Problem Set 03

Foreword

The goal of this exercise session is to review different regularization techniques, both, in theory and application. We start out by briefly reviewing **ordinary least squares (OLS)** estimation and quickly move on to a more in-depth analysis of **ridge** and **lasso regression**.

Exercises

```r
library("tidyverse")
library("tidymodels")
library("glmnet")
```

Exercise 1: Loss functions in the context of OLS, ridge, and lasso regression

In Statistics I/II, we learned that OLS is the cornerstone of linear regression analysis. It allows us to explore and quantify the relationship between the response variable and the regressors in a relatively simple but meaningful way. We can extend the idea of a simple linear regression by adding a penalty term to the loss function we want to minimize. This process is called regularization and has been introduced in the lecture in terms of ridge and lasso regression.

The goal of this initial exercise is to review some theoretical aspects of OLS, ridge, and lasso regression.

Exercise 1a : OLS

Consider a simple linear model, with a quantitative response variable Y and a single predictor X. The simple linear model then assumes (among other things) that there is approximately a linear relationship between Y and X, i.e.,

$$Y \approx \beta_0 + \beta_1 X.$$

with unknown coefficients β_0, β_1. In order to obtain the best estimate β_0 and β_1 for a given sample we can minimize the MSE

$$\min_{\beta_0,\beta_1} MSE = \frac{1}{N}\sum_{n=1}^{N}(y_n - (\beta_0 + \beta_1 x_n))^2 \tag{1}$$

where $N = \text{length(Y)}$, $y_1, \ldots, y_N$ is a realized sample of Y, and $x_1, \ldots, x_N$ is a realized sample of X.

Show, that

$$\hat{\beta}_1 = \frac{\sum_{n=1}^{N}(x_n - \bar{x})(y_n - \bar{y})}{\sum_{n=1}^{N}(x_n - \bar{x})^2}$$
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}.$$

with $\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$ and $\bar{y} = \frac{1}{N}\sum_{n=1}^{N} y_n$ minimizes the minimization problem above. You can assume that the critical points calculated using the partial derivatives are in fact minima and that $\sum_{n=1}^{N}(x_n - \bar{x})^2 \neq 0$.

Exercise 1b: Ridge and lasso regression

Consider a linear model, where there is a quantitative response variable Y and a predictor $X = (1, X_1, ..., X_k)$, i.e. there are $k \in \mathbb{N}$ different features. The linear model then assumes (among other things) that there is approximately a linear relationship between Y and X, i.e.,

$$Y \approx \beta_0 + \beta_1 X_1 + ... + \beta_k X_k.$$

with unknown coefficients $\beta_0, ..., \beta_k$.

In the lecture, we have already seen that the loss function for ridge and lasso regression is given by

$$\mathcal{L}_{\text{ridge}}(\beta, \lambda) = \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 + \lambda \sum_{i=0}^{k} \beta_j^2$$

and

$$\mathcal{L}_{\text{lasso}}(\beta, \lambda) = \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 + \lambda \sum_{i=0}^{k} \mid \beta_j \mid,$$

where $\lambda \in [0, \infty)$.

Exercise 1b i:

Explain the following statement from the lecture:

"Ridge regression reduces the variance, but introduces bias."

Exercise 1b ii:

Consider the following statements and decide whether ridge or lasso regression should be applied.

1. You are building a predictive model for stock price prediction, and you have a large number of potential predictors. Some of these predictors might be highly correlated with each other.
2. You are modeling housing prices, and you want to prevent the model from overfitting to the training data.
3. You are working on a marketing project where you have a dataset with a mix of numerical and categorical features. You need to build a regression model to predict customer lifetime value.

Exercise 1b iii:

Come up with a scenario where a mixed model, i.e. an elastic net might be a good choice.

Exercise 2: Another lesson on data preparation and model specification

In this exercise, we will revisit the `rent` dataset, but instead of looking at rent prices in Augsburg, we will now consider rent prices in Munich. We will briefly prepare the dataset and create a recipe similar to Exercise 2b ii on Sheet 02 but in a more sophisticated fashion.

```
data_muc <- read.csv("data_muc_filtered.csv")
```

Exercise 2a: Data preparation

Explain how we preprocess our data in the following code chunk.

```
data_muc_filtered <- data_muc %>%
  select(!c("X","serviceCharge","heatingType","picturecount","totalRent",
            "firingTypes","typeOfFlat","noRoomsRange", "petsAllowed",
            "livingSpaceRange","regio3","heatingCosts", "floor",
            "date", "pricetrend")) %>%
  na.omit %>%
  mutate(
    interiorQual = factor(
      interiorQual,
      levels = c("simple", "normal", "sophisticated", "luxury"),
      labels = c("simple", "normal", "sophisticated", "luxury"),
      ordered = TRUE
    ),
    condition = factor(
      condition,
      levels = c("need_of_renovation", "negotiable","well_kept","refurbished",
                 "first_time_use_after_refurbishment",
                 "modernized", "fully_renovated", "mint_condition",
                 "first_time_use"),
      ordered = TRUE
    ),
    geo_plz = factor(geo_plz)
  ) %>%
  mutate_if(is.logical, ~ as.numeric(.)) %>%
  filter(baseRent <= 4000, livingSpace <= 200)
```

Exercise 2b: Setting up resampling

Similar to exercise Exercise 2a: i-iii on Sheet 02, create

- an initial split object called `split` with the `data_muc_filtered` tibble,
- a training set called `data_train` and a test set called `data_test` using the `training` and `testing` functions respectively, and

- an instance `folds` of the `vfold_cv` calss with the parameters `data = data_train` and `v = 10`.

```
set.seed(1)
  ##################
  ## Continue here ##
  ##################
```

Exercise 2c: Setting up a recipe

On the last exercise sheet, we created a simple recipe, only containing the formula we wanted to use on our simple linear model. This process can be extended by adding a multitude of different steps for preprocessing the underlying training data.

For each of the following updates and steps, explain their purpose and what they aim to achieve.

```
rec_lm <- recipe(
    formula = baseRent ~.,
    data = data_train
  ) %>%
  update_role(scoutId, new_role = "ID") %>%
  step_ordinalscore(interiorQual, condition)%>%
  step_dummy(geo_plz)%>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())
```

Exercise 3: Regularizing a linear model using lasso, ridge, and mixed models

In this last exercise we will make use of the previously performed data preparation by modeling a workflow and selecting the best model based on some performance metrics we will specify later.

Exercise 3a: Setting up and evaluating Lasso Regression

The approach is similar to Exercise 2c on Exercise Sheet 02, where I demonstrated how to train multiple models. So if you get stuck in some of the exercises you should revisit this sheet and try to reproduce the steps this way.

While this sub-exercise will seem kind of lengthy again, the other sub-exercises can be solved a lot quicker, since we can recycle many of the objects we create throughout this sub-exercise.

Exercise 3a i:

First, create an instance of the `linear_reg` class called `model_lasso` with parameters `penalty = tune()` and `mixture = 1.0`. By setting the `penalty` parameter to `tune()`, we specify that the penalty is a tuning parameter that we want to optimize later. By setting the `mixture` parameter to `1.0` we specify that the model should be a pure lasso regression (setting it to 0.0 indicates that we are using a pure ridge regression). If calling the model `model_lasso` results in the same output as below, you solved the exercise correctly.

```
Linear Regression Model Specification (regression)

Main Arguments:
  penalty = tune()
  mixture = 1

Computational engine: lm
```

Exercise 3a ii:

Since we have set the penalty value to `tune()` we have to specify a grid in which want to check for the best value. We can pass the grid values to our model by using the `set_engine` function.

```r
penalty <- seq(0, 4, length.out = 100)
```

Update the model `model_lasso` by completing the following Code snippet. Fill the gap by piping `model_lasso`to the `set_engine` function where you pass the parameters `engine = "glmnet"` and `path_values = penalty`. If you are interested in why we specifically have to use the `path_values` argument, you can check out this manual.

```r
model_lasso <- model_lasso %>%
  ##################
  ## Continue here ##
  ##################
```

You can check whether you have successfully updated the model by calling `model_lasso` and comparing your output to the one below. If they coincide you have solved the exercise correctly.

```
Linear Regression Model Specification (regression)

Main Arguments:
```

```
  penalty = tune()
  mixture = 1

Engine-Specific Arguments:
  path_values = penalty

Computational engine: glmnet
```

Exercise 3a iii:

Similarly to Exercise 2c i on Exercise Sheet 02, create a workflow called `glmnet_wflow` by creating an instance of the `workflow` class without passing any additional arguments, piping it to the `add_model` function with `model_lasso` as an argument, and finally piping it to the `add_recipe` function with `rec_lm` as an argument.

You can check whether you have successfully set up the workflow by calling `glmnet_wflow` and comparing your output to the one below. If they coincide you have solved the exercise correctly.

```
== Workflow ====================================================================
Preprocessor: Recipe
Model: linear_reg()

-- Preprocessor ----------------------------------------------------------------
4 Recipe Steps

* step_ordinalscore()
* step_dummy()
* step_zv()
* step_normalize()

-- Model -----------------------------------------------------------------------
Linear Regression Model Specification (regression)

Main Arguments:
  penalty = tune()
  mixture = 1

Engine-Specific Arguments:
  path_values = penalty

Computational engine: glmnet
```

Exercise 3a iv:

Given the following metric set and the previously created workflow, we can now finally train our lasso model.

```r
multi_metric <- metric_set(rsq,rmse)
```

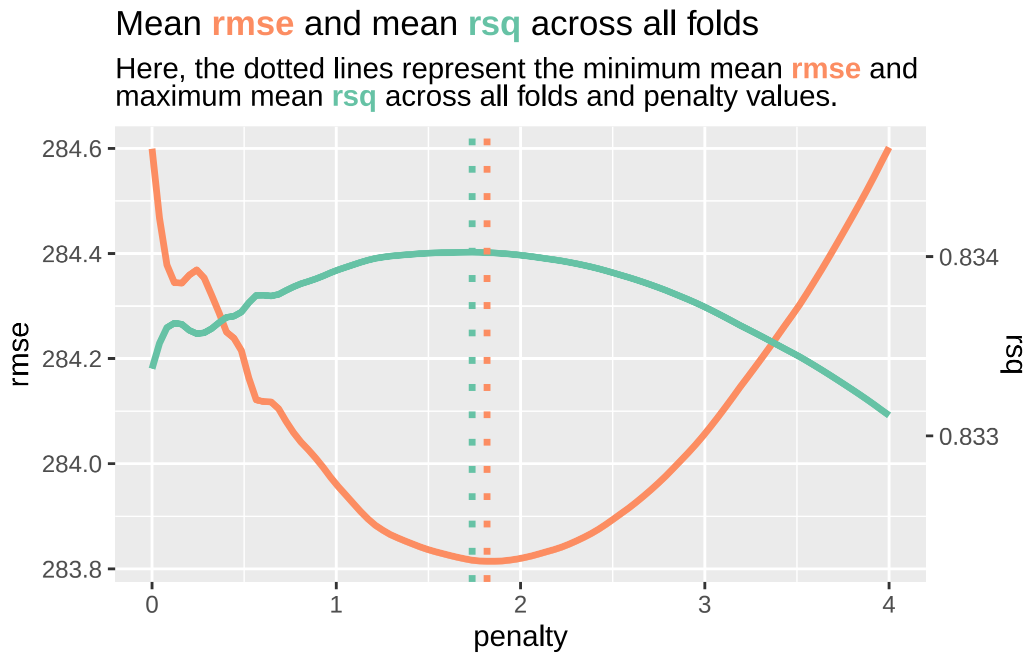
In order to do so, create an object called `glmnet_res` (res stands for resampling in that context) by assigning the `glmnet_wflow` object to it and piping it to the `tune_grid` function. Recall, that this is the exact same process as in Exercise 2c i on Sheet 02. As arguments for the `tune_grid` function, you have to pass `tibble(penalty), multi_metric`, and `folds`.

You can check whether you have successfully trained the model by calling `head(glmnet_res)` and comparing your output to the one below. If they coincide you have solved the exercise correctly.

```
# A tibble: 6 x 4
  splits             id     .metrics           .notes
  <list>             <chr>  <list>             <list>
1 <split [1471/164]> Fold01 <tibble [200 x 5]> <tibble [0 x 3]>
2 <split [1471/164]> Fold02 <tibble [200 x 5]> <tibble [0 x 3]>
3 <split [1471/164]> Fold03 <tibble [200 x 5]> <tibble [0 x 3]>
4 <split [1471/164]> Fold04 <tibble [200 x 5]> <tibble [0 x 3]>
5 <split [1471/164]> Fold05 <tibble [200 x 5]> <tibble [0 x 3]>
6 <split [1472/163]> Fold06 <tibble [200 x 5]> <tibble [0 x 3]>
```

Exercise 3a v:

Given the following plot. Should you rather choose the best model according to the metric `rmse` or `rsq` ?



Exercise 3a vi:

Given all the different models created with the penalty grid we specified in Exercise 3a ii, how can we select the best model with respect to a given metric? It turns out that this is rather simple! Explain each of the steps below and recreate the result for the metric `rsq`.

```
glm_res_best<- glmnet_res %>%
  select_best(metric = "rmse")

best_penalty <- glm_res_best$penalty

last_glm_model <- linear_reg(penalty = best_penalty, mixture = 1)

last_glm_wflow <- glmnet_wflow %>%
  update_model(last_glm_model)

last_glm_fit <-
  last_glm_wflow %>%
  last_fit(split)
```

Et voilà, we have now created our final model `last_glm_fit` based on the best value for the penalty with respect to the `rmse` or `rsq` metric.

Exercise 3a vii:

A question that surely arises is, how we can see which of the the coefficients were set to 0 by the lasso regression.

By piping the `last_glm_fit` model to the `extract_fit_parsnip` function and piping the result to the `tidy` function, we can effectively extract the parameters for our final mode. Complete the following sequence of operations to filter for all the variables set to `0`. Which variables were set to `0`?

```
last_glm_fit %>%
  extract_fit_parsnip %>%
  tidy %>%
  #################
  ##Continue Here##
  #################
```

Exercise 3b: Ridge Regression

In this last exercise, we want to reap the fruits of our labor by easily training a ridge regression model.

Exercise 3b i:

Consider the following code snippet which is all we need for training a new model. Explain each of the following steps which are performed to train the new model.

```
model_ridge <- linear_reg(penalty = tune(), mixture = 0.0) %>%
  set_engine(engine = "glmnet", path_values = penalty)

glmnet_wflow <- glmnet_wflow %>%
  update_model(model_ridge)

glmnet_res <-
  glmnet_wflow %>%
  tune_grid(
    grid = tibble(penalty),
    metrics = multi_metric,
    resamples = folds
  )
```

Exercise 3b i:

Given the tibble `ridge_metrics`, create two plots, where you display both, the mean `rmse` and mean `rsq` of the model across all folds for different penalty values. Additionally, mark the optimal penalty value.

An example of what one of those plots could look like is below.

```
ridge_metrics <- glmnet_res %>% collect_metrics
```