Exercise Collection – Supervised Regression

Contents

Lecture exercises

Exercise 1: HRO in `mlr3`

Throughout the lecture, we will frequently use the `R` package `mlr3`, resp. the `Python` package `sklearn`, and its descendants, providing an integrated ecosystem for all common machine learning tasks. Let's recap the HRO principle and see how it is reflected in either `mlr3` or `sklearn`. An overview of the most important objects and their usage, illustrated with numerous examples, can be found at `https://mlr3book.mlr-org.com/basics.html` and `https://scikit-learn.org/stable/index.html`.

a) How are the key concepts (i.e., hypothesis space, risk and optimization) you learned about in the lecture videos implemented?

b) Have a look at `mlr3::tsk("iris")`/`from sklearn.datasets import load_iris`. What attributes does this object store?

c) Pick a module for classification or regression of your choice. What are the different settings for this learner?
(`R` Hint: use `mlr3::mlr_learners$keys()` to see all available learners.)
(`Python` Hint: Import the specific module and use `get_params()` to see all available settings.)

Solution 1:

a) Model classes representing a certain **hypothesis** are stored in `learner` objects. Before training them on actual data, they just contain information on the functional form of f. Once a learner has been trained we can examine the parameters of the resulting model. The empirical **risk** can be assessed after training by several performance measures (e.g., based on $L2$ loss). **Optimization** happens rather implicitly as `mlr3` only acts as a wrapper for existing implementations and calls package-specific optimization procedures.

b)
```r
library(mlr3)
mlr3::tsk("iris")

## <TaskClassif:iris> (150 x 5): Iris Flowers
## * Target: Species
## * Properties: multiclass
## * Features (4):
##   - dbl (4): Petal.Length, Petal.Width, Sepal.Length, Sepal.Width
```

We obtain the following information:

- `iris` is a classification task.
- It has 150 observations of 5 variables, one of which is the target.
- The target `Species` contains more than 2 classes.
- We have 4 features, all of them floating numbers (`dbl`).

If necessary, we can specify further task attributes. For example, we might have one feature that merely stores unique identifiers for each observation. `mlr3` allows us to set the *role* of this variable to an ID variable. We can also assign roles of weighting or stratifying variables in analogous fashion. Other task attributes include the number of missing values per feature and the so-called *backend* (the raw data we created our task from – besides using predefined tasks like `iris` it is possible to specify tasks from any data of suitable format).

c) Let's have a look at the available learners (in case you are wondering why this list is so short: there is a dedicated extension package, `mlr3learners`, that holds other learners besides these most basic ones, and there is even `mlr3extralearners`):

```r
mlr3::mlr_learners$keys()

## [1] "classif.debug"       "classif.featureless" "classif.rpart"
## [4] "regr.debug"          "regr.featureless"    "regr.rpart"
```

Let's check out the **regression tree** learner. Roughly speaking, regression trees create small, homogeneous subsets ("nodes") by repeatedly splitting the data at some cut-off (e.g., for `iris`: partition into observations with `Sepal.Width` ≤ 3 and > 3), and predict the mean target value within each final group.

```r
mlr3::lrn("regr.rpart")

## <LearnerRegrRpart:regr.rpart>: Regression Tree
## * Model: -
## * Parameters: xval=0
## * Packages: mlr3, rpart
## * Predict Types:  [response]
## * Feature Types: logical, integer, numeric, factor, ordered
## * Properties: importance, missings, selected_features, weights
```

We obtain the following information:

- `regr.rpart` is a *regression* learner (as you may have noticed above, there is a separate tree learner for classification).
- It has not been trained yet, so no model is stored.
- The underlying package is `rpart`.
- `regr.rpart` predicts *response* (unsurprisingly, but classification learners might also predict *probabilities*).
- It supports boolean, numerical and categorical features (but no date variables, for instance).
- Special properties include the ability to handle missing values and compute feature importance.

- Regarding hyperparameters, we see that some `xval` has been set (the function reference can be found at https://cran.r-project.org/web/packages/rpart/rpart.pdf). However, there is typically a whole bunch of configurable hyperparameters:

```
mlr3::lrn("regr.rpart")$param_set

## <ParamSet>
##                 id    class lower upper nlevels       default value
##  1:             cp ParamDbl     0     1     Inf          0.01
##  2:     keep_model ParamLgl    NA    NA       2         FALSE
##  3:     maxcompete ParamInt     0   Inf     Inf             4
##  4:       maxdepth ParamInt     1    30      30            30
##  5:    maxsurrogate ParamInt    0   Inf     Inf             5
##  6:      minbucket ParamInt     1   Inf     Inf <NoDefault[3]>
##  7:       minsplit ParamInt     1   Inf     Inf            20
##  8: surrogatestyle ParamInt     0     1       2             0
##  9:    usesurrogate ParamInt    0     2       3             2
## 10:           xval ParamInt     0   Inf     Inf            10     0
```
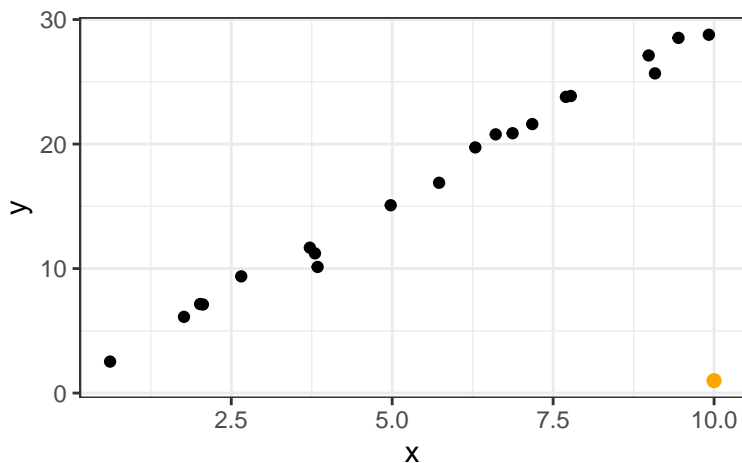
We might, for example, override the default of `minsplit`, which states the minimum number of observations a node must contain to be split further.

Exercise 2: Loss Functions

In this exercise, we will examine loss functions for regression tasks somewhat more in depth.

a) Consider the above linear regression task. How will the model parameters be affected by adding the new outlier point (orange) if you use

   i) $L1$ loss

   ii) $L2$ loss

   in the empirical risk? (You do not need to actually compute the parameter values.)
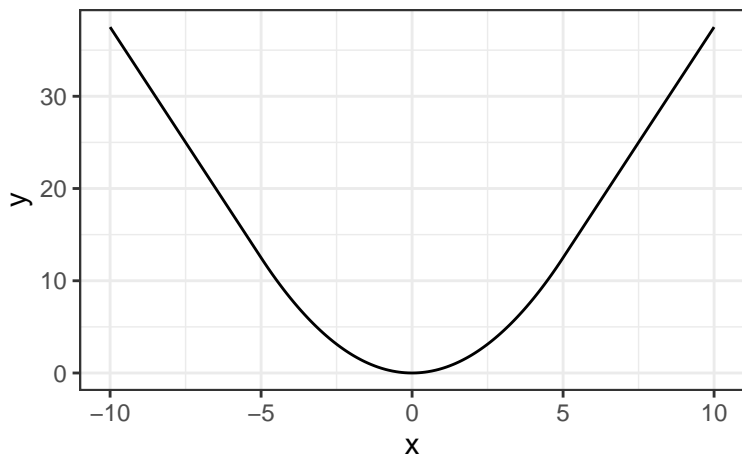
b) The second plot visualizes another loss function popular in regression tasks, the so-called *Huber loss* (depending on $\epsilon > 0$; here: $\epsilon = 5$). Describe how the Huber loss deals with residuals as compared to $L1$ and $L2$ loss. Can you guess its definition?

c) Derive the least-squares estimator, i.e., the solution to the linear model when using $L2$ loss, analytically via

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\Theta} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2.$$

and the regression line will pivot to the bottom right to minimize the resulting empirical risk. A model trained with $L1$ loss is less susceptible to the outlier and will adjust only slightly to the new data.

```
## Warning:  Using 'size' aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use 'linewidth' instead.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was generated.

## Warning:  Computation failed in 'stat_quantile()'
## Caused by error in 'compute_group()':
## !  The package "quantreg" is required for 'stat_quantile()'
```
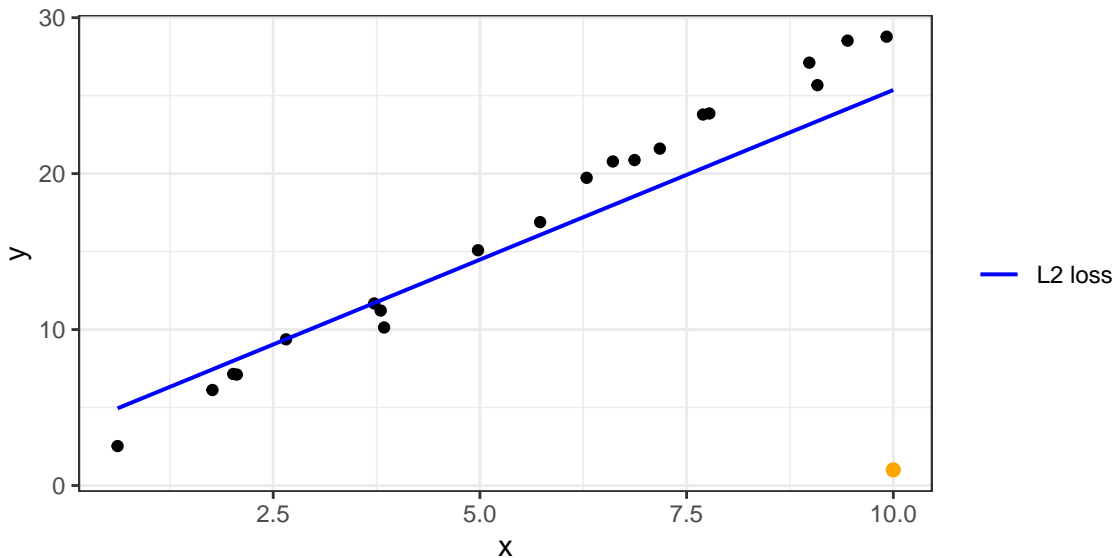

b) The Huber loss combines the respective advantages of $L1$ and $L2$ loss: it is smooth and (once) differentiable like $L2$ but does not punish larger residuals as severely, leading to more robustness. It is simply a (weighted) piecewise combination of both losses, where ϵ marks where $L2$ transits to $L1$ loss. The exact definition is:

$$L\left(y, f(\mathbf{x})\right) = \begin{cases} \frac{1}{2}(y - f(\mathbf{x}))^2 & \text{if } |y - f(\mathbf{x})| \leq \epsilon \\ \epsilon|y - f(\mathbf{x})| - \frac{1}{2}\epsilon^2 & \text{otherwise} \end{cases} , \quad \epsilon > 0$$

In the plot we can see how the parabolic shape of the loss around 0 evolves into an absolute-value function at $|y - f(\mathbf{x})| > \epsilon = 5$.

c) We solve this just like any other optimization problem: setting the derivative to 0 and solving for $\boldsymbol{\theta}$.

$$\frac{\partial}{\partial \boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2 = 0$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} \left((\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \right) = 0$$

$$\frac{\partial}{\partial \boldsymbol{\theta}} \left(\mathbf{y}^\top \mathbf{y} - 2\boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{y} + \boldsymbol{\theta}^\top \mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} \right) = 0$$

$$-2\mathbf{X}^\top \mathbf{y} + 2\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} = 0$$

$$\mathbf{X}^\top \mathbf{X}\boldsymbol{\theta} = \mathbf{X}^\top \mathbf{y}$$

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

Don't be spooked by the matrix notation – just make sure you know basic linear algebra, e.g.,

$$(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top,$$

and remind yourself of the analogies between scalar and matrix-valued calculations (e.g., x^2 translates to $\mathbf{X}^\top \mathbf{X}$, and $\frac{1}{x}$ to $\mathbf{X}^{-1}$). As this is a tool you will need to handle frequently, refresh your algebra if necessary.
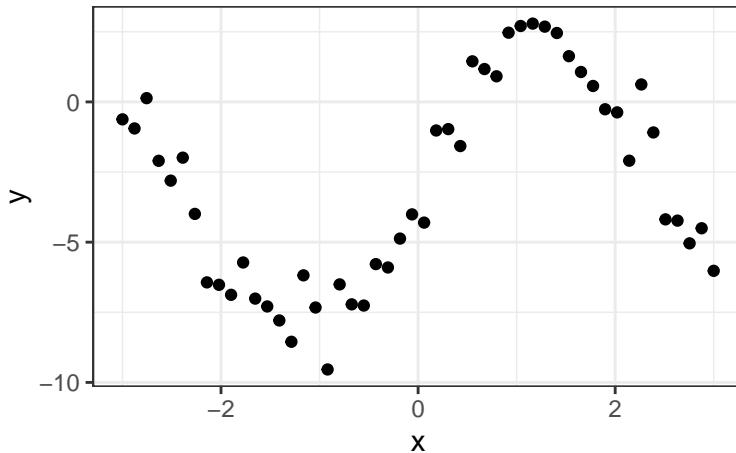
A good reference in general is "Mathematics for Machine Learning" by Deisenroth et al. (for the above, you might want to have a look at *vector calculus*), freely available at https://mml-book.github.io/book/mml-book.pdf.

Exercise 3: Polynomial Regression

Assume the following (noisy) data-generating process from which we have observed 50 realizations:

$$y = -3 + 5 \cdot \sin(0.4\pi x) + \epsilon$$

with $\epsilon \sim \mathcal{N}(0,1)$.

a) We decide to model the data with a cubic polynomial (including intercept term). State the corresponding hypothesis space.

b) Demonstrate that this hypothesis space is simply a parameterized family of curves by plotting curves for 3 different models belonging to the considered model class.

c) State the empirical risk w.r.t. $\boldsymbol{\theta}$ for a member of the hypothesis space. Use $L2$ loss and be as explicit as possible.

d) We can minimize this risk using gradient descent. In order to make this somewhat easier, we will denote the transformed feature matrix, containing x to the power from 0 to 3, by $\tilde{\mathbf{X}}$, such that we can express our model by $\tilde{\mathbf{X}}\boldsymbol{\theta}$ (note that the model is still linear in its parameters, even if $\mathbf{X}$ has been transformed in a non-linear manner!). Derive the gradient of the empirical risk w.r.t $\boldsymbol{\theta}$.

e) Using the result from d), state the calculation to update the current parameter $\boldsymbol{\theta}^{[t]}$.

f) You will not be able to fit the data perfectly with a cubic polynomial. Describe the advantages and disadvantages that a more flexible model class would have. Would you opt for a more flexible learner?

Solution 3:

a) *Cubic* means degree 3, so our hypothesis space will look as follows:

$$\mathcal{H} = \{f(\mathbf{x} \mid \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \mid (\theta_0, \theta_1, \theta_2, \theta_3)^\top \in \mathbb{R}^4\}$$

b) Choose 3 different parameterizations and plot the resulting polynomials:

```r
library(ggplot2)

# Simulate data
set.seed(123L)
x <- seq(-3, 3, length.out = 50)
y <- -3 + 5 * sin(0.4 * pi * x) + rnorm(50, sd = 1)
data <- data.frame(x, y)
```

```r
# Generate design matrix by taking x to the power of 0 through 3
X <- as.matrix(sapply(0:3, function(i) x^i))
head(X)
```

```
##      [,1]      [,2]     [,3]      [,4]
## [1,]    1 -3.000000 9.000000 -27.00000
## [2,]    1 -2.877551 8.280300 -23.82699
## [3,]    1 -2.755102 7.590587 -20.91284
## [4,]    1 -2.632653 6.930862 -18.24656
## [5,]    1 -2.510204 6.301125 -15.81711
## [6,]    1 -2.387755 5.701374 -13.61349
```

```r
# Define 3 different values for each theta_j
thetas <- matrix(cbind(
  c(-3, 1, 1.5),
  c(5, -1.6, 2),
  c(0, -0.3, -0.7),
  c(-0.8, 0.2, 0)),
  ncol = 4)
thetas
```

```
##      [,1] [,2] [,3] [,4]
## [1,] -3.0  5.0  0.0 -0.8
## [2,]  1.0 -1.6 -0.3  0.2
## [3,]  1.5  2.0 -0.7  0.0
```

```r
# Compute the resulting models
f_hat <- sapply(1:3, function(i) X %*% thetas[i, ])
data_models <- data.frame(x, f_hat)
names(data_models) <- c("x", sprintf("f_hat_%i", 1:3))
head(data_models)
```

```
##            x      f_hat_1     f_hat_2     f_hat_3
## 1 -3.000000   3.60000000 -2.30000000 -10.800000
## 2 -2.877551   1.67383318 -1.64540540 -10.051312
## 3 -2.755102  -0.04523625 -1.05158140  -9.323615
## 4 -2.632653  -1.56602096 -0.51632483  -8.616910
## 5 -2.510204  -2.89733359 -0.03743253  -7.931195
## 6 -2.387755  -4.04798681  0.38729866  -7.266472
```

```r
# Convert data to long format
data_models_long <- reshape2::melt(
  data_models,
  id.vars = "x",
  measure.vars = c("f_hat_1", "f_hat_2", "f_hat_3"))
```

```
## Error in loadNamespace(x):  es gibt kein Paket namens 'reshape2'
```

```r
# Plot the corresponding polynomial functions
ggplot2::ggplot(data_models_long, aes(x = x, y = value, col = variable)) +
  ggplot2::geom_line() +
  ggplot2::scale_color_viridis_d(
    "Models",
    end = 0.9,
    labels = c(
      bquote(-3 + 5 * x - 0.8 * x**3),
      bquote(1 - 1.6 * x - 0.3 * x**2 + 0.2 * x**3),
      bquote(1.5 + 2 * x - 0.7 * x**2))) +
  ggplot2::geom_point(data, mapping = aes(x, y), inherit.aes = FALSE) +
  ggplot2::ylab("")
```

We see that our hypothesis space is simply a family of curves. The 3 examples plotted here already hint at the amount of flexibility third-degree polynomials offer over simple linear functions.

c) The empirical risk is:

$$\mathcal{R}_{\mathrm{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{50} \left(y^{(i)} - \left[\theta_0 + \theta_1 x^{(i)} + \theta_2 \left(x^{(i)} \right)^2 + \theta_3 \left(x^{(i)} \right)^3 \right] \right)^2$$

d) Denoting our transformed feature matrix by $\tilde{\mathbf{X}}$, we can find the gradient just as we did for an intermediate result when we derived the least-squares estimator:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathcal{R}_{\mathrm{emp}}(\boldsymbol{\theta}) &= \frac{\partial}{\partial \boldsymbol{\theta}} \left\| \mathbf{y} - \tilde{\mathbf{X}} \boldsymbol{\theta} \right\|_2^2 \\
&= \frac{\partial}{\partial \boldsymbol{\theta}} \left(\left(\mathbf{y} - \tilde{\mathbf{X}} \boldsymbol{\theta} \right)^{\top} \left(\mathbf{y} - \tilde{\mathbf{X}} \boldsymbol{\theta} \right) \right) \\
&= -2 \tilde{\mathbf{X}}^{\top} \mathbf{y} + 2 \tilde{\mathbf{X}}^{\top} \tilde{\mathbf{X}} \boldsymbol{\theta} \\
&= 2 \cdot \left(-\tilde{\mathbf{X}}^{\top} \mathbf{y} + \tilde{\mathbf{X}}^{\top} \tilde{\mathbf{X}} \boldsymbol{\theta} \right)
\end{aligned}$$

e) Recall that the idea of gradient descent (*descent*!) is to traverse the risk surface in the direction of the *negative* gradient as we are in search for the minimum. Therefore, we will update our current parameter set $\boldsymbol{\theta}^{[t]}$ with the negative gradient of the current empirical risk w.r.t. $\boldsymbol{\theta}$, scaled by learning rate (or step size) α:

$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \cdot \nabla_{\boldsymbol{\theta}} \mathcal{R}_{\mathrm{emp}}(\boldsymbol{\theta}^{[t]}).$$

Note that the $L2$-induced multiplicative constant of 2 in the gradient can simply be absorbed by $\tilde{\alpha} := \frac{1}{2}\alpha$:

$$\underbrace{\boldsymbol{\theta}^{[t+1]}}_{p \times 1} = \underbrace{\boldsymbol{\theta}^{[t]}}_{p \times 1} - \tilde{\alpha} \cdot \left(-\underbrace{\tilde{\mathbf{X}}^{\top}}_{p \times n} \underbrace{\mathbf{y}}_{n \times 1} + \underbrace{\tilde{\mathbf{X}}^{\top} \tilde{\mathbf{X}}}_{p \times p} \underbrace{\boldsymbol{\theta}^{[t]}}_{p \times 1} \right)$$

$$\begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_p \end{pmatrix}^{[t+1]} = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_p \end{pmatrix}^{[t]} - \tilde{\alpha} \cdot \left(-\tilde{\mathbf{X}}^{\top} \mathbf{y} + \tilde{\mathbf{X}}^{\top} \tilde{\mathbf{X}} \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_p \end{pmatrix}^{[t]} \right)$$

What actually happens here: we update each component of our current parameter vector $\boldsymbol{\theta}^{[t]}$ in the *direction* of the negative gradient, i.e., following the steepest downward slope, and also by an *amount* that depends on the value of the gradient.

In order to see what that means it is helpful to recall that the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{R}_{\mathrm{emp}}(\boldsymbol{\theta})$ tells us about the effect (infinitesimally small) changes in $\boldsymbol{\theta}$ have on $\mathcal{R}_{\mathrm{emp}}(\boldsymbol{\theta})$. A parameter vector component with large gradient has strong impact on the empirical risk, and it is reasonable to assume it to be rather important for the model.

Therefore, gradient updates focus on influential components, and we proceed more quickly along the important dimensions.

f) We see that, for example, the first model in exercise b) fits the data fairly well but not perfectly. Choosing a more flexible function (a polynomial of higher degree or a function from an entirely different, more complex, model class) might be advantageous:

- We would be able to trace the observations more closely if our function were less smooth, and thus reduce empirical risk.

8

- Also, we might achieve a better fit on the boundaries of the input space where the cubic polynomials diverge pretty quickly.

On the other hand, flexibility also has some drawbacks:

- Flexible model classes often have more parameters, making training harder.
- We might run into a phenomenon called **overfitting**. Recall that our ultimate goal is to make predictions on *new* observations (after all, we know the labels for the training data). However, fitting every quirk of the training observations – possibly caused by imprecise measurement or other factors of randomness/error – will not generalize so well to new data.

In the end, we need to balance model fit and generalization. We will discuss the choice of hypotheses quite a lot since it is one of the most crucial design decisions in machine learning.

Exercise 4: Predicting `abalone`

We want to predict the age of an abalone using its longest shell measurement and its weight.

See `https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/` for more details.

```r
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
abalone <- read.table(url, sep = ",", row.names = NULL)
colnames(abalone) <- c(
  "sex", "longest_shell", "diameter", "height", "whole_weight",
  "shucked_weight", "visceral_weight", "shell_weight", "rings")
abalone <- abalone[, c("longest_shell", "whole_weight", "rings")]
```

```python
import pandas as pd
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data"
abalone = pd.read_csv(url, sep=",",
      names=["sex", "longest_shell", "diameter", "height", "whole_weight",
             "shucked_weight", "visceral_weight", "shell_weight", "rings"])
abalone = abalone[["longest_shell", "whole_weight", "rings"]]
```

a) Plot `LongestShell` and `WholeWeight` on the x- and y-axis, respectively, and color points according to `Rings`.

b) R: Create an `mlr3` task for the `abalone` data. Define a linear regression learner (for this you will need to load the `mlr3learners` extension package first) and use it to train a linear model on the `abalone` data.
   Python: Initiate a linear regression learner (for this you will need to import the `from sklearn.linear_model import LinearRegression` extension package first) and use it to train a linear model on the `abalone` data.

c) Compare the fitted and observed targets visually.
   (R Hint: use `autoplot()`.)
   (Python Hint: use `import matplotlib.pyplot as plt`.)

d) Assess the model's training loss in terms of MAE.
   (R Hint: losses are retrieved by calling `$score()`, which accepts different `mlr_measures`, on the prediction object.)
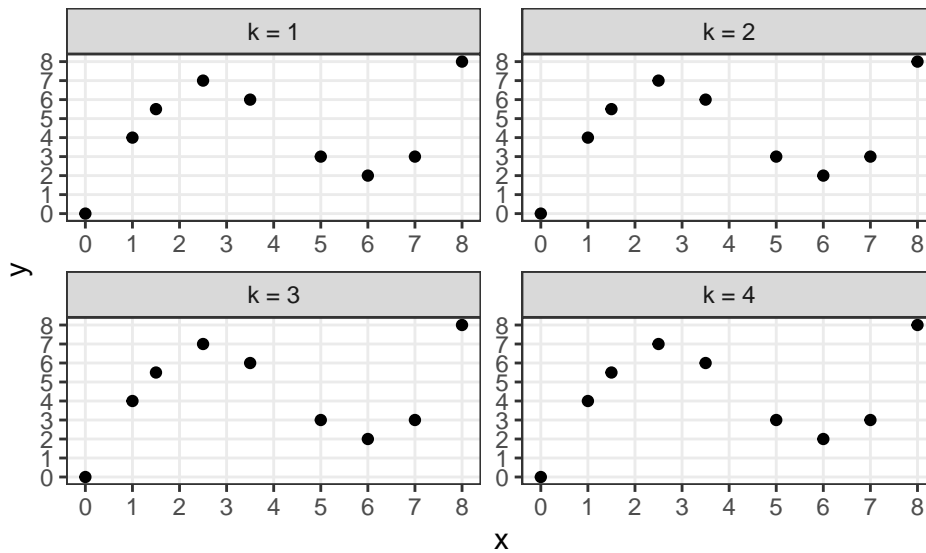   (Python Hint: The MAE metric is retrieved by calling `from sklearn.metrics import mean_absolute_error`.)

Solution 4:

See R code

Further exercises

Exercise 5: WS2020/21, second, question 1

ID	x	y
1	0.0	0.0
2	1.0	4.0
3	1.5	5.5
4	2.5	7.0
5	3.5	6.0
6	5.0	3.0
7	6.0	2.0
8	7.0	3.0
9	8.0	8.0

(a) Assume we use k-nearest neighbours regression with the L1 norm (Manhattan distance) as a distance function. For every $k \in \{1, 2, 3, 4\}$, do the following:

    (i) Mark the k-nearest neighbours of a new observation $\mathbf{x} = 4$ in the graphic below.

    (ii) Calculate the predicted value $\hat{y}$ for $\mathbf{x} = 4$ as the unweighted mean of the k-nearest neighbours and draw it in the graphic below.

(b) Would using the euclidean distance as distance measure in a) have made a difference? Explain your answer.

Solution 5:

(a) $\{x_5\}$, $\{x_5, x_6\}$, $\{x_4, x_5, x_6\}$, $\{x_4, x_5, x_6, x_7\}$

(b) No. In the case of a single feature L1 and L2 are identical.

Ideas & exercises from other sources