Basic Machine Learning with R with Toy Examples from Aquatic Ecology

Thomas Petzoldt

Table of contents

1 Introduction

The following examples aim to give an idea about analysing water quality data sets with machine learning (ML) techniques. The examples are selected to start as easy as possible and can of course not give a comprehensive overview over contemporary modelling techniques.

Problems solved with ML techniques can be sub-divided in different cateogries, supervised and non-supervised, classification and regression. The examples below concentrate exclusively on supervised regression problems. Examples for other problems can be found in textbooks like Lantz (2019) or Rhys (2020) and in many online resources.

In our research we are working with additional methods like Gradient Boosting (Friedman, 2001) and Keras (Chollet et al., 2015), but had not time yet to document didactical examples here. Additional references are found in the "Further Reading" section.

We start with a technical example and then apply the methodology to practical data sets.

2 Methods

2.1 How to work with this tutorial

The tutorial and the accompanying data are hosted as github repository. This allows to reproduce the examples in different ways:

1. Read the tutorial online, download the data to a local folder on your PC and copy and paste the code to own scripts in RStudio or another source code editor. The data sets are found at https://github.com/tpetzoldt/ml-intro/tree/main/data
2. Read the tutorial online and access the data directly from the internet without downloading. The "Dauta" data set can for example accessed via:

```
dauta <- read.csv("https://raw.githubusercontent.com/tpetzoldt/ml-intro/main/data/dauta2.csv
```

3. Clone the complete repository to a local folder and work directly with its source code. To do this, go to the repository https://github.com/tpetzoldt/ml-intro/ and use the green `<> Code` button.

2.2 Classical neural networks

In the following, let's start with one of the early packages for neural networks in **R** (R Core Team, 2022), the package **nnet** from Ripley (1996) and Venables & Ripley (2002). Later we can switch to other packages and algorithms.

The idea behind artificial neural networks (ANNs) is an analogy to biological neural networks. Neural networks consist of many simple and similar building blocks, called neurons (Figure 1) because of their analogies to their biological counterpart.

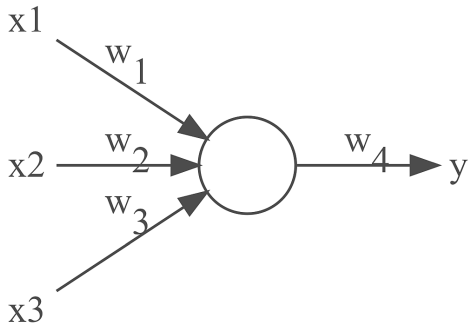

Figure 1: Neuron with 3 inputs (x1, x2, x3), weights (w1, w2, w3) and one output (y).

A neuron has several inputs x_i with individual weighting factors w_i. The weighted inputs are then combined with a weighted sum and then passed on to the output y via a transfer function.

The transfer function can have different structure, where in the classical case, a sigmoidal function was one of the most common:

$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\sum_{i=1}^{n} x_i \cdot w_i}}$$

The sigmoidal function can look very different, depending on which weights and x-values are used (Figure 2). The picture can become even more flexible, if several sigmoidal functions are added. This way, a neural network consisting of a sufficient number of neurons can be fit to any multidimensional and nonlinear data. The neurons can be connected in different topologies, e.g. a three layer feed-forward network (Figure 3).

Note: In contrast to the classical sigmoidal function, modern deep neural network frameworks like Keras prefer other transfer functions, e.g. a linear ramp function (ReLU, rectified linear unit). The ReLU is linear and surprisingly much simpler than the sigmoidal, but has several advantages. It needs less computation, can simplified to "zero" (allows sparse activation) and stabilizes training. Nonlinearity can then be introduced by additional layers. A short explanation and an overview of some relevant papers can be found in Wikipedia.

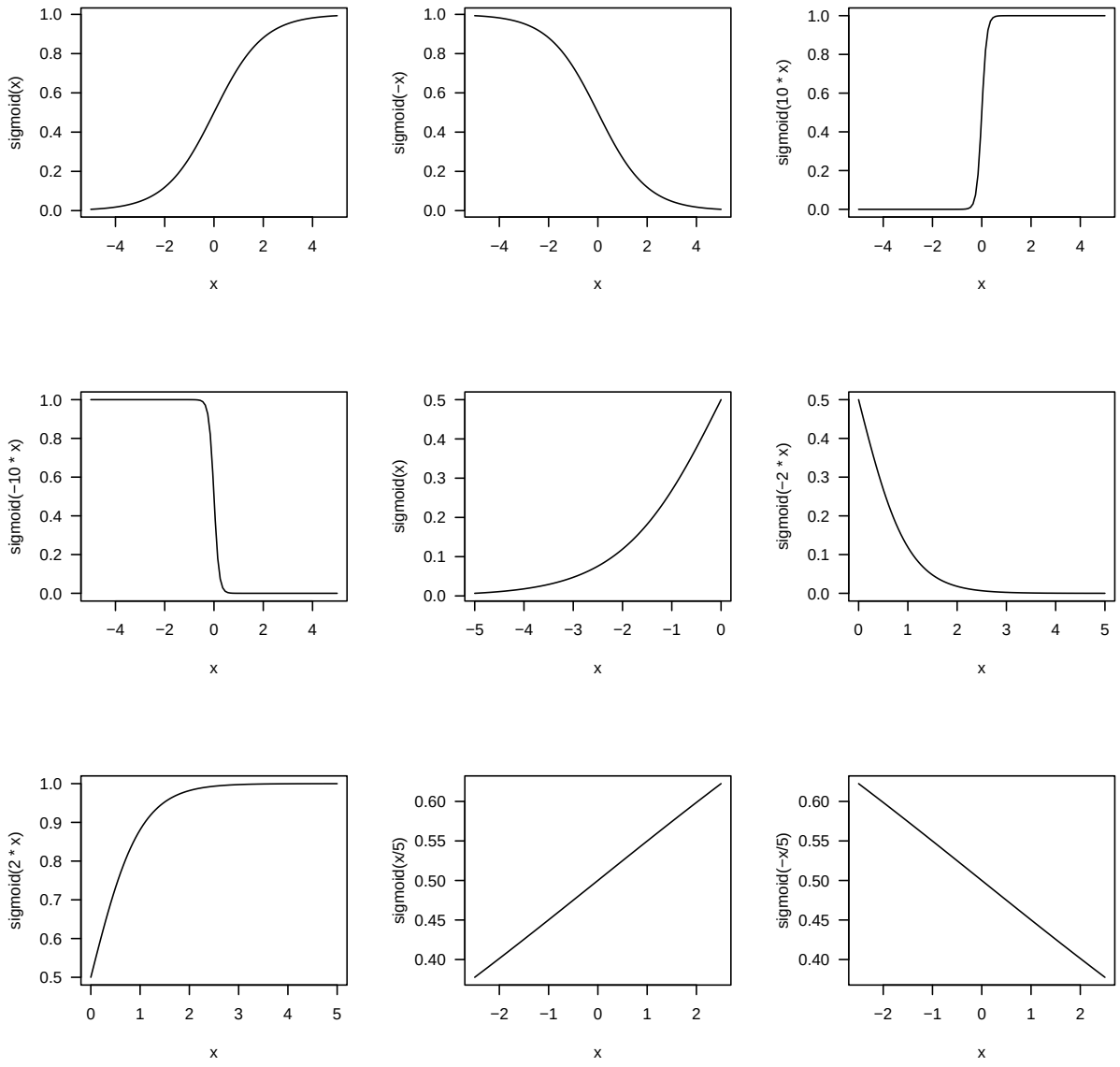

Figure 2: Flexibility of a sigmoidal transfer function. A single neuron can exhibit increasing and decreasing sigmoidal pattern, step functions, near-exponential increase and decrease, saturation or linear shapes.

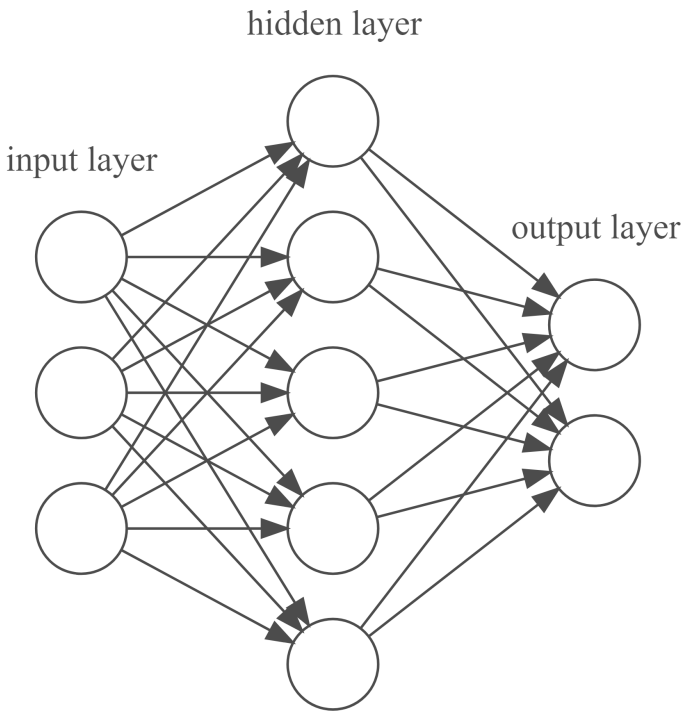

Figure 3: Fully connected feed-forward network with three layers, for example and 3 neurons in the input layer, 5 in the hidden layer and 2 in the output. Deep neural networks have more than one hidden layer.

2.3 A toy example

Let's assume a simple toy data set with a single variabble y depending on x following an optimum function with some noise (Figure 4). We can see clearly that the ANN with 5 hidden neurons can fit the data well, while one neuron was obviously not enough. But if we look closely, we see also, that the latter ANN shows some sign of overfitting at the beginning.

```
# Generate some test data
set.seed(123)
x <- seq(0, 100, 1)
y <- 100 * dlnorm(x, 4, .7) + rnorm(x, sd=0.05)

plot(x, y)
```

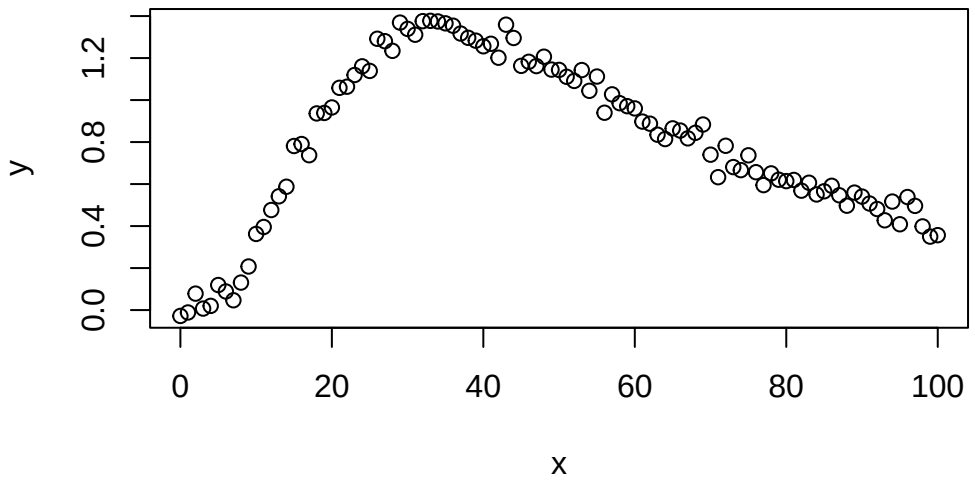

Figure 4: Toy data set.

Now, we transform the y variable to the interval [0, 1] and fit a neural network with 1 or 5 hidden neurons. The `trace` argument is set to `FALSE` to suppress intermediate results in the script. For interactive use, I would recommended to set `trace=TRUE`.

```
library(nnet)

set.seed(3142)

# Transform y-data (y must be between 0 and 1)
y <- (y - min(y))/(max(y) - min(y))

# Plot the transformed data
```

```r
plot(x, y)

# Fit the neural net
nn1 <- nnet(x, y, size=1, trace = FALSE)
nn2 <- nnet(x, y, size=5, trace = FALSE, maxit=500)

lines(x, predict(nn1), col="blue")
lines(x, predict(nn2), col="red")
```

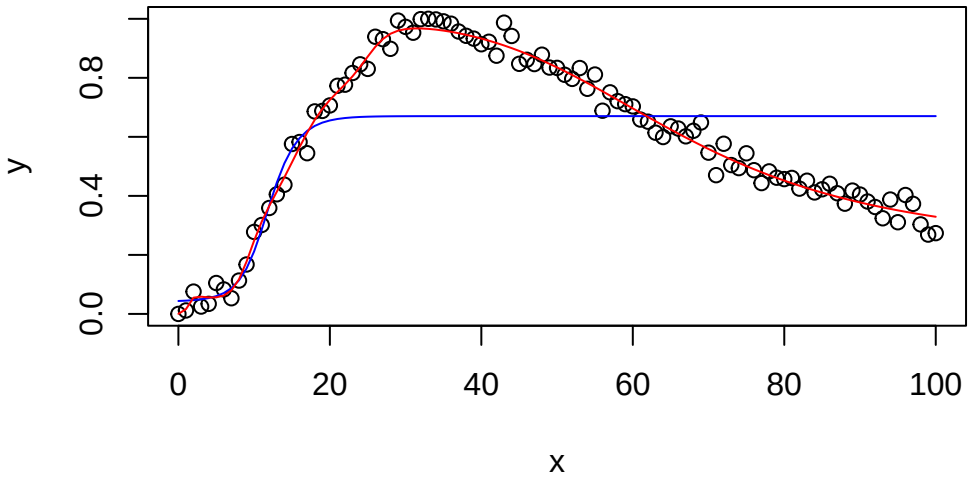

Figure 5: Toy data set scaled to [0,1] and two fitted neural networks with 1 hidden neuron (blue) and with 5 hidden neurons (red).

3 Case study 1: Dependence of phytoplankton growth rate on temperature and light

3.1 Data set

The data set was digitized from figures of Dauta et al. (1990), who analyzed growth rate dependency of four algae species on light intensity at different temperatures. Now, we aim to create a regression model to predict growth rate (μ) at any values of light and temperature within the measured range for each species. To save space, we demonstrate the procedure for only two of the four species.

The data can also be fitted with parametric models instead of ML techniques. This is explained in another tutorial.

First we load and plot the data set (Figure 6).

```
library("dplyr")
library("ggplot2")
dauta <- read.csv("data/dauta2.csv")
ggplot(data=dauta, aes(light, growthrate)) + geom_point() +
  facet_grid(species ~ temperature)
```

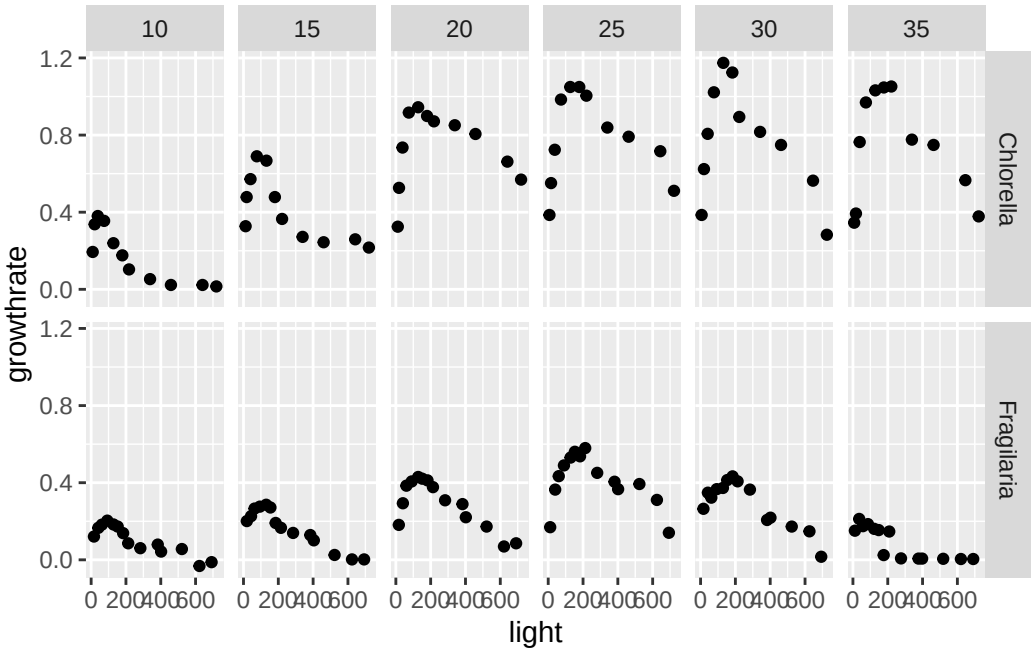

Figure 6: Growthrate (1/d) dependent on temperature (°C) and light (mol · m−2 · s−1).

For a first run, we use only the two species with the best data quality and scale the data to the interval [0, 1]:

```
dauta <- subset(dauta, species %in% c("Chlorella", "Fragilaria"))
ymin  <- min(dauta$growthrate)
ymax  <- max(dauta$growthrate)
y     <- (dauta$growthrate - ymin)/(ymax - ymin)
```

Then we encode the species from character to a numeric variable and assign the predictors (species, temperature and light) to a separate variable `x`.

```
dauta$species_i <- as.numeric(factor(dauta$species))
x <- dauta[, c("species_i", "temperature", "light")]
```

3.2 Fit of a neural network

Now we can fit a neural network. In the code below, we do this several times in a `for`-loop and select the best fitting network. This is a quite basic method, but it can lead to overfitting.

```
set.seed(1423)
hidden <- 10    # number of hidden neurons
maxit  <- 1000  # maximum number of iterations per training replicate
rep    <- 5     # number of training replicates
value <- Inf
for (i in 1:rep) {
  net <- nnet(x, y, size=hidden, maxit=maxit, trace=FALSE)
  if (net$value < value) {
    n1 <- net
    value <- net$value
  }
}
```

Now, we can compare the fitted values with the original data (Figure 7).

```
plot(y, n1$fitted.values, pch = "+",
     col = dauta$species_i, xlab = "observed", ylab = "predicted")
cat("R^2=", 1 - var(residuals(n1))/var(y), "\n") # coefficient of determination
```

```
R^2= 0.9802464
```

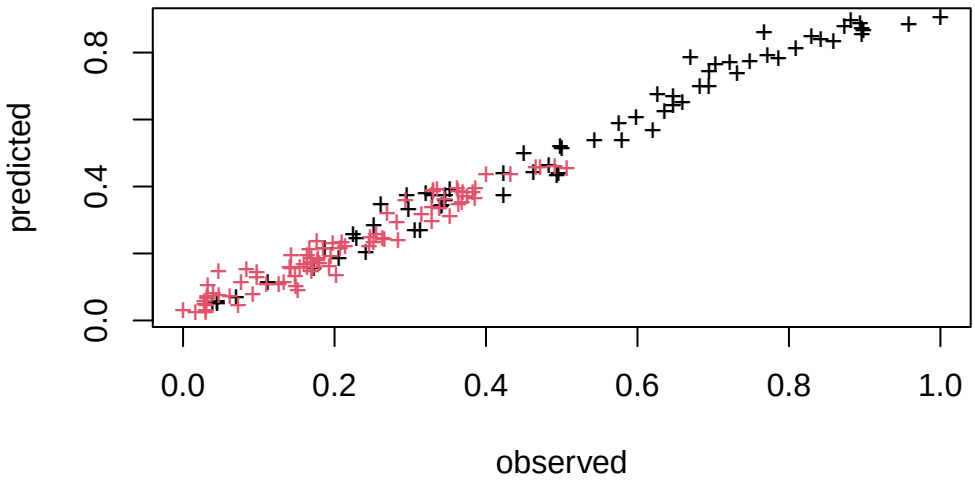

Figure 7: Comparison between predicted and observed values of the Dauta data set.

3.3 Visualization

There are of course many different ways to visualize the results. First we may compare the model with individual data sets (Figure 8).

```
## Test of neural net with a single data set
sdat <- subset(dauta, species_i == 1 & temperature == 25)

## set a series of values for the x axis
light  <- seq(0, 700, 5)
yy   <- predict(n1, data.frame(species_i = 1, temperature = 25, light = light))

## retransform growth rate from [0, 1] to original scale
growthrate   <- yy * (ymax - ymin) + ymin

plot(sdat$light, sdat$growthrate)
lines(light, growthrate, col = "red", lwd = 2)
```

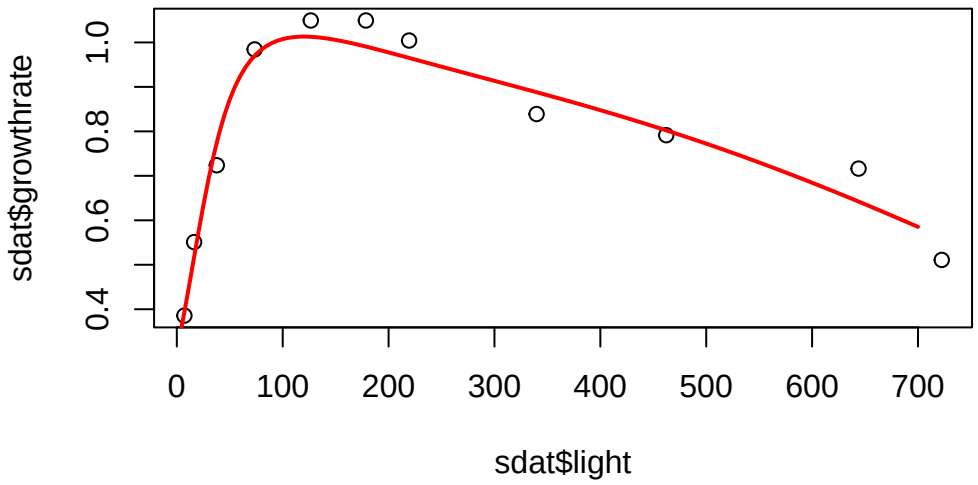

Figure 8: Comparison between observed (points) and predicted growth rates for species = 1 at 25°C.

We can also plot the outcome for all species and temperatures together (Figure 9). Here the function `expand.grid` creates a data frame with all combinations of the explanation variables `species_i`, `light` and `temperature`. This data frame can then be used in `predict` where the variable names of the new data must exactly match the variable names in the original data set used for fitting the model.

```r
newdata <- expand.grid(
  species_i = unique(dauta$species_i),
  temperature = unique(dauta$temperature),
  light = seq(0, 700, 5)
)

yy <- predict(n1, newdata)[, 1]

## retransform predicted values to original scale
newdata$growthrate <- yy * (ymax - ymin) + ymin

## assign species names corresponding to the species number
species <- levels(factor(dauta$species))
newdata$species <- species[newdata$species_i]

ggplot(data=dauta, mapping=aes(x=light, y=growthrate)) +
  geom_point() +
  geom_line(data=newdata, mapping=aes(x=light, y=growthrate), color="red", linewidth=1) +
  facet_grid(species ~ temperature)
```

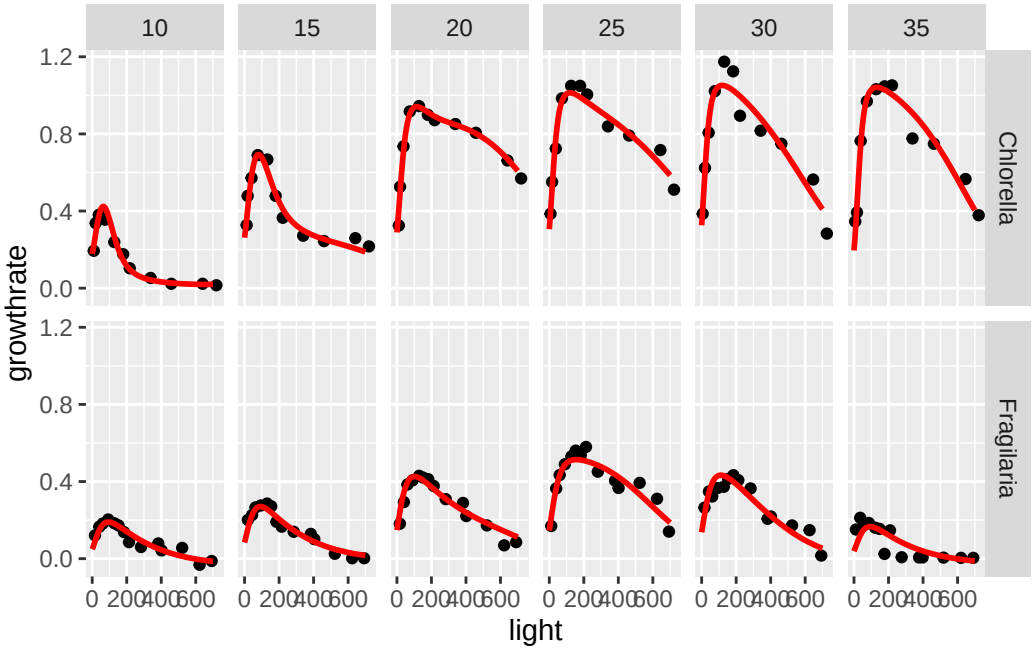

Figure 9: Comparison between observed (points) and predicted growth rates for two species.

3.4 Exercises

1. Modify the example above and test the procedure with a different number of neurons. Modify also the value in `set.seed()`. You may experience, that not all trials will work equally well. Sometimes you will see that models show additional wiggles, a clear sign of overfitting.
2. Now try to fit the whole data set with all four species. Find good settings by trial and error that produce both a high coefficient of determination and plausible figures.

4 Case Study 2: Relationship between environmental variables and phytoplankton in a lake

The example is taken from early experiments of Kielb (1996), Petzoldt (1996) and Petzoldt & Benndorf (1999) using parts of the long-term dataset from the Saidenbach Reservoir. The selection of input variables was rather explorative at that time, but should be ok for demonstration purposes.

The question was, if it is possible to reconstruct the amount of phytoplankton from environmental data. Here we have to note, that this is a pattern recognition example and no forecast, because phytoplankton and environmental data were measured at the same time.

4.1 The data set

For the simulation experiments, measured data of the Saidenbach dam were used (cf. Horn et al. (2006)). As the data were not always exactly equidistant, an interpolation was carried out with cubic splines to an interval of 14d, with subsequent manual post-correction to avoid wiggles.

The variables shown in the table, except Date and XBM were used as inputs for a neural network, while phytoplankton biomass (XBM) in the epilimnion was used as output. The DATE variable will be used to select rows. The phytoplankton biomass XBM is then square root transformed and then normalized to the interval [0, 1]. Square root transformation aims to downweight large peaks while normalizing is necessary because the standard learning algorithm requires this.

Table 1: Variables in the plankton data set.

No	Variable	Abbreviation
1	date	DATE
2	phytoplankton in epilimnion	XBE
3	lake depth	DEPTH

Table 1: Variables in the plankton data set.

No	Variable	Abbreviation
4	mixing depth	ZMIX
5	secchi depth	ST
6	temperature in epilimnion	TE
7	temperatur in hypolimnion	TH
8	conc. of PO4-P in Epilimnion	PO4_PE
9	conc. of PO4-P in hypolimnion	PO4_PH
10	conc of NO3-N in epilimnion	NO3_E
11	conc of NO3-N in hypolimnion	NO3_H
12	conc of SiO2 in epilimnion	SIE
13	conc of SiO2 in hypolimnion	SIH
14	saturation of O2 in epilimnion	O2_SATE
15	saturation of O2 in hypolimnion	O2_SATH

4.2 Application of a feed-forward neural network

4.2.1 Data Management

We start with some data management:

- convert the DATE column to a date format
- optionally, apply a square root transformation to downweight the influence of extreme values and to increase resolution for lower values.
- rescale the y variable to fit in the interval [0, 1]. In the example below, some additional "room" was added, so that minimum and maximum does not exactly match zero and one. This makes the procedure more robust, but must be taken into account for the backtransformation.
- split the data matrix to separate the target variable y and the explanation variables x

```
library(nnet)

## load datas set and convert DATE to date format
dat <- read.csv("data/phytoplankton-cqc.csv")
dat$DATE <- as.Date(dat$DATE)

## optional: transform dependent variable,
## because it contains very extreme values
dat$XBE <- dat$XBE^0.5
```

```
## rescale dependent variable to interval [0, 1]
dat$XBE <- 0.1 + dat$XBE/(1.2 * max(dat$XBE))


## split into target and explanation variables
select <- c("DEPTH", "ZMIX", "ST", "TE", "TH",
            "PO4_PE", "PO4_PH", "NO3E", "NO3H",
            "SIE", "SIH", "O2_SATE", "O2_SATH")

y <- dat$XBE
x <- dat[select]
```

4.2.2 Modell fitting

Now we can fit a first neural network. Because some networks did not converge with the default settings, some of the learning parameters (`decay` and `maxit`) were adapted by trial and error.

Note also the `set.seed()` function that starts the random number with a fixed value (the "seed") to ensure reproducibility of the examples in this document. It is a good idea to play with it in order to see how different the obtained results can be.

```
set.seed(123)

n_wts <- 15
nn <- nnet(x, y, size = n_wts, decay = 1e-3, maxit = 500, trace=FALSE)
err <- nn$value

yhat <- predict(nn)[,1]

plot(dat$DATE, dat$XBE, xlab="Date", ylab="Phytoplankton (transformed axis)")
lines(dat$DATE, yhat, col="red")
```

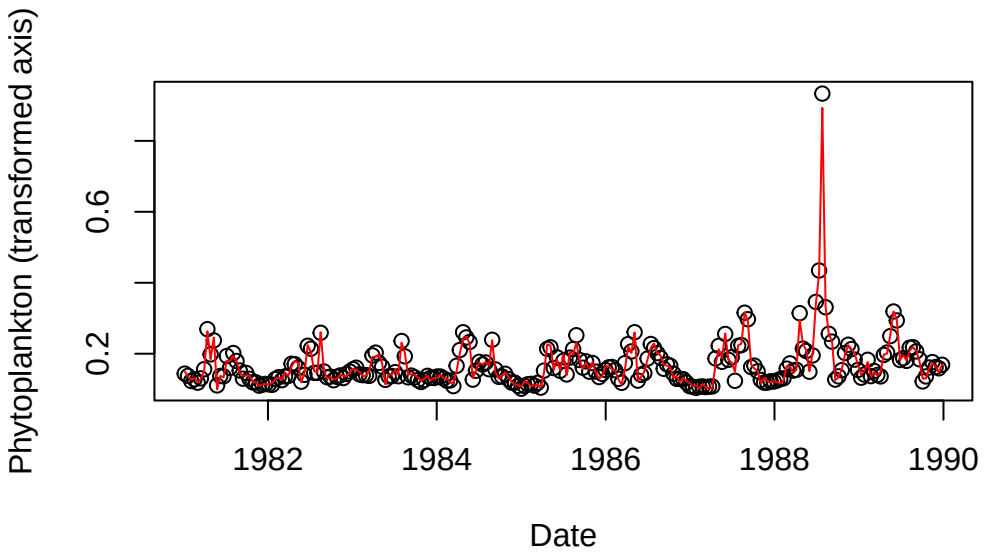

Figure 10: Measured phytoplankton data (circles, transformed scale) and fitted neural network (red line).

We can then also compute model quality criteria (cf. Jachner et al. (2007)) or plot predicted versus measured (Figure 11).

```
plot(dat$XBE, yhat, xlab="observed", ylab="predicted")
abline(a=0, b=1, col="grey", lty="dotted")
```

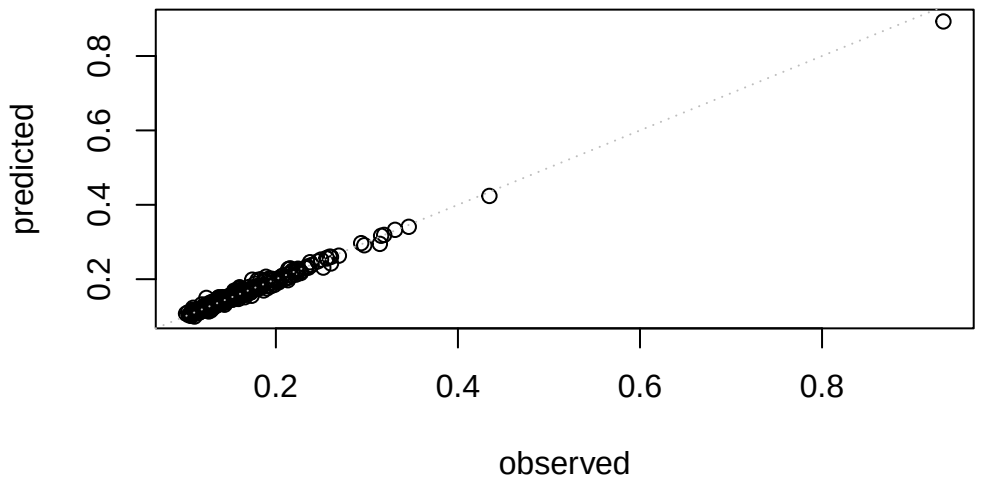

Figure 11: Neural network predictions versus measured phytoplankton data. The dotted line shows the 1:1 ratio between predicted and observed

Sometimes, it makes sense to run more than one trial and then to select the best. Note however that this can result in overfitting. We will learn later, how this can be avoided.

```
for (i in 1:10) {
  ## fit another candidate network nn_try
  nn_try <- nnet(x, y, size=n_wts, decay=1e-3, abstol=1e-6, trace = FALSE, maxit=500)

  ## and store it if better
  if (nn_try$value < err) {
    err <- nn_try$value
    nn <- nn_try
    #cat(err, "\n")
  }
}

yhat <- predict(nn)[,1]

plot(dat$DATE, dat$XBE, xlim="Date", ylim="Phytoplankton (transformed axis)")
lines(dat$DATE, yhat, col="red")
```

We can see that the network approximates the original data quite well. But as we used the full data set for training, we cannot check if the network can successfully applied to new, unkown data sets. Therefore, it is common, to split the data set at least into two data sets, where one is used for training and one for model validation.

Often, the data set is split in three subsets "trainig", "validation" and "test", where the first two are used in an automatic procedure to avoid overfitting and the 3rd "test" data set is used for an independent test of the model.

4.2.3 Exercises

1. Split the phytoplankton data set in two subsets, where years with equal number are in the training and years with unequal numbers are in the validation data set. Then fit a suitable ANN and evaluate goodnes of fit for the training data set and the validation data set.
2. Repeat the same procedure with opposite assignment to training and validation data. Compare the results.

4.3 Model fitting with mlr3

In the last years, a large number of machine learning techniques and of related **R** packages were developed, each with its own advantages and philosophy. Mastering all these and selecting the best one can be a big challenge. On the other hand, so-called "frameworks" or "meta packages" appeared, trying to integrate existing technologies in a unified application programming interface (API). These packages differ again with respect to the included features, syntax and philosophy, so it can again be difficult where to start.

One of the most popular packages in **R** is still the **caret** (Classification and Regression Training) package (Kuhn, 2022). However in the following, let's explore some possibilities of the **mlr3** package (Lang et al., 2019). This has surely a personal component, because I had the chance to meet the authors in an enthusiastic workshop, and after exploring the features of **mlr3** once again, I think it is a good decision due to its vast flexibility.

The design of the package follows a clear structure, the included algorithms are very flexible and configurable, it comes with extensive documentation and examples and is relatively easy to start with. The syntax is somewhat special due to the employed **R6** class system, but this in turn offers high flexibility and helps to ensure a clean and organized workspace.

4.3.1 Set up of a ML task

In a first step, we load the packages, the ML framework **mlr3**, a package with optional learners **ml3learners** and a package for date and time computation.

Then we load the data set and may do some data managament. First, we transform the target variable `XBM` to a reasonable scale as above and prepare 4 separate vectors `date`, `all` (all years), `train` (uneven years) and `test` (all even years) that we will need later.

Then we create a task, that contains our data set and names the target variable. To exclude `DATE` and `XBE` from the list of explanation variables, we can use `select`. This is one of the above mentioned object oriented **R6** functions. In effect, the `task`-object modifies itself without the need of an assignment with `<-` or `=`.

```
library("mlr3")
library("mlr3learners")
library("lubridate")

dat <- read.csv("data/phytoplankton-cqc.csv")
dat$XBE <- sqrt(dat$XBE/max(dat$XBE))

date <- as.Date(dat$DATE)

all  <- 1:nrow(dat)
```

```
train <- which(year(date) %% 2 == 0) # uneven years
test  <- which(year(date) %% 2 == 1) # even years

task <- as_task_regr(dat, target="XBE")

## select all columns except DATE and XBE
task$select(c("DEPTH", "ZMIX", "ST", "TE", "TH", "PO4_PE",
  "PO4_PH", "NO3E", "NO3H", "SIE", "SIH", "O2_SATE", "O2_SATH"))
```

Now we can select a learner out of a huge number of available ML algorithms. The list of available learners can be shown with.

```
mlr_learners
```

The list can be even bigger, depending on which packages were actually loaded. The learner `regr.nnet` is the one from the package **nnet** that we have used for the experiments above. For our first experiment with **mlr3**, let's use random forest algorithm (Breiman, 2001), a very powerful and robust ML algorithm, with a fast implementation in the package **ranger** (Wright & Ziegler, 2017).

If not yet done, install the package **ranger** first:

```
install.packages("ranger")
```

Now we set up a learner. Here it is possible to provide additional options, like in the `nnet` examples above. The line is shown below and can be uncommented if you want. The random forest learner is more robust and works with the default settings.

```
learner = lrn("regr.ranger")
#learner = lrn("regr.nnet", maxit=1500, decay=1e-3)
```

Now comes the essential step to train the learner. In contrast to the `nnet` example above we use only the training data subset and compare it later to the test data set, not used for training. You may wonder that the function returns very quickly. It does indeed highly CPU intensive computations, but the data set is small, our computers are fast and the "ranger" package is very efficient. We set a random seed to be able to reproduce the outcome at a later time.

```
set.seed(123)
learner$train(task, row_ids = train)
```

Now we can evaluate the results numerically:

```
print(learner$model)
```

```
Ranger result

Call:
 ranger::ranger(dependent.variable.name = task$target_names, data = task$data(),      case.w

Type:                             Regression
Number of trees:                  500
Sample size:                      104
Number of independent variables:  13
Mtry:                             3
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       0.008254066
R squared (OOB):                  0.3383215
```

The values of RMSE and R squared can be misleading, because they compare only with the training data. To get them for the test data, we first do a prediction for the test data subset and then compute the scores. Function `msr` defines a measure. The full dictionary of measures is available via:

```
mlr_measures
```

```
pred <- learner$predict(task, row_ids = test)
pred$score(list(msr("regr.mse"),
                msr("regr.rmse"),
                msr("regr.rsq")))
```

```
   regr.mse   regr.rmse         rsq
0.002552549 0.050522761 0.150215184
```

The value of r^2 may be considered surprisingly small, so let's look at it graphically (Figure 12).

To plot the results, we can make a prediction for all data, and then separately for the training (green) and the test data (red).

```
pred_all <- learner$predict(task, row_ids = all)

plot(date, pred_all$truth, pch=16, cex=0.7,
     col="navy", ylab="sqrt(XBM/max(XBM)")
lines(date, pred_all$response)
lines(date[test], pred_all$response[test], type="h", col="red")
lines(date[train], pred_all$response[train], type="h", col="green")
```

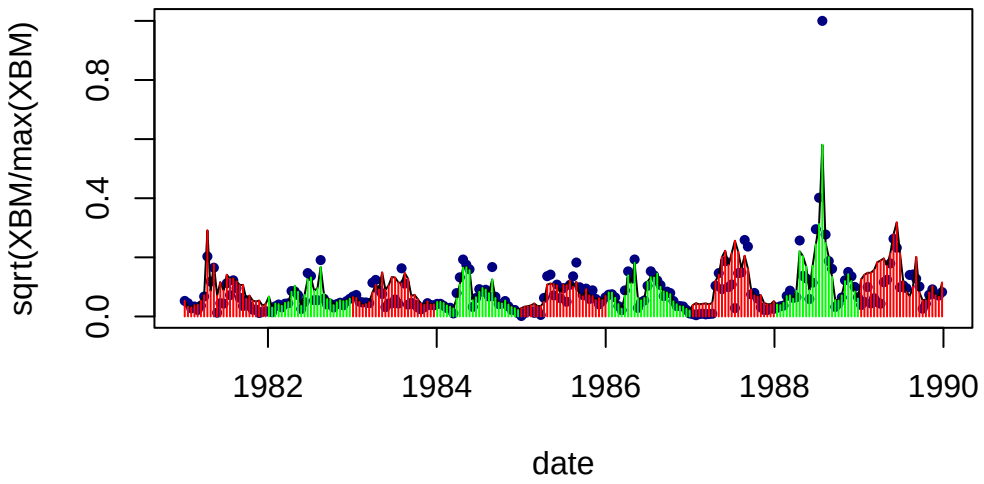

Figure 12: Measured phytoplankton data (dots, transformed scale) and fitted neural network for training (green) and test data (red).

We see that the general pattern is well matched by the model for both, the training and the test data, while the short-term behaviour differs for the test case, that's why the small r^2. However, we should not forget that this is a toy example with quite naive predictors. To improve this, we would need to consider short-term forcing.

4.3.2 Variable importance

Originally, we put all variables in that we had, but an important question of ML is, which of the predictor variables have the best predictive power and are really needed. This problem is called "variable importance" or "feature importance" and then "feature selection", that is available in package **mlr3filters**. Random forest has a nice property, to have feature importance embedded.

```
library("mlr3filters")
set.seed(123)
```

```
learner = lrn("regr.ranger", importance="impurity")
filter <- flt("importance", learner = learner)
filter$calculate(task)
as.data.table(filter)
```

```
    feature      score
     <char>      <num>
 1: O2_SATE 0.40940171
 2:      ST 0.22722108
 3:      TE 0.14492339
 4:    NO3H 0.14232261
 5:  PO4_PE 0.12669678
 6:      TH 0.07740165
 7:  PO4_PH 0.06880431
 8:    ZMIX 0.06510928
 9:    NO3E 0.06425026
10:   DEPTH 0.05720722
11: O2_SATH 0.05637095
12:     SIE 0.05052818
13:     SIH 0.03427231
```

While variable importance with `flt("importance", ...)` is available for random forest models, it cannot be applied to all learners. A list of available filters is found at https://mlr3filters.mlr-org.com/

We see that oxygen saturation (`O2_SATE`) has highest predictive power, followed by Secchi depth (`ST`) and epilimnion (suface layer) temperature `TE`. This is not surprising, as oxygen and temperature have a clear seasonal component and `ST` is inversely related to plankton. The selection of variables looks rather naive, but even in Kielb (1996) and Petzoldt (1996), this was just one of the first steps before testing suitability of neural networks for forecasting. Forecasting was not yet very successful at that time due to lack of high frequeny data. Such data are now available, so it is time to make a new start.

4.3.3 Exercise

1. Repeat the example with other learnes, e.g. neural networks (`regr.nnet`), k nearest neighbours (`regr.kknn`) or linear models (`regr.lm`). A description of the learners can be found in the mlr3 reference web page.
2. Repeat the example with a subset of predictive variables, identified by feature importance selection.
3. Do a recherche about feature importance methods and apply it to a neural network learner.

5 Outlook

The examples above should give a short introduction, so that the big world of ML techniques can be explored step by step. The upcoming mlr3 book (Kotthoff et al., 2023) covers (or will cover) additional topics like classification problems, resampling and benchmarking, hyperparameter optimization and model interpretation.

6 Further reading

- Hands-On Machine Learning with R (Boehmke & Greenwell, 2019)
- An introduction to statistical learning with applications in R (James et al., 2022)
- Deep learning with R (Chollet et al., 2022)
- Feature Engineering A-Z (Hvitfeldt, n.d.)
- TensorFlow for R (The TensorFlow Authors & RStudio, 2022)

References

Boehmke, B., & Greenwell, B. M. (2019). *Hands-on machine learning with R*. Chapman; Hall/CRC. https://bradleyboehmke.github.io/HOML/

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32. https://doi.org/10.1023/A:1010933404324

Chollet, F. et al. (2015). *Keras*. https://keras.io

Chollet, F., Kalinowski, T., & Allaire, J. J. (2022). *Deep learning with R* (Second edition). Manning Publications Co. https://learning.oreilly.com/library/view/-/9781633439849/?ar

Dauta, A., Devaux, J., Piquemal, F., & Boumnich, L. (1990). Growth rate of four algae in relation to light and temperature. *Hydrobiologia*, *207*, 221–226. https://doi.org/10.1007/BF00041459

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*(5), 1189–1232.

Horn, H., Horn, W., Paul, L., Uhlmann, D., & Röske, I. (2006). *Drei Jahrzehnte kontinuierliche Untersuchungen an der Talsperre Saidenbach – Fakten, Zusammenhänge, Trends. Abschlussbericht*. Arbeitsgruppe "Limnologie von Talsperren" der Sächsischen Akademie der Wissenschaften zu Leipzig.

Hvitfeldt, E. (n.d.). *Feature engineering a-z*. https://feaz-book.com/

Jachner, S., Boogart, K. G. van den, & Petzoldt, T. (2007). Statistical methods for the qualitative assessment of dynamic models with time delay (R package qualV). *Journal of Statistical Software*, *22*(8). https://doi.org/10.18637/jss.v022.i08

James, Gareth, Witten, D., Hastie, T., & Tibshirani, R. (2022). *An introduction to statistical learning with applications in R* (second edition). Springer.

Kielb, P. (1996). *Ein hybrides wissensbasiertes System für Umweltschutz.* (pp. 1–125) [Dissertation]. TU Dresden, Fakultät Informatik.

Kotthoff, L., Sonabend, R., Lang, M., & Bischl, B. (2023). *Flexible and robust machine learning using mlr3 in R.* https://mlr3book.mlr-org.com/

Kuhn, M. (2022). *Caret: Classification and regression training.* https://CRAN.R-project.org/package=caret

Lang, M., Binder, M., Richter, J., Schratz, P., Pfisterer, F., Coors, S., Au, Q., Casalicchio, G., Kotthoff, L., & Bischl, B. (2019). mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software.* https://doi.org/10.21105/joss.01903

Lantz, B. (2019). *Machine learning with R: Expert techniques for predictive modeling.* Packt publishing ltd.

Petzoldt, T. (1996). *Möglichkeiten zur Vorhersage von Phytoplanktonmassenentwicklungen: Von der statischen Betrachtungsweise zur Kurzfristprognose.* [Dissertation]. TU Dresden, Fakultät Forst-, Geo- und Hydrowissenschaften.

Petzoldt, T., & Benndorf, J. (1999). Modelle zur kurzzeitigen Vorhersage der Algenentwicklung in Talsperren. *Trinkwasserversorgung aus Talsperren. ATT Schriftenreihe Band 1.*

R Core Team. (2022). *R: A language and environment for statistical computing.* R Foundation for Statistical Computing. https://www.R-project.org/

Rhys, H. (2020). *Machine learning with R, the tidyverse, and mlr.* Simon and Schuster.

Ripley, B. D. (1996). *Pattern recognition and neural networks.* Cambridge University Press. https://doi.org/10.1017/CBO9780511812651

The TensorFlow Authors, & RStudio. (2022). *TensorFlow tutorials.* https://tensorflow.rstudio.com/tutorials/

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with s* (Fourth). Springer. https://www.stats.ox.ac.uk/pub/MASS4/

Wright, M. N., & Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, *77*(1), 1–17. https://doi.org/10.18637/jss.v077.i01