address the following questions:

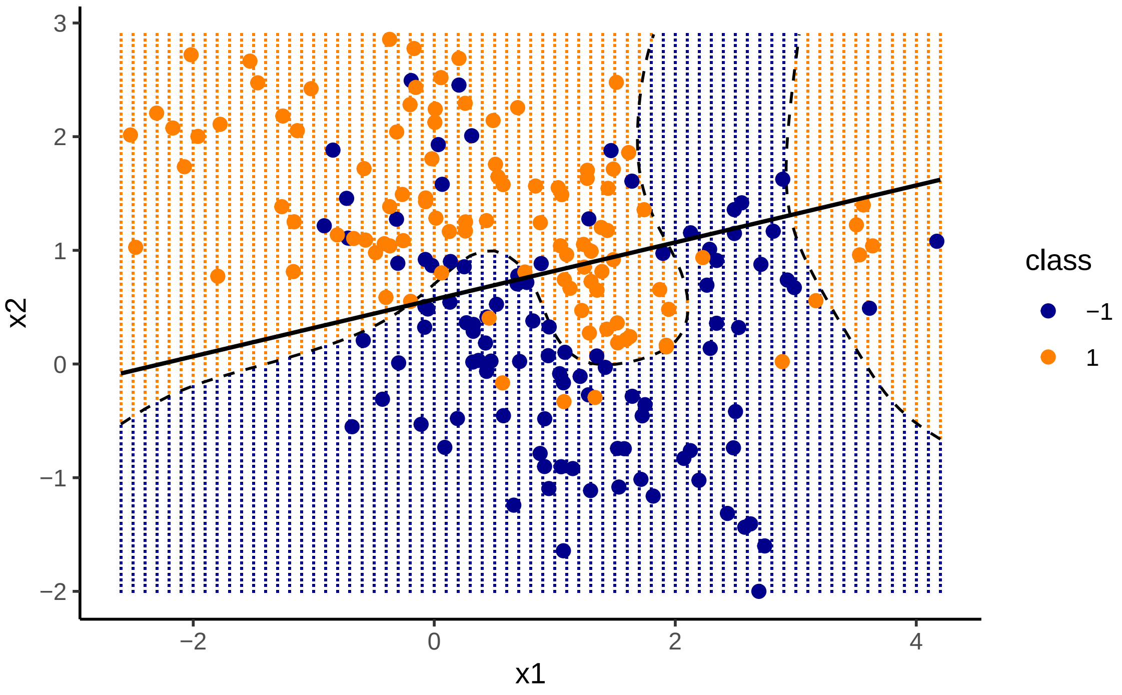
i. What is a *support vector*? What can you say about its position relative to and effect on the separating hyperplane? How many support vectors do you find for each model?
ii. What is the difference between the *support vector classifier* (*SVC*) and the *support vector machine* (*SVM*)?
iii. What is meant by the *kernel* specified in the `svm()` function?
iv. What is the purpose of the `cost` hyperparameter? How does it work in the context of the SVM?

Hint: Among others, ch. 9 of the ISLR text book should help you provide concise answers to these questions.

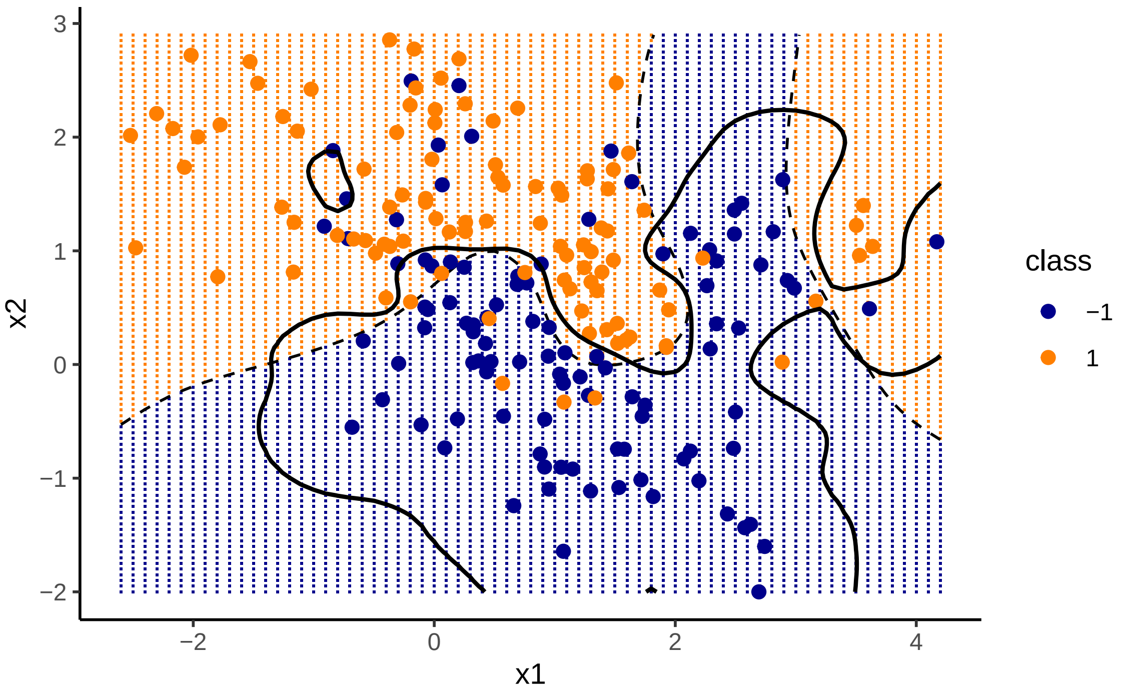
Task 3.4

Finally, use `wf_lin_final` and `wf_rbf_final` for prediction. For each classifier, the decision values (i.e. the distances from the separating hyperplane) are extracted to construct a decision boundary.

```r
# Linear SVM
svm_lin_preds <-
  contour_data_bayes %>%
  dplyr::bind_cols(
    decision_value =
      predict(wf_lin_final, ., type = "raw", opts = list(type = "decision"))
  )

p +
  geom_contour(
    data = svm_lin_preds,
    aes(x = x1, y = x2, z = decision_value),
    color = "black", size = .75, bins = 2
  )
```



```
# Non-Linear SVM
svm_rbf_preds <-
  contour_data_bayes %>%
  dplyr::bind_cols(
    decision_value =
      predict(wf_rbf_final, ., type = "raw", opts = list(type = "decision"))
  )

p +
  geom_contour(
    data = svm_rbf_preds,
    aes(x = x1, y = x2, z = decision_value),
    color = "black", size = .75, bins = 2
  )
```



How would you evaluate the respective model's decision boundary compared to the Bayes decision boundary? Which model do you expect to be more prone to overfitting when being tested on unseen data and why?