Machine Learning

Notes Part 1, HSG-MiQEF, Spring 2022

David Preinerstorfer

Last updated on 2022-02-22 13:37:25

Contents

Goals:

- Revise necessary notions from mathematics and probability.
- Introduce the general setting of supervised learning (using classification and regression as particular examples).
- Introduce and discuss a general method for learning decision rules in that setting: (penalized) empirical risk minimization.
- Revise some examples from previous courses in the regression setting, which are examples of (potentially

regularized) empirical risk minimizers: least squares estimation, LASSO, Ridge estimator, elastic net.

This lecture draws (to a certain extent) on Chapters 1-2 of Richter [2019] and Chapters 1-2 of Hastie et al. [2009]. The focus is on fixing notation and recalling concepts.

1 Revision of elementary notions in probability theory

In Machine Learning, we formulate our algorithms in mathematical terms and often use terminology and results from probability theory, simply because many models we work with are stochastic. Before we introduce the setup of supervised learning, we therefore need to re-activate that knowledge (and we will encounter more as we proceed); it is assumed that the terms and rules collected in this *revision section* are already familiar to you from previous courses in mathematics, statistics, econometrics or data analytics.

One important measure of centrality of a **random variable** Z is its **expectation** $\mathbb{E}(Z)$, which is linear, i.e., for real numbers a and b it holds that

$$\mathbb{E}(aZ + b) = a\mathbb{E}(Z) + b$$

(You may recall that there are random variables for which the expectation does not exist, because the integral/sum defining it does not "converge". Throughout, when working with expectations, without being too picky about that aspect, we shall assume that the quantities we operate with are well defined.). One way to measure the variability of the random variable Z around its expectation is to use its **variance** (you may check the second equality as an exercise)

$$\mathbb{V}ar(Z) := \mathbb{E}((Z - \mathbb{E}(Z))^2) = \mathbb{E}(Z^2) - [\mathbb{E}(Z)]^2$$

(the variance might not exist either, but again, when working with variances, we assume that it is well defined; recall that the symbol := reads "is defined as"). For real numbers a and b it holds that (you may check this as an exercise)

$$\mathbb{V}ar(aZ + b) = a^2\mathbb{V}ar(Z).$$

Recall that the cumulative distribution function (cdf for short) of a random variable Z is the function

$$F_Z(z) := \mathbb{P}(Z \leq z),$$

that is, the function which associates to *every* $z \in \mathbb{R}$ the probability that Z takes on values less than or equal to z. From its definition it is obvious that F_Z is non-decreasing (it is also continuous from the right and left-sided limits exist everywhere). If $Z \geq 0$, i.e., the random variable Z is non-negative, one can show that

(you may check this as an exercise)

$$\mathbb{E}(Z) = \int_0^\infty (1 - F_Z(z))dz. \tag{1}$$

In this course, we will mostly work with random variables that are either **discrete** (i.e., take on only a finite number of values) or are **continuous** (i.e., have a (Lebesgue) density). A random variable is called **Bernoulli distributed** if it can only take on two different values. Recall that a random variable Z is called **Gaussian** (or **normally distributed**) with mean $\mu \in \mathbb{R}$ and variance $\sigma^2 > 0$ if the density of Z equals

$$f_Z(z) = \frac{1}{\sqrt{2\pi\sigma}} \exp(-(z-\mu)^2/(2\sigma^2)).$$

We then write $Z_1 \sim \mathbb{N}(\mu, \sigma^2)$. Recall that f is the **density** of the random variable Z if it holds that $\mathbb{P}(Z \in A) = \int_A f(z)dz$ for every event A, i.e., probabilities of events can be computed by integrating the density over the event.[1]

Given two random variables Z_1 and Z_2, say, the (linear) association between them can be measured in terms of their **covariance**

$$\mathbb{C}ov(Z_1, Z_2) := \mathbb{E}((Z_1 - \mathbb{E}(Z_1))(Z_2 - \mathbb{E}(Z_2))) = \mathbb{E}(Z_1 Z_2) - \mathbb{E}(Z_1)\mathbb{E}(Z_2).$$

The **correlation** between Z_1 and Z_2 is obtained by dividing $\mathbb{C}ov(Z_1, Z_2)$ by $\sqrt{\mathbb{V}ar(Z_1)\mathbb{V}ar(Z_2)}$ (if the latter is nonzero). Two random variables Z_1 and Z_2 are called **independent** if

$$\mathbb{P}(Z_1 \in A, Z_2 \in B) = \mathbb{P}(Z_1 \in A) \times \mathbb{P}(Z_2 \in B)$$

holds for all events A and B (recall that under independence, the joint density of Z_1 and Z_2 equals the product of the marginal densities). Equivalently, Z_1 and Z_2 are independent, if

$$\mathbb{E}(f(Z_1)g(Z_2)) = \mathbb{E}(f(Z_1))\mathbb{E}(g(Z_2))$$

for all functions f and g. It follows from the definition (check this as an exercise!) that independent random variables Z_1 and Z_2 have covariance (and thus correlation) 0. But recall that the converse does not hold, in general.

Recall that a **random vector** $\boldsymbol{Z} = (Z_1, \ldots, Z_p)'$ (here "$'$" denotes the transposition operator, that in particular turns a row vector into a column vector) is nothing else than a tuple where every coordinate is a random variable. The **expectation of a random vector** $\boldsymbol{Z}$ is *defined as* the vector of coordinate-wise

[1] If you are wondering: I will not pay attention to measurability concerns in this course. If you do not wonder, just keep on reading ;). If you are curious, grab a book on probability theory and enjoy ...

expectations, i.e.,

$$\mathbb{E}(\boldsymbol{Z}) = (\mathbb{E}(Z_1), \ldots, \mathbb{E}(Z_p))'.$$

Furthermore, the **covariance matrix** $\mathbb{V}ar(\boldsymbol{Z})$ of the random vector $\boldsymbol{Z}$ is defined as the matrix with ij-th entry $\mathbb{C}ov(Z_i, Z_j)$, which can equivalently be written as

$$\mathbb{V}ar(\boldsymbol{Z}) := \mathbb{E}((\boldsymbol{Z} - \mathbb{E}(\boldsymbol{Z}))(\boldsymbol{Z} - \mathbb{E}(\boldsymbol{Z}))'). \tag{2}$$

It is not difficult to verify that for an $m \times p$ dimensional matrix $\boldsymbol{A}$ of real numbers, and m-dimensional vectors of real numbers $\boldsymbol{a}$ and $\boldsymbol{b}$, the **following rules** apply (you may check this as an exercise)

- $\mathbb{E}(\boldsymbol{AZ} + \boldsymbol{a}) = \boldsymbol{A}\mathbb{E}(\boldsymbol{Z}) + \boldsymbol{a}$
- $\mathbb{V}ar(\boldsymbol{AZ} + a) = \boldsymbol{A}\mathbb{V}ar(\boldsymbol{Z})\boldsymbol{A'}$

A random vector $\boldsymbol{Z}$ is called multivariate Gaussian with mean $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_p)'$ and covariance matrix $\boldsymbol{\Sigma}$ if its multivariate density equals

$$f_{\boldsymbol{Z}}(z) = (2\pi)^{-n/2}\det(\boldsymbol{\Sigma})^{-1/2}e^{-(\boldsymbol{z}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\boldsymbol{z}-\boldsymbol{\mu})}. \tag{3}$$

Here $\det(\boldsymbol{\Sigma})$ denotes the determinant of $\boldsymbol{\Sigma}$ (which is assumed to be non-zero). Recall that the random vector $\boldsymbol{Z}$ has density f if for every event $A \subseteq \mathbb{R}^p$ it holds that $\mathbb{P}(\boldsymbol{Z} \in A) = \int_A f(\boldsymbol{z})d\boldsymbol{z}$.

For random vectors $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$ the **conditional expectation** of $\boldsymbol{Z}_1$ given $\boldsymbol{Z}_2 = \boldsymbol{z}_2$ is denoted as $\mathbb{E}(\boldsymbol{Z}_1|\boldsymbol{Z}_2 = \boldsymbol{z}_2)$. Note that the conditional expectation is a function of $\boldsymbol{z}_2$ (that is, depending on the value of $\boldsymbol{z}_2$ we can expect $\boldsymbol{Z}_1$ on average to equal $\mathbb{E}(\boldsymbol{Z}_1|\boldsymbol{Z}_2 = \boldsymbol{z}_2)$). Recall the **iterated law of conditional expectations**

$$\mathbb{E}(\boldsymbol{Z}_1) = \mathbb{E}(\mathbb{E}(\boldsymbol{Z}_1|\boldsymbol{Z}_2 = \boldsymbol{z}_2)),$$

where the outer expectation is taken w.r.t. $\boldsymbol{Z}_2$. This means, that we can compute expectations in a two-step manner. First, we can treat $\boldsymbol{Z}_2$ as fixed, then we average out $\boldsymbol{Z}_2$. This oftentimes turns out to be convenient. For functions f and g we have

$$\mathbb{E}(f(\boldsymbol{Z}_2)g(\boldsymbol{Z}_1)|\boldsymbol{Z}_2 = \boldsymbol{z}_2) = f(\boldsymbol{z}_2)\mathbb{E}(g(\boldsymbol{Z}_1)|\boldsymbol{Z}_2 = \boldsymbol{z}_2),$$

that is, we can "pull out" functions of the random vector we condition on (intuitively this is clear as we treat everything we condition on as fixed).

Finally, we recall some convergence statements. Let Z_n for $n = 1, 2, 3, \ldots$ be a sequence of random variables (or vectors). The sequence Z_n **converges in probability** to the random variable (or vector) Z, say, if $\mathbb{P}(\|Z_n - Z\| \geq \varepsilon) \to 0$ holds for every $\varepsilon > 0$ (here $\|.\|$ denotes the Euclidean norm which coincides with the absolute value in dimension 1). The sequence Z_n **converges in distribution** to Z if the sequence of cdfs $F_{Z_n}(z) \to F_Z(z)$ for every $z \in \mathbb{R}$, where F_Z denotes the cdf of Z, which we assume to be continuous (typically it will be a standard normal distribution). The most important statements for us are (i) the **law of large numbers**, which states that if the Z_n are i.i.d. random variables with mean μ, it holds that

$$\frac{1}{n}\sum_{i=1}^{n} Z_i \to \mu \quad \text{in probability} \quad [LLN];$$

and (ii) the **central limit theorem**, which states that if the Z_n are i.i.d. random variables with mean μ and variance $\sigma^2 > 0$, then

$$\frac{1}{\sqrt{n}}\sum_{i=1}^{n} (Z_i - \mu) \to \mathbb{N}(0, \sigma^2) \quad \text{in distribution} \quad [CLT].$$

with mean μ.

One last **remark**: Throughout this course we write upper case letters for random variables (e.g., Z), vectors are written in bold ($\boldsymbol{z}$), and realizations are written in lower case letters (e.g., z is a realization of the random variable Z or $\boldsymbol{z}$ is a realization of $\boldsymbol{Z}$).

2 A framework for supervised learning

2.1 Deterministic vs. stochastic models

The main goal in supervised learning is to learn about the relationship between a set of **features**, also called **inputs**, and an **outcome**. Classic examples would be

- predicting a stock price (output) or economic indicators (e.g., inflation rate) as a function of previous prices and the past prices of other stocks or economic/news indicators;
- predict whether an e-mail is spam (output, “yes” or “no”) based on characteristics of the mail text, such as the number of times the name of the recipient is mentioned or the occurrence of a dollar symbol);
- fraud detection, e.g., tax evasion (output, “yes” or “no”) based on characteristics in the tax report;
- handwritten number recognition based on pixel information from a scan;
- emotion recognition based on facial expressions;
- face recognition; etc.

That is, we try to come up with a good predictor of the outcome based on the set of features available. We are not primarily interested in building a causal model; if a model predicts well, we are happy. Machine learning provides a set of tools that apply to a wide variety of settings. In order to formulate these tools in the most efficient way, we usually describe them formally, and abstract away from the specific application.

Notationally, the features are typically denoted by

$$\boldsymbol{X} = (X_1, \ldots, X_p)'$$

and the outcome is denoted by Y. Without much loss of generality, we will typically assume that the p features are real numbers (potentially taking on only discretely many different value) and that also the outcome is a real number.

The relationship between the features $\boldsymbol{X}$ and the outcome Y we would ideally like to know is a function

$$f^* : \mathbb{R}^p \to \mathbb{R}$$

that maps $\boldsymbol{X} \mapsto f^*(\boldsymbol{X}) = Y$. The function f^* would then encode the **deterministic relationship** between the predictors and the output. In most situations[2] working with deterministic models (which are immediately **falsified** by a single observation for which $Y \neq f^*(X)$) is too restrictive, and we therefore give ourselves some room and assume a **stochastic model**, which describes how the outcome behaves "on average" given the predictor. That is, we are interested in properties of the conditional distribution of Y given $\boldsymbol{X}$. From now on, we therefore interpret the predictors $\boldsymbol{X}$ as well as the outcome Y as *random variables.*

In this interpretation, $(Y, \boldsymbol{X}')'$ (which we write $(Y, \boldsymbol{X})$ whenever this does not cause confusion) is a $p + 1$ dimensional random vector, which therefore has a distribution on $\mathbb{R} \times \mathbb{R}^p$. As pointed out above, we will mostly be interested in properties of the conditional distribution of Y given $\boldsymbol{X}$, as this summarizes all information about Y given we observed $\boldsymbol{X} = \boldsymbol{x}$. Note that assuming a stochastic model does per se not rule out a deterministic relationship between $\boldsymbol{X}$ and Y. A deterministic relationship is obtained as a special case of a stochastic model whenever the conditional distribution of Y given $\boldsymbol{X} = \boldsymbol{x}$ is concentrated at a single point (and thus has 0 variance), which we may then call $f^*(x)$. In this sense, working with stochastic models is more general, and indeed gives us a higher level of flexibility.

The joint distribution of Y and $\boldsymbol{X}$ describes the regularities of the relationship between the outcome and the predictor. If we *knew* this distribution, in order to solve any decision problem, we would not need any data and could derive solutions to prediction problems theoretically from our knowledge of that distribution. However, the distribution of $(Y, \boldsymbol{X})$ is rarely known. The task therefore is to solve problems concerning the relationship between $\boldsymbol{X}$ and Y by inferring the relevant aspects of that distribution from data to solve the problem under study.

[2]Exceptions can be found in the natural sciences, e.g., classical mechanics.

2.2 Training sample

Learning from data necessitates that we are given what is called a **training set** of data, also referred to as the **training sample**. We interpret the training sample as **independent and identically distributed** (i.i.d.) copies of $(Y, \boldsymbol{X})$, which we write as

$$(Y_i, \boldsymbol{X}_i) \quad \text{for} \quad i = 1, \ldots, n,$$

where n denotes the size of the training set. The underlying assumption is that (i) the training data is representative of the phenomenon under study (i.e., we want to avoid biases that arose through mistakes in the sampling process); and (ii) that the observations are (at least approximately) independent of each other. The second assumption is mostly for convenience. There exist results and methods also for many situations with dependence, but typically this requires taking the dependence into account, which adds another layer of complexity to the problem we would like to avoid.

Note that whenever one comes up with an algorithm, this algorithm can be applied to data *regardless* of whether we view them as being generated from an i.i.d. model or not. Assumptions on the joint distribution of $(Y, \boldsymbol{X})$ and the training sample are mainly imposed to derive algorithms prove that the algorithm has certain guarantees under certain conditions (which we can never be sure of being satisfied in the first place). This then tells us when one algorithm should be preferred to another one.

2.3 Statistical decision theory: decision rule, loss, risk and optimality

We now introduce the decision theoretic framework of a supervised learning problem. For unsupervised learning or reinforcement learning, similar notions are used and we will discuss them when needed.

The theoretical framework for deriving and analyzing algorithms for supervised learning problems incorporates three ingredients:

1. A **decision rule** which associates to every $\boldsymbol{x} \in \mathbb{R}^p$ an outcome $y \in \mathbb{R}$. Formally, a decision rule is nothing else than a function $f : \mathbb{R}^p \to \mathbb{R}$.
2. A **loss function**, which helps us to formalize how "good" a decision is. This function associates to every pair of numbers y (output) and s (prediction) a non-negative number, i.e., the **loss** $L(y, s)$. Here, we interpret y as the "true outcome" and $s = f(\boldsymbol{x})$ as the "outcome predicted by a decision rule" f. Then, $L(y, f(\boldsymbol{x}))$ describes the loss one makes in guessing $s = f(\boldsymbol{x})$ but observing y. Formally, the loss function L is a function from $\mathbb{R} \times \mathbb{R}$ to the non-negative real numbers, that is,

$$L : \mathbb{R} \times \mathbb{R} \to [0, \infty).$$

We'll see some examples below; in any case you should interpret the loss function as a way of measuring the "distance" between y and $f(x)$ that is appropriate in the specific context under study.

3. The **risk** of a decision rule f, which we write as $\mathsf{R}(f)$, is its expected loss, i.e.,

$$\mathsf{R}(f) = \mathbb{E}\left(L(Y, f(\boldsymbol{X}))\right).$$

   Note that as the expectation of a non-negative function the risk is always non-negative. Furthermore, note that the risk of a decision rule f depends on the underlying distribution of $\boldsymbol{X}$ and Y. Hence, we already see that something like an "optimal" decision rule (which we shall think about next) will most likely depend on that distribution (unless the decision problem is trivial).

The general idea is that a decision rule with "small" risk is desirable, whereas a decision rule with "large" risk needs to be avoided.

The precise loss function one uses depends on the context. The following two situations, the first being called a **regression problem**, the second a **classification problem**, are the most important examples we shall encounter (many other specifications being used in the specialized literature):

- If the outcome Y takes on values in all of $\mathbb{R}$, one often uses $L(y, s) = (y - s)^2$. This is called **quadratic loss function** and should be familiar to you from linear regression problems.
- If the outcome Y is known to only take on a finite number of values $\{1, \ldots, K\} \subseteq \mathbb{R}$, say, we may want to use a loss function that is 0 if $y = s$ and 1 otherwise. In other words, there is no loss if we predict correctly, and if we predict incorrectly, the loss is the same, regardless of what our specific prediction was: we penalize all errors equally ("all errors are equally bad"). Formally, the loss function we use in such problems can be written as

$$L(y, s) = \begin{cases} 1 & \text{if } y \neq s \\ 0 & \text{else.} \end{cases}$$

Now that we have a notion to measure the quality of a decision rule, we can go on to determine **optimal decision rules**. Given a loss function L, a decision rule f^* is called **optimal** if for any other decision rule f it holds that

$$\mathsf{R}(f^*) \leq \mathsf{R}(f).$$

Note that such optimal decision rules will depend on the specific distribution of $(Y, \boldsymbol{X})$ (because the risk depends on that distribution) and (even though we may be able to write them down in dependence on this distribution theoretically) they are not directly available to the practitioner who does *not* know the distribution of $(Y, \boldsymbol{X})$. However, deriving optimal decision rules will be the first step to construct algorithms that provide data-driven approximations to such "optimal" decision rules.

2.3.1 Optimal decision rule in the regression problem

Let's first take a look at the **regression problem**, i.e., when the outcome Y takes on values in $\mathbb{R}$ (i.e., not only in a finite subset). We need a lemma:

Lemma 1. *Let Z be a random variable. Then, its expectation $\mathbb{E}(Z)$ minimizes the function $a \mapsto \mathbb{E}((Z-a)^2)$.*

Proof. Let a be a real number and write

$$\mathbb{E}((Z-a)^2) = \mathbb{E}(((Z-\mathbb{E}(Z))+(\mathbb{E}(Z)-a))^2) = \mathbb{E}((Z-\mathbb{E}(Z))^2)+2\mathbb{E}((Z-\mathbb{E}(Z))(\mathbb{E}(Z)-a))+\mathbb{E}((\mathbb{E}(Z)-a)^2).$$

The second term to the far-right equals 0 (why?). Furthermore

$$\mathbb{E}((Z-\mathbb{E}(Z))^2)+\mathbb{E}((\mathbb{E}(Z)-a)^2) \geq \mathbb{E}((Z-\mathbb{E}(Z))^2$$

with equality if and only if $a = \mathbb{E}(Z)$. [Alternatively, expand $\mathbb{E}((Z-a)^2)$, take derivatives and set equal to 0.] □

Applying this lemma to the conditional distribution of Y given $\boldsymbol{X}$ shows that the conditional expectation $\mathbb{E}(Y|\boldsymbol{X}=\boldsymbol{x})$ minimizes $\mathbb{E}((Y-a)^2|\boldsymbol{X}=\boldsymbol{x})$ for every $\boldsymbol{x}$. It hence follows, that if L is the quadratic loss function, the risk of a decision rule $f : \mathbb{R}^p \to \mathbb{R}$ satisfies

$$\mathsf{R}(f) = \mathbb{E}_{Y,\boldsymbol{X}}(L(Y, f(\boldsymbol{X}))) = \mathbb{E}_{Y,\boldsymbol{X}}((Y-f(\boldsymbol{X}))^2) \geq \mathbb{E}_{Y,\boldsymbol{X}}((Y-\mathbb{E}(Y|\boldsymbol{X}=\boldsymbol{x}))^2).$$

In other words, the decision rule $\boldsymbol{x} \mapsto f^*(\boldsymbol{x}) = \mathbb{E}(Y|\boldsymbol{X}=\boldsymbol{x})$ is optimal.

2.3.2 Optimal decision rule in the classification problem

Next, let's consider the **classification problem**, i.e., when the outcome Y takes on values in a finite set $\{1,\ldots,K\}$. Here, $L(Y, f(X))$ equals 1 if $Y \neq f(X)$ and is zero else. Hence, $L(Y, f(X)) = 1$ if and only if $Y \neq f(X)$. Therefore, the risk of a decision rule $f : \mathbb{R}^p \to \{1,\ldots,K\}$ is given by (using the iterated law of conditional expectations)

$$\mathsf{R}(f) = \mathbb{E}(L(Y, f(\boldsymbol{X}))) = \mathbb{E}_{\boldsymbol{X}}(\mathbb{E}(L(Y, f(\boldsymbol{X}))|\boldsymbol{X}=\boldsymbol{x})) = \mathbb{E}_{\boldsymbol{X}}(\mathbb{P}(Y \neq f(X)|\boldsymbol{X}=\boldsymbol{x})).$$

From this expression it is clear that the decision rule

$$f^*(\boldsymbol{x}) \in \arg\min_{i=1,\ldots,K} \mathbb{P}(Y \neq i|\boldsymbol{X}=\boldsymbol{x}) = \arg\max_{i=1,\ldots,K} \mathbb{P}(Y = i|\boldsymbol{X}=\boldsymbol{x})$$

is optimal. In words, given $\boldsymbol{X} = \boldsymbol{x}$ this decision rule returns that class i, say, which maximizes the conditional probability that Y is in that class, given the information provided through $\boldsymbol{x}$. This is a very intuitive rule, which is called the **Bayes classifier** or **Bayes rule**. Note that the maximum may not be unique (the maximal probabilities could be the same for more than one class).

2.4 Data-driven learning of decision rules approximating optimal decision functions

We have seen how optimal decision rules look like in the regression and the classification problem. In both cases, the optimal decision rules depend on the unknown distribution of $(Y, \boldsymbol{X})$.

- For the regression problem, the optimal decision rule is $\boldsymbol{x} \mapsto \mathbb{E}(Y|\boldsymbol{X} = \boldsymbol{x})$. Note that we do not know this conditional expectation.
- For the classification problem, the optimal decision rule is $\boldsymbol{x} \mapsto \arg\max_{i=1,\ldots,K} \mathbb{P}(Y = i|\boldsymbol{X} = \boldsymbol{x})$, but, again, we do not know these probabilities.

To overcome this problem, we obviously want to exploit the training sample $(Y_i, \boldsymbol{X}_i)$ for $i = 1, \ldots, n$, and use this sample to learn a decision rule $\hat{f}_n$, say, that hopefully is *close* to the optimal one at least "on average". The "hat" here emphasizes that the decision rule $\hat{f}_n$ is obtained as a function of the training sample $(Y_i, \boldsymbol{X}_i)$ for $i = 1, \ldots, n$. That is, we use the training sample to construct a good decision rule in a data-driven way. This is necessary, because the training sample allows us to learn about the distribution of $(Y, \boldsymbol{X})$. Note also that the subscript shows that this decision rule we learned depends on a training sample of size n. The larger n, the more we can learn (however data may be costly).

There are two questions that need our attention:

- How can we actually learn a decision rule from data? Is there a "generic approach" to do so; in particular one that does not exploit particularities of the loss function we work with? Or do we need to obtain ad-hoc solutions for every setup, i.e., every loss function?
- How can we evaluate the decision rule we learned, once we know how to answer the first question?

2.4.1 How can we learn a decision rule? Some intuition.

Consider a regression problem. Here, we have a training sample $(Y_i, \boldsymbol{X}_i)$ for $i = 1, \ldots, n$ and we know that the optimal decision rule is $\boldsymbol{x} \mapsto \mathbb{E}(Y|\boldsymbol{X} = \boldsymbol{x})$. That is, based on the training sample, we need to try figure out what this conditional expectation precisely is. In the simplest case, there is only one feature, i.e., $p = 1$. Then, we can plot the training sample in the form of a scatterplot. One hypothetical example is the training sample depicted in Figure 1, where the true underlying model is $Y = X + 5X^2 + U$ with the error $U \sim N(0, 9)$ independent of X, so that the optimal decision rule (which we do not know) is $\mathbb{E}(Y|\boldsymbol{X} = \boldsymbol{x}) = \boldsymbol{x} + 5\boldsymbol{x}^2$, which is shown in the plot in blue

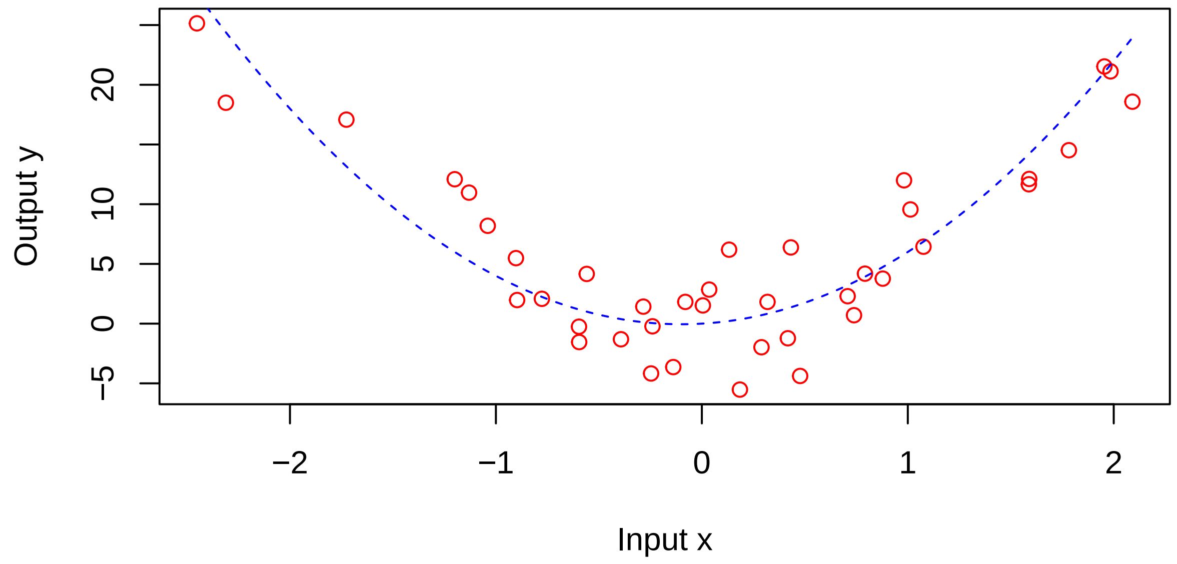

Figure 1: Scatterplot of the training sample in case $p = 1$.

Learning a decision rule now amounts to figuring out which function “fits” the data best. One naive approach could be to just “interpolate” the data points, i.e., take as the inferred decision rule the orange line in Figure 2. In that figure, the decision rule in orange is a step function that “interpolates” the given data points. It is clear from that figure that the function shown in orange does not make any error on the training sample. To every point X_i the fitted value $\hat{f}_n(X_i) = Y_i$. It is nevertheless clear as well that there is a strong **overfit**. This stems from the fact that we somewhat ignore that the model generating the data is stochastic. We should perhaps take into account that there are random fluctuations or deviations from the average behavior (the conditional expectation which we would actually like to target), which we should ideally neglect in the modeling process. Therefore, we have to **constrain the class of decision rules** we work with!

What happens if we constrain this class to severely? If we fit a linear regression model (as usual through the ordinary least squares estimator, OLS for short) we would obtain the decision rule which is shown in Figure 2. From this figure we see that there is remaining structure in the data that is not captured by the simple linear model. From our knowledge of the true optimal decision rule this is clear, since it is a quadratic polynomial.

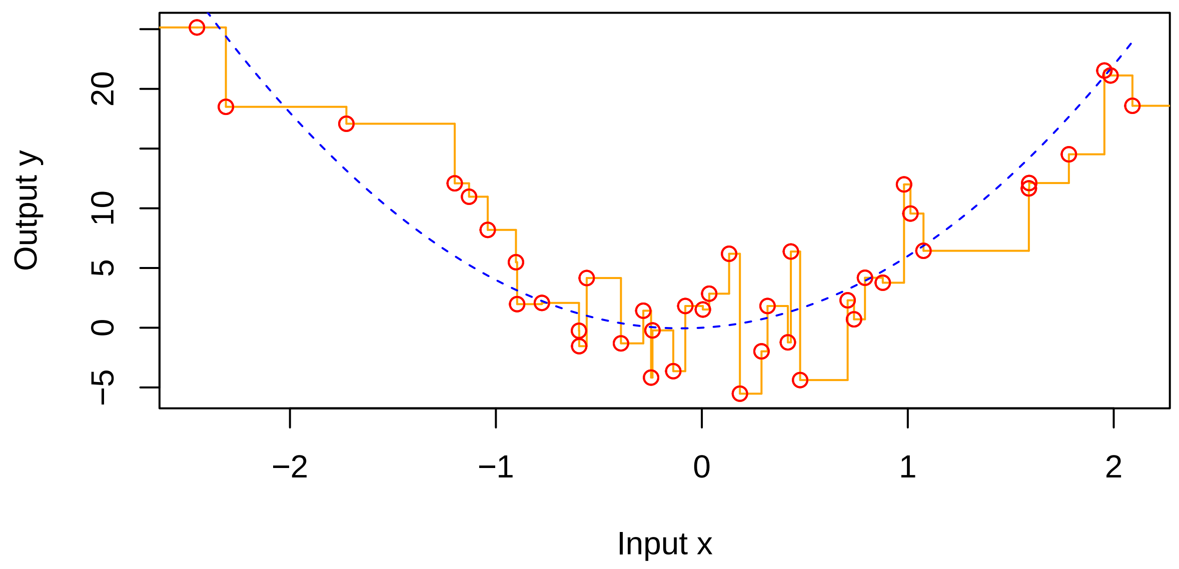

Figure 2: Interpolating the data results in an enormous overfit.

Hence, the second order term is just missing, which results in a poor approximation. Of course, this is just purely hypothetical as in reality we do not know the true optimal decision function, but need to guess it from the data. But as an example, this provides some illustration into what is going on behind the scenes.

Summarizing: While we have to constrain the family of decision rules we work with, we should not constrain them too severely.

We have now seen that if we fit a decision rule to the data that is very complex, this may result in an overfit, while if we fit a decision rule to the data that is too simplistic, this results in underfit. In any case, we need to think about the problem at hand, and select some set of decision rules we want to learn the best from. The general situation is that we have a set of "candidate" decision rules, which we shall call a **model** and which in the present context is nothing else than a set of functions $\mathcal{F}$. Three examples:

- The set $\mathcal{F}$ could be the set of functions of the form $x \mapsto a + bx$, i.e., the set of all linear (better: affine) functions, which we fit in the second example above by OLS, and which resulted in an underfit. Note that the optimal decision rule is not an element of this model.
- The set $\mathcal{F}$ could be the set of all step functions, which corresponds to the first example above. Note that the optimal decision rule in our example is not an element of this model (but is arbitrarily well

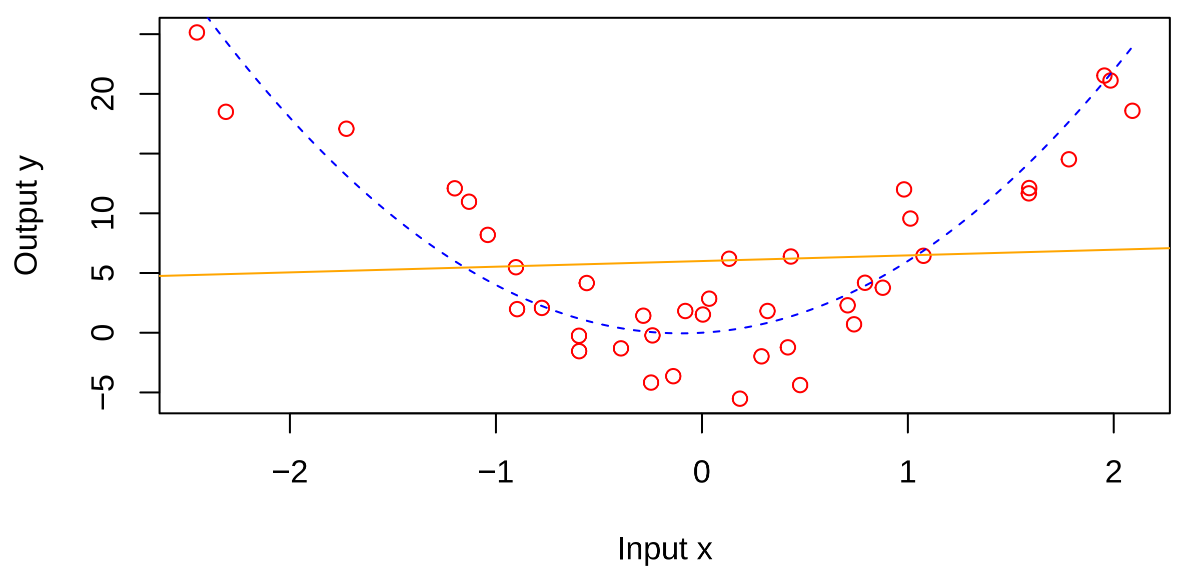

Figure 3: Simple linear regression results in an underfit.

approximated by elements of the model).

- The set $\mathcal{F}$ could also be the set of all polynomials of degree less than or equal to 3, for example. For this model, it actually holds that the optimal decision rule is an element of $\mathcal{F}$.

The first two models are **misspecified**, as they do not contain the optimal decision rule. The third model is **correctly specified**, as it contains the optimal decision rule. In contrast to much of statistics or econometrics, in Machine Learning one does not care as much about using a correctly specified model or estimating the true underlying model. Instead (while one insists on working with a "reasonable" model) the focus here is on determining the "best" decision rule from a *given* set of functions $\mathcal{F}$, which one believes to contain a good approximation to the optimal one.

The fit we obtain when fitting a polynomial regression model of order 3 is shown in Figure 4. The fit is quite good (but obviously does not exactly reproduce the unknown optimal decision function in dashed blue).

2.4.2 How can we learn a decision rule: Empirical risk minimization

Let us return to the general case, where we have p features. Let's also assume for now that we have decided on which model to work with. That is, we fix a set of functions $\mathcal{F}$ which we think contains a good approximation to the optimal predictor. The question now is: how do we pick one of the functions in the model $\mathcal{F}$. In order

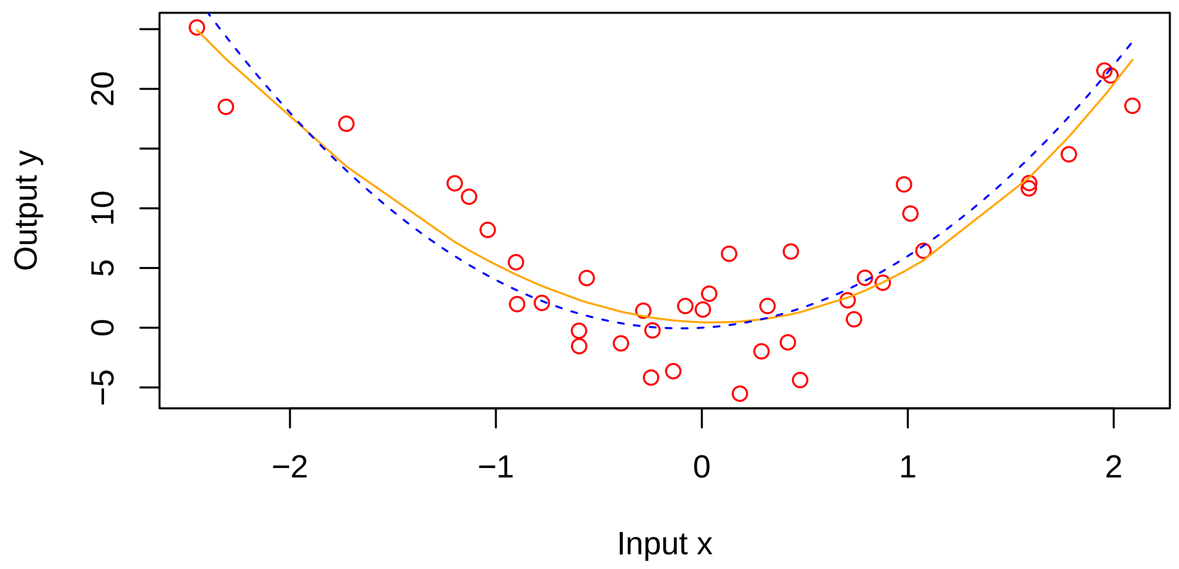

Figure 4: Fitting a polynomial of degree 3.

to develop a general procedure, let's recall what you learned concerning fitting linear models. In this case, $\mathcal{F}$ would coincide with the set of all linear functions of $\boldsymbol{x}$, i.e., functions of the form $\boldsymbol{x} \mapsto b_0 + b_1 x_1 + \ldots + b_p x_p$. One would then proceed to determine estimates $\hat{b}_0, \ldots, \hat{b}_p$ as minimizers to the objective function

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - (b_0 + b_1 X_{i1} + \ldots + b_p X_{ip}))^2 = \frac{1}{n}\|\boldsymbol{Y} - \mathbb{X}(b_0, \ldots, b_p)'\|^2,$$

where, for a vector $\boldsymbol{z}$ we recall that the expression $\|\boldsymbol{z}\|$ denotes its Euclidean norm, which equals $\sqrt{\sum z_i^2}$. Setting $\boldsymbol{Y} = (Y_1, \ldots, Y_n)'$ and $\mathbb{X}$ the matrix with i-th row equal to $(1, X_{i1}, \ldots, X_{ip})$, as you know, this results in the OLS estimator

$$\hat{\boldsymbol{\beta}}_n = (\hat{b}_0, \ldots, \hat{b}_p)' = (\mathbb{X}'\mathbb{X})^{-1}\mathbb{X}'\boldsymbol{Y},$$

with corresponding data-driven decision rule $\hat{f}_n(\boldsymbol{x}) = (1, \boldsymbol{x})\hat{\boldsymbol{\beta}}_n$. So far so good.

To abstract the underlying heuristic, we re-interpret the ingredients in terms of our loss function. In a regression setup we work with a squared loss. In particular, we see that for every linear decision rule $f(\boldsymbol{x}) \mapsto b_0 + b_1 x_1 + \ldots + b_p x_p$ it holds that

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - (b_0 + b_1 X_{i1} + \ldots + b_p X_{ip}))^2 = \frac{1}{n}\sum_{i=1}^{n} L(Y_i, b_0 + b_1 X_{i1} + \ldots + b_p X_{ip}) = \frac{1}{n}\sum_{i=1}^{n} L(Y_i, f(X_i)).$$

The data-driven decision rule we chose was that element f in the set of all linear decision rules $\mathcal{F}$, which minimized $\frac{1}{n}\sum_{i=1}^n L(Y_i, f(X_i))$! One may interpret the latter quantity as an "empirical version" of the unknown risk $\mathsf{R}(f) = \mathbb{E}((Y - f(X))^2)$.

The standard, general approach to learning a decision rule is called **empirical risk minimization**, which is nothing else than the generalization we just described. To see how this works, note that our goal in the *general setup* is to find a decision rule $f^* \in \mathcal{F}$ with minimal risk $\mathsf{R}(f^*)$. The problem is that while we know the loss function L, we cannot compute the risk, because we don't know the distribution of $(Y, \boldsymbol{X})$. To overcome this problem we use the approximation

$$\mathsf{R}(f) = \mathbb{E}(L(Y, f(\boldsymbol{X}))) \approx \frac{1}{n}\sum_{i=1}^n L(Y_i, f(\boldsymbol{X}_i)).$$

The approximation is here justified through the law of large numbers, by which

$$\frac{1}{n}\sum_{i=1}^n L(Y_i, f(\boldsymbol{X}_i)) \to \mathbb{E}(L(Y, f(\boldsymbol{X}))) \quad \text{in probability.}$$

The quantity $\frac{1}{n}\sum_{i=1}^n L(Y_i, f(\boldsymbol{X}_i))$ is called **empirical risk** of the decision rule f; we denote it as $\hat{\mathsf{R}}_n(f)$. Empirical risk minimization is: obtain $\hat{f}_n$ by minimizing $\hat{\mathsf{R}}_n(f)$ over $\mathcal{F}$, i.e.,

$$\boxed{\hat{f}_n \in \arg\max_{f \in \mathcal{F}} \hat{\mathsf{R}}_n(f) \quad \text{Empirical Risk Minimization}}$$

Empirical risk minimization is a very flexible heuristic. The goal obviously is to extract from the data a predictor $\hat{f}_n$ that has similar risk as f, i.e., a predictor which satisfies[3]

$$\inf_{f \in \mathcal{F}} \mathsf{R}(f) \approx \mathsf{R}(\hat{f}_n) = \mathbb{E}_{Y,\boldsymbol{X}}\left(L(Y, \hat{f}_n(\boldsymbol{X}))\right). \tag{4}$$

In words, we hope that the risk of the learned decision rule is close to the minimal achievable risk in the model $\mathcal{F}$ we decided to work with. Note that the quantity to the right is the risk of $\hat{f}_n$. There are two sources of randomness here and the expectation is only taken over $(Y, \boldsymbol{X})$, which we emphasize in the subscript of $\mathbb{E}$.

That the approximation statement in the previous display is indeed true (under suitable assumptions) is the **main theorem of empirical risk minimization** we will take a closer look at (for those of you who have taken an econometrics class and are familiar with M-estimators: the proof is similar as that of consistency of

[3]Recall that the **infimum** of a set A of real numbers is defined as the largest lower bound of that set, where any number that is less than or equal to every element of A is called a lower bound of A. The infimum may be $-\infty$. *If* the infimum is an element of A it is also called its **minimum**. In case this makes you feel very uncomfortable, mentally replace inf by min throughout, and add the assumption that the infimum is attained ...

an M-estimator). Recall that the law of large numbers already implies

$$\hat{\mathsf{R}}_n(f) \to \mathsf{R}(f) \quad \text{in probability, for every } f \in \mathcal{F}.$$

The condition we need in the theorem is stronger, and imposes that the convergence is "uniform" in f, i.e., that the approximation $\hat{\mathsf{R}}_n(f) \approx \mathsf{R}(f)$ is becoming equally better for all $f \in \mathcal{F}$ at the same pace as n grows large.

Theorem 1. *Let $\mathcal{F}$ be a set of decision rules, and assume (using the notation introduced above) that*

$$\sup_{f\in\mathcal{F}} |\mathsf{R}(f) - \hat{\mathsf{R}}_n(f)| \to 0 \quad \textit{in probability.} \tag{5}$$

Then, $\mathsf{R}(\hat{f}_n) \to \inf_{f\in\mathcal{F}} \mathsf{R}(f)$ in probability, where $\hat{f}_n$ minimizes $\mathsf{R}_n(\hat{f}_n)$.

Proof. We need to verify that for every $\varepsilon > 0$ it holds that

$$\mathbb{P}\left(\mathsf{R}(\hat{f}_n) - \inf_{f\in\mathcal{F}} \mathsf{R}(f) \geq \varepsilon\right) \to 0.$$

To this end, fix $\varepsilon > 0$ and let $g \in \mathcal{F}$ be such that $\inf_{f\in\mathcal{F}} \mathsf{R}(f) + \varepsilon/2 \geq \mathsf{R}(g)$. Then, bound

$$\mathsf{R}(\hat{f}_n) - \inf_{f\in\mathcal{F}} \mathsf{R}(f) \leq \mathsf{R}(\hat{f}_n) - \mathsf{R}(g) + \varepsilon/2.$$

Now, we write the upper bound as

$$\mathsf{R}(\hat{f}_n) - \hat{\mathsf{R}}_n(\hat{f}_n) + \hat{\mathsf{R}}_n(\hat{f}_n) - \mathsf{R}(g) + \varepsilon/2 \leq \mathsf{R}(\hat{f}_n) - \hat{\mathsf{R}}_n(\hat{f}_n) + \hat{\mathsf{R}}_n(g) - \mathsf{R}(g) + \varepsilon/2,$$

where we used that $\hat{f}_n$ minimizes the empirical risk, which implies in particular that $\hat{\mathsf{R}}_n(\hat{f}_n) \leq \hat{\mathsf{R}}_n(g)$. By the triangle inequality ($|a-b| \leq |a| + |b|$) and since the absolute value of a number is never smaller than that number, the last upper bound can further be bounded by

$$|\mathsf{R}(\hat{f}_n) - \hat{\mathsf{R}}_n(\hat{f}_n)| + |\hat{\mathsf{R}}_n(g) - \mathsf{R}(g)| + \varepsilon/2 \leq 2\sup_{f\in\mathcal{F}} |\mathsf{R}(f) - \hat{\mathsf{R}}_n(f)| + \varepsilon/2.$$

Summarizing, we have just shown that

$$\mathsf{R}(\hat{f}_n) - \inf_{f\in\mathcal{F}} \mathsf{R}(f) \leq 2\sup_{f\in\mathcal{F}} |\mathsf{R}(f) - \hat{\mathsf{R}}_n(f)| + \varepsilon/2.$$

It hence follows that

$$\mathbb{P}\left(\mathsf{R}(\hat{f}_n) - \inf_{f\in\mathcal{F}} \mathsf{R}(f) \geq \varepsilon\right) \leq \mathbb{P}\left(2\sup_{f\in\mathcal{F}} |\mathsf{R}(f) - \hat{\mathsf{R}}_n(f)| + \varepsilon/2 \geq \varepsilon\right) = \mathbb{P}\left(\sup_{f\in\mathcal{F}} |\mathsf{R}(f) - \hat{\mathsf{R}}_n(f)| \geq \varepsilon/4\right).$$

The upper bound converges to 0 by assumption. Since probabilities are always non-negative, we are done.[4] □

[4]Here we used that for non-negative sequences of real numbers a_n and b_n such that $a_n \leq b_n$, the convergence $b_n \to 0$ implies

2.4.3 Penalized empirical risk minimization

Oftentimes the model $\mathcal{F}$ one intends to work with contains elements of different "complexity". Think of the set of linear decision rules: if the predictor $\boldsymbol{b}$ has more coordinates that equal 0 than another predictor, one may say that the $\boldsymbol{b}$ is less complex than the competitor, as it corresponds to a simpler model; i.e., one with less variables involved. In such a situation, it would be unfair not to incorporate this into the optimization routine. The general idea is to add an additional term $J(f)$, where $J : \mathcal{F} \to \mathbb{R}$, to the empirical risk that penalizes the complexity of a decision rule. One then ends up with a penalized version of empirical risk minimization, which in general is that of determining

$$\hat{f}_n \in \arg\max_{f \in \mathcal{F}} \hat{\mathsf{R}}_n(f) + \lambda J(f) \quad \text{Penalized Empirical Risk Minimization}$$

where $\lambda \geq 0$ is a **penalty** (also called **complexity**) parameter that the user needs to choose.

2.5 Examples for regression problems

We have already seen that the ordinary least squares estimator is obtained as minimizing the empirical risk in a regression problem when the squared loss function is used. We now *revise* three important examples of penalized empirical risk minimizers, namely the **ridge estimator**, the **lasso estimator** and the **elastic net**. The main reason is to see how these fit into the picture.

The situation is as follows: we believe that the set of linear decision rules $\mathcal{F}_{lin}$, say, (potentially in the originally given variables and transformations thereof such as monomials) will result in a good approximation of the true (unknown) optimal decision rule (w.r.t. squared loss). However, there are *many* features available, i.e., p is large and potentially much larger than n, the size of the training sample. In particular, this results into the OLS estimator not being uniquely defined (and in particular the matrix inverse in its traditional form is not well defined). Hence, while we still try to minimize the sum of squares, we want to penalize the complexity of the predictor, which corresponds to $\hat{b}_0, \ldots, \hat{b}_p$. One can quantify the complexity of a predictor in different ways, and the below discussed penalty functions penalize different ways of measuring the complexity. All methods have in common that they penalize the "length" of $(\hat{\beta}_1, \ldots, \hat{\beta}_p)$ (not penalizing the intercept) and therefore "shrink" the coefficient estimates towards 0. This of course makes sense only if we believe that only few of the features have explanatory power concerning the output, which is oftentimes reasonable, but is an aspect that we should keep in mind when applying the decision rules

the convergence $a_n \to 0$, which is sometimes referred to as the "sandwich lemma" in calculus.

2.5.1 Ridge estimator

Here, for a linear decision rule $f_{\boldsymbol{b}}(\boldsymbol{x}) = b_0 + b_1 x_1 + \ldots + b_p x_p$, the penalty function J_{Ridge} used is

$$J_{Ridge}(f_{\boldsymbol{b}}) = \sum_{i=1}^{p} b_i^2.$$

The penalized empirical risk minimizer corresponding to this penalty function then is and element of

$$\arg \min_{f_{\boldsymbol{b}} \in \mathcal{F}_{lin}} \hat{\mathsf{R}}_n(f) + \lambda J_{Ridge}(f) = \frac{1}{n} \sum_{i=1}^{n} (Y_i - (b_0 + b_1 X_{i1} + \ldots + b_p X_{ip}))^2 + \lambda \sum_{i=1}^{p} b_i^2,$$

which we can write more compactly as

$$\arg \min_{f_{\boldsymbol{b}} \in \mathcal{F}_{lin}} n^{-1} \|\boldsymbol{Y} - \mathbb{X}\boldsymbol{b}\|^2 + \lambda \|(b_1, \ldots, b_p)'\|^2.$$

It is good to know that the solution of this problem can be written down explicitly. It equals

$$(\mathbb{X}'\mathbb{X} + \lambda \boldsymbol{I}_p)^{-1} \mathbb{X}' \boldsymbol{Y}.$$

To see this, solve the first order conditions of the objective function.[5]

2.5.2 Lasso estimator

The lasso estimator is obtained similarly as the ridge estimator, but is based on the penalty function $J_{Lasso}(f_{\boldsymbol{b}}) = \sum_{i=1}^{p} |b_i|$. The resulting optimization problem then is

$$\arg \min_{f_{\boldsymbol{b}} \in \mathcal{F}_{lin}} n^{-1} \|\boldsymbol{Y} - \mathbb{X}\boldsymbol{b}\|^2 + \lambda \sum_{i=1}^{p} |b_i|.$$

In general, giving an explicit, simple solution to this optimization problem is not possible. Therefore, the problem has to be solved numerically. This means that the optimization problem is solved by a **numerical optimization algorithm**. For the Lasso, the main algorithm to obtain the solution is called **least angle regression** (LAR) algorithm (cf. also the discussion in Hastie et al. [2009] Chapter 3.4). This algorithm proceeds as follows, where we denote $\overline{\boldsymbol{Y}} = (\overline{Y}, \ldots, \overline{Y})'$, i.e., the (constant vector) with entry $\overline{Y} = n^{-1} \sum_{i=1}^{n} Y_i$:

To formulate the algorithm, for a non-empty set $A \subseteq \{0, 1, \ldots, p\}$ we denote by $\mathbb{X}_A$ the matrix consisting of all columns with indices in A (0 corresponding to the first column of $\mathbb{X}$, etc.). Furthermore, denote for any such A and for any $\boldsymbol{R}$ the vector $\delta_A(\boldsymbol{R}) = (\mathbb{X}_A'\mathbb{X}_A)^{-1}\mathbb{X}_A'\boldsymbol{R}$. Finally, for a vector $\boldsymbol{b}$ we denote by $\boldsymbol{b}_A$ the vector of all coordinates of $\boldsymbol{b}$ with indices in A and the other coordinates dropped.

[5] Alternatively, perhaps more elegantly, define $\boldsymbol{X}_* = (\boldsymbol{X}', \sqrt{\lambda}\boldsymbol{I}_p)'$, $\boldsymbol{Y}_* = (\boldsymbol{Y}', 0, \ldots, 0)'$ with 0 zeros added, and note that $\|\boldsymbol{Y}_* - \boldsymbol{X}_*\boldsymbol{b}\|^2 = \|\boldsymbol{Y} - \boldsymbol{X}b\|^2 + \lambda\|b\|^2$, which by the general OLS theory is minimized for $(\boldsymbol{X}_*'\boldsymbol{X}_*)^{-1}\boldsymbol{X}_*'\boldsymbol{Y}_*$, which coincides with $(\boldsymbol{X}'\boldsymbol{X} + \lambda\boldsymbol{I}_p)^{-1}\boldsymbol{X}'\boldsymbol{Y}$.

LARS Algorithm

1. Assume that the feature vectors $\boldsymbol{X}_{\cdot 1}, \dots, \boldsymbol{X}_{\cdot p}$ (i.e., the column vectors of $\boldsymbol{X}$) have mean zero (otherwise subtract from each column vector the average of its coordinates and continue) and unit norm (otherwise divide each column vector by its Euclidean norm and continue). Start with the residual $\boldsymbol{R} = \boldsymbol{Y} - \overline{\boldsymbol{Y}}$, and set $\boldsymbol{b} = (\overline{Y}, b_1, \dots, b_p)' = (\overline{Y}, 0, \dots, 0)$.
2. Determine the index $i \in \{1, \dots, p\}$ which maximizes $|\delta_{\{i\}}(\boldsymbol{R})|$, and initialize the "active set" as $A = \{i\}$.
3. Repeatedly update $\boldsymbol{b}_A \mapsto \boldsymbol{b}_A + \alpha \delta_A(\boldsymbol{R})$ (α a small positive number), re-computing the residual $\boldsymbol{R} = \boldsymbol{Y} - \mathbb{X}_A \boldsymbol{b}_A$ accordingly, until some $j \in A^c$ satisfies $|\delta_{\{j\}}(\boldsymbol{R})| \geq |\delta_A(\boldsymbol{R})|$. Then, update the active set to $A \cup \{j\}$.
4. Repeat 3, where if a non-zero coefficient hits zero in the updating process, we drop its variable from the active set of variables.
5. Continue this way until all p features have been entered.

An implementation of this algorithm is available in the `R` package lars.

2.5.3 Elastic net

The elastic net is a hybrid version of the Ridge and Lasso estimator, as it uses a penalty function which is a combination of J_{Ridge} and J_{Lasso}, i.e., is of the form $J_{EN} = (1-\alpha)/2 J_{Ridge} + \alpha J_{Lasso}$ for a real number $\alpha \in [0, 1]$. For example, note that if $\alpha = 1$ we recover the Lasso estimator.

We don't go into much detail. Let us just mention that an algorithm for computing this estimator is available in the `R` package glmnet.

The LAR algorithm can not be used here. Instead, for this package, the authors use (cyclical) **coordinate decent**, as they describe in the vignette to the package, which you can access here and which contains further details and examples.

2.6 Simple multivariable optimization algorithms

To minimize a differentiable function $f : \mathbb{R}^p \to \mathbb{R}$ by **gradient descent**, we proceed as follows (recall that the gradient of f, i.e., ∇f is nothing else than the vector of partial derivatives of f):

0. Specify a sequence $\gamma_m > 0$ (e.g., $\gamma_m = 1/m$) and a starting point $\boldsymbol{a} \in \mathbb{R}^p$.
1. For $m = 1, 2, \dots$ repeat:
    - Compute the gradient of f at $\boldsymbol{a}$, i.e., $\nabla f(\boldsymbol{a})$.

- Update $\boldsymbol{a}$ to $\boldsymbol{a} - \gamma_n \nabla f(\boldsymbol{a})$.

In practice the iteration stops, e.g., after a maximal number of initially specified iterations or after the function values improve only marginally (or a combination of the two).

The "step size" γ_m can also be constructed automatically in each step, e.g., via Barzilai–Borwein method (cf. Barzilai and Borwein [1988])

$$\gamma_m = |(\boldsymbol{a}_m - \boldsymbol{a}_{m-1})'(\nabla f(\boldsymbol{a}_m) - \nabla f(\boldsymbol{a}_{m-1}))| / \|\nabla f(\boldsymbol{a}_m) - \nabla f(\boldsymbol{a}_{m-1})\|^2,$$

where $\boldsymbol{a}_m$ denotes the current vector $\boldsymbol{a}$ at step n of the algorithm. One can also do a simple line search along $\boldsymbol{a} - \gamma \nabla f(\boldsymbol{a})$ (i.e., find a solution to this one-dimensional optimization problem) to determine the "best" step size, which is more costly.

If there are constraints, one can modify the algorithm and try to project the updated value in each iteration on the sets specified by the constraints.

To minimize a (possibly non-differentiable) function $f : \mathbb{R}^p \to \mathbb{R}$ by **cyclical coordinate descent**, we may simplify the task to minimize the function f in a single coordinate in every step:

0. Specify a starting point $\boldsymbol{a} \in \mathbb{R}^p$.
1. For $n = 1, 2, \ldots$ repeat (with i cycling through the coordinates $1, \ldots, p$):
    - Obtain $\boldsymbol{a}$ by changing the i-th coordinate of the current $\boldsymbol{a}$ to an element of

$$\arg\min_{y \in \mathbb{R}} f(a_1, \ldots, a_{i-1}, y, a_{i+1}, \ldots, a_p).$$

The (one-dimensional) minimum we need to determine in each step in this algorithm can be determined by standard methods for univariate optimization.

2.7 Training, validation and testing

We now know that $\hat{f}_n$ (when obtained through empirical risk minimization) has a risk $\mathsf{R}(\hat{f}_n)$ that approaches the minimal risk attainable over $\mathcal{F}$ as the size of the training sample grows large. Unfortunately, we do not know the risk $\mathsf{R}(\hat{f}_n)$ in practice, as this depends on the underlying distribution of $(Y, \boldsymbol{X})$, which is unknown.

A naive first approach (that is invalid) would be to estimate $\mathsf{R}(\hat{f}_n)$ via the empirical risk of $\hat{f}_n$, i.e.,

$$\hat{\mathsf{R}}_n(\hat{f}_n) = \frac{1}{n} \sum_{i=1}^{n} L(Y_i, \hat{f}_n(\boldsymbol{X}_i)).$$

This quantity is also called **training error**. However, one should not use $\hat{\mathsf{R}}_n(\hat{f}_n)$ to estimate $\mathsf{R}(\hat{f}_n)$:

the training data was used to estimate $\hat{f}_n$, and it therefore typically holds that $\hat{\mathsf{R}}_n(\hat{f}_n) \leq \mathsf{R}(\hat{f})$, i.e., one underestimates the risk in that way. Put differently, this also leads to overfitting (note that in the extreme case $\hat{\mathsf{R}}_n(\hat{f}_n) = 0$, because we just fit a step function, which does not come close to the true risk of $\hat{f}_n$).

To overcome this disadvantage, one can (if available) estimate $\mathsf{R}(\hat{f}_n)$ using a **testing sample** $(\tilde{Y}_i, (\tilde{X})_i)$ for $i = 1, \dots, m$ that is independent of the training sample, is i.i.d., and has the same distribution as $(Y, \boldsymbol{X})$. One estimates $\mathsf{R}(\hat{f}_n)$ via

$$\mathsf{empR}(\hat{f}_n) = \frac{1}{m} \sum_{i=1}^{m} L(\tilde{Y}_i, \hat{f}_n((\tilde{X})_i)).$$

By the law of large numbers, it is clear that (conditionally on the training sample) and as $m \to \infty$, $\mathsf{empR}(\hat{f}_n)$ converges in probability to $\mathsf{R}(\hat{f}_n)$.

Since typically there is no additional testing sample available, one has to split the training sample in one part used for the actual training, and another part that is used for testing. In the testing sample one can then obtain an estimate of the risk of the decision rule learned on behalf of the training sample.

If the decision rule is based on a tuning parameter (e.g., the quantity λ in the definition of the Lasso), this tuning parameter can also be chosen in a data-driven way. One can do this by what is called **validation**. To do this, suppose that there is a **validation sample** $(\check{Y}_i, (\check{X})_i)$ for $i = 1, \dots, l$ available that is independent of the training sample, is i.i.d., and has the same distribution as $(Y, \boldsymbol{X})$.

1. Determine $\hat{f}_{n,\lambda}$ using the training sample and for different values of the tuning parameter λ.
2. Compute $\mathsf{empR}(\hat{f}_{n,\lambda})$ for every λ for which a decision rule was computed by using the testing sample, i.e.,
$$\mathsf{empR}(\hat{f}_{n,\lambda}) = \frac{1}{l} \sum_{i=1}^{l} L(\check{Y}_i, \hat{f}_n((\check{X})_i)).$$
3. Choose $\hat{\lambda}$ as that value of the tuning parameter which minimizes $\mathsf{empR}(\hat{f}_{n,\lambda})$ among the considered values of λ.

Note that to estimate the risk of such an estimator $\hat{f}_{n,\hat{\lambda}}$, one needs another test sample (that is independent of the training sample and the validation sample). In practice, one splits up the original data into three parts (e.g., 60%, 20%, 20%) and uses the first as the training sample, the second as the validation sample, and the last as the testing sample.

Another approach is **cross validation**. Here the underlying idea is to

1. Split the training sample into M equally sized subsamples. That is $\{1, \dots, n\} = I_1 \cup \dots \cup I_M$, where the sets I_j are disjoint and non-empty.

2. For $j = 1, \dots, M$ determine the decision rule $\tilde{f}_{j,n,\lambda}$ for a range of values for the data $(Y_i, \boldsymbol{X}_i)$ with $i \notin I_j$.
3. Select $\hat{\lambda}^{cv}$ as a minimizer of

$$CV_n(\lambda) = \frac{1}{n} \sum_{j=1}^{M} \sum_{i \in I_j} L(Y_i, \tilde{f}_{j,n,\lambda}(\boldsymbol{X}_i)).$$

If $\hat{\lambda}^{cv}$ was obtained that way, we say that the tuning parameter was obtained through M-fold cross validation. Note that $M = n$ corresponds to dropping one observation at a time (that is, the condition $i \notin I_j$ coincides with $i \neq j$). The latter is called **leave-one-out cross-validation**.

3 Data example

We look at the Boston housing data from the **MASS** library, which we can simply load via

```
data("Boston", package = "MASS")
```

The goal is to build a predictive model for the median house value `medv` in Boston Suburbs, using multiple features (among which: `crim` = per capita crime rate by town, or `rm` = average number of rooms per dwelling).

The package we shall be using is **glmnet** which we already introduced. Let's load the packages via:

```
library(glmnet)
```

We now want to specify `y` the vector of outputs and `x` the feature matrix. To apply `glmnet` the data needs to be pre-processed, because that function can only take numerical inputs (and no factors or the like). We can do this by using the function `model.matrix` which generates the regressor matrix (e.g., dummies instead of factors) automatically from the regressors in the sample.

```
# Features
x <- model.matrix(medv ~ ., Boston)[,-1]
# Outcome variable
y <- Boston$medv
```

Next, for later use, we choose a training and a testing sample. To this end, we sample the coordinates of the training sample randomly (we set a seed to guarantee reproducibility). These coordinates are the indices that are in the training sample, and the remaining ones will be used for testing (i.e., for comparing the different decision rules we obtain).

```
set.seed(1)
n <- dim(Boston)[1]
tsa <- sample(1:n, size = floor(0.8*n), replace = FALSE)
xtr <- x[tsa,]
ytr <- y[tsa]
```

The syntax of `glmnet`, which we will use to obtain the ridge, lasso and elastic net estimator, is very simple. It amounts to `glmnet(x, y, alpha = 1, lambda = NULL)`, the α and λ corresponding to the parameters we had in the section on the elastic net. We shall choose λ through cross validation (the number of folds being the default value of `10`), i.e., we actually the function `cv.glmnet` which has the same syntax (with more options) but drops the λ. The value α should be supplied by the user and determines which estimator ($\alpha = 0$ is the Lasso; $\alpha = 1$ is the Ridge; and $\alpha \in (0, 1)$ corresponds to some genuine elastic net decision rule).

Let's obtain the tuning parameters through cross-validation and fit the decision rules:

```
cv.lasso  <- cv.glmnet(xtr, ytr, alpha = 1)
cv.ridge  <- cv.glmnet(xtr, ytr, alpha = 0)
cv.en     <- cv.glmnet(xtr, ytr, alpha = 1/3)


c(cv.lasso$lambda.min, cv.ridge$lambda.min, cv.en$lambda.min)
```

```
## [1] 0.03180808 0.70140971 0.02847123
```

```
fit.lasso  <- glmnet(xtr, ytr, alpha = 1, lambda = cv.lasso$lambda.min)
fit.ridge  <- glmnet(xtr, ytr, alpha = 0, lambda = cv.ridge$lambda.min)
fit.en     <- glmnet(xtr, ytr, alpha = 1/2, lambda = cv.en$lambda.min)
```

It is interesting to see how the selected tuning parameters perform in comparison to a naive computation of the training error, which is provided in Figure 5

```
## [1] -3.4480350 -0.3546631 -3.5588613
```

We can compare the coefficients obtained through

```
cbind(coef(fit.lasso), coef(fit.ridge), coef(fit.en))
```

```
## 14 x 3 sparse Matrix of class "dgCMatrix"
##                       s0           s0           s0
## (Intercept)  30.251329815  24.929066941  31.215046299
```

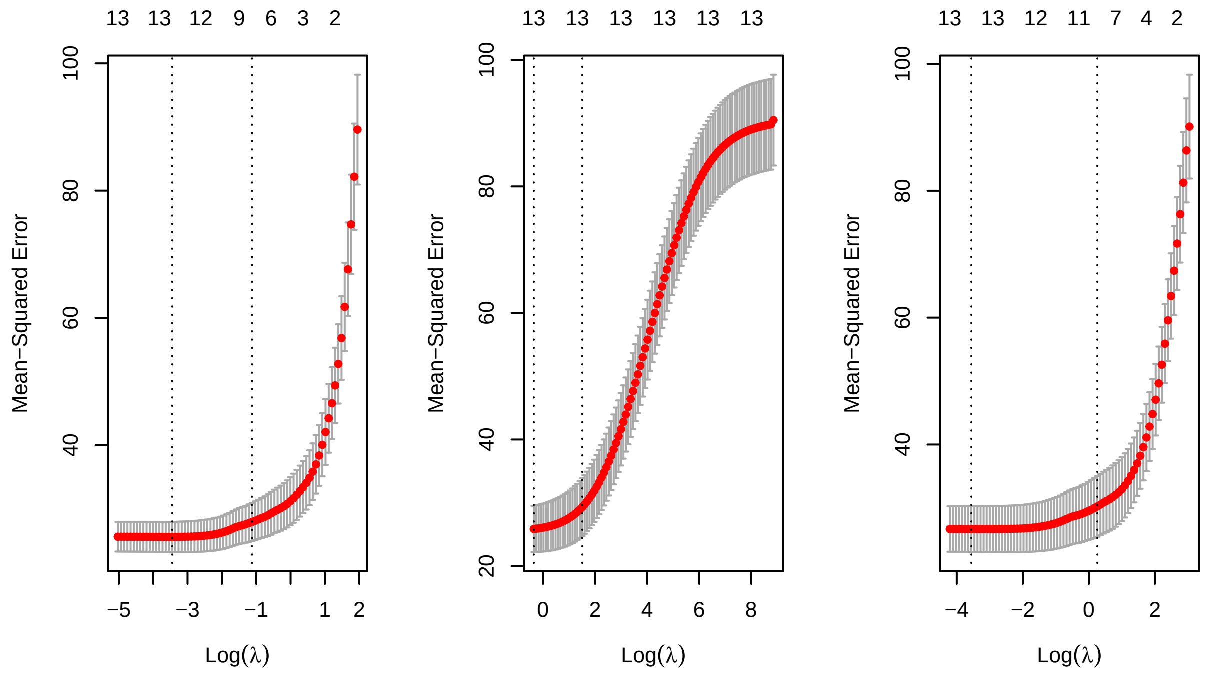

Figure 5: Cross validation in the Boston housing example.

```
## crim          -0.083268107  -0.075610625  -0.088901131
## zn             0.035763379   0.028493595   0.037877507
## indus          .            -0.027765327   0.008554279
## chas           3.154103723   3.386558856   3.168688709
## nox          -13.411964098 -10.071302245 -14.075035227
## rm             4.112054916   4.244517016   4.094520683
## age           -0.002777576  -0.007544708  -0.004156368
## dis           -1.343403034  -1.101665762  -1.391881035
## rad            0.266988157   0.172837434   0.290386571
## tax           -0.010529715  -0.006426103  -0.011580502
## ptratio       -0.859183661  -0.808316248  -0.869391812
## black          0.010259730   0.010234262   0.010418228
## lstat         -0.552679835  -0.497844279  -0.550305276
```

More important than the coefficients is the predictive performance of the respective method. To this end, we can now make use of the testing sample.

```
xte <- x[-tsa,]
yte <- y[-tsa]
```

To obtain the predictions, we do

```
pred.lasso <- predict(fit.lasso, newx = xte)
pred.ridge <- predict(fit.ridge, newx = xte)
pred.en    <- predict(fit.en, newx = xte)

empR.lasso <- mean((yte - pred.lasso)^2)
empR.ridge <- mean((yte - pred.ridge)^2)
empR.en    <- mean((yte - pred.en)^2)

RL <- list("Risk Lasso" = empR.lasso, "Risk Ridge" = empR.ridge,
        "Risk EN" = empR.en)
RL
```

```
## $`Risk Lasso`
## [1] 17.30721
##
## $`Risk Ridge`
## [1] 17.13236
##
## $`Risk EN`
## [1] 17.29465
```

Hence, in this case the risk of the ridge estimator is the smallest, which we would now take as our predictive model (among the three methods considered). One could now go on and look for transformations of the original regressors, which one could add, or interactions, etc. to increase the feature space. Furthermore, one could try to take a look at other parameters α in the elastic net, which could perform better than $\alpha = 1/3$ which was used in the above analysis. One could also repeat the analysis a couple of times with different training and testing samples (just to check whether the conclusion is "robust" to the training/testing sample split we obtained, which was random and hence arbitrary).

It also makes sense to take the square root of the risk estimates, because these numbers are then on the same scale as the outcomes, and hence easier to interpret.

```
sapply(RL, sqrt)
```

```
## Risk Lasso Risk Ridge    Risk EN
##   4.160193   4.139126   4.158684
```

This also illustrates that the predictive performance of the methods is very comparable. The average error in the comparison is roughly 4.

An automatic (but less pedagogical way, perhaps) to do model comparisons that is good to know is the package **caret**, which you are encouraged to take a look at!

4 Problems for next week's exercise session

Do one of the following exercises:

1. Write an `R` program that implements the LAR algorithm. The program should take as an input a vector $\boldsymbol{Y}$ and a matrix $\mathbb{X}$ and should output the coefficient path. You may want to read the above-mentioned chapter in Hastie et al. [2009] for more information, but essentially the algorithm description above should be enough to implement that function. Illustrate your algorithm on an example of your choice (potentially the data example we used in class or a simulated data set). (worth a maximum of **5 points** in total)
2. Similarly as in 1., but here you are asked to write an `R` program that implements the Ridge estimator. Apply this program to a data example of your choice (possibly chosen as in 1.) for various values of the tuning parameter, and plot the path (i.e., the estimators in dependence on the tuning parameter for different values of λ). (worth a maximum of **3 points** in total)
3. Write an `R` program that takes as input a function $f : \mathbb{R}^p \to \mathbb{R}$ and returns a minimizer of that function that is obtained through steepest descent (you can also incorporate restrictions into your program by projecting onto the feasible set in case the feasible set is of the form $[a, b] \times [a, b] \times \ldots \times [a, b]$ for real numbers $a < b$). To compute the gradient (you will need in your function) use the `R` package **numDeriv** which determines the gradient of a function numerically. Illustrate the behavior of your program on the function $f : \mathbb{R}^2 \to \mathbb{R}$ with $f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$ (called Himmelblau's function). Plot the path of function values obtained for this example for a number of starting values and describe what you observe. (worth a maximum of **5 points** in total)

You may also want to optimize your code (write as much in terms of matrix manipulations as possible and avoid loops; to measure the runtime of your code (or subroutines) you can use the approaches as detailed here. If you want a challenge (that is worthwhile taking if you have time), you can recode bottlenecks in your code in C++ and incorporate that relatively easily via the package **Rcpp** as detailed here.

References

J. Barzilai and J. Borwein. Two-Point Step Size Gradient Methods. *IMA Journal of Numerical Analysis*, 8 (1):141–148, 01 1988.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer: New York, 2009.

Stefan Richter. *Statistisches und maschinelles Lernen.* Springer: Berlin, 2019.