Probabilistic aspects of machine learning

by Peter Pfaffelhuber

Version: January 24, 2023

Contents

1 Introduction

> Here are some general textbooks on machine learning and related topics, which I used in the preparation of this manuscript: [6, 13], [11], [10], [17], [12], [1].

In this lecture, we will connect the fields of machine learning and probability theory (or statistics). Both fields aim to learn something from data. Let us use a popular definition of machine learning from the textbook by Mitchell (McGraw Hill, 1997):

> A computer program is said to learn from experience E with respect to some class of tasks T, and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

The experience which is mentioned here, usually comes from data. One task T described can be the prediction of future data, and the performace measure P is some metric relating the truth to the predicted future data. Another case would be the classification of items into one of finitely many classes. The experience E is based on some training data, and P is the fraction of mis-classified items.

For the statistical side, we start with a formal definition:

Definition 1.1 (Statistical model). *Let $\mathbb{S}, \Theta, \Theta'$ be sets. A* statistical model *is a pair $(X, (\mathbf{P}_\vartheta)_{\vartheta\in\Theta})$, where X is an $\mathbb{S}$-valued random variable, with a distribution with free (i.e. undetermined) parameter $\vartheta \in \Theta$. In other words, there is a function $\vartheta \mapsto \rho_\vartheta$ with*[1]

$$\mathbf{P}_\vartheta(X \in da) = \rho_\vartheta(a)da.$$

The set Θ is called parameter space*, and $\mathbb{S}$ is observational space. Every Θ'-valued random variable $h(X)$ is called* statisics*.*

The obvious difference between these two *definitions* is the use of probabilities. Whereas machine learning usually comes without probabilistic interpretations, the statistical model already comes with a family of probability distributions. Let us discuss this in some more detail using a standard example.

Remark 1.2 (Notation). We will try to make the following conventions on the notation: Deterministic variables (and constants) are lower case letters $a, b, ..., x, y, z$, while random variables are upper case letters $X, Y, ...$ In both cases, we also write x (or X) for a vector (or a matrix). Greek letters $\beta, \vartheta, ...$ are reserved for model parameters, and $\hat\beta, \hat\vartheta, ...$ for their estimators, which are also random variables (in addition to capital letters). Sets are upper case math bold font letters $\mathbb{R}, \mathbb{A}, \Theta, ...$ Probability measures are bold letters $\mathbb{P}$, and $\mathbf{E}$ their expectation.

[1] In the sequel, we want to avoid to distinguish between random variables with density, and discrete random variables, and write . $\mathbf{P}(X \in da)$ in both cases. If X is discrete and $a \in \mathbb{S}$, we write

$$\mathbf{P}(X \in da) := \mathbf{P}(X = a)$$

gemeint. If X has density $f(a)da$, then.

$$\mathbf{P}(X \in da) := f(a)da.$$

1.1 Linear regression

When looking at quantitative data, we might be interested in understanding connections between different coordinates. For example, there is a relationship between the size and the price of an appartment, or the number of cars on a road and the average velocity. In addition, a *covariate* (e.g. price of an appartment) might depend on other covariates (e.g. the location of the appartment). In most such situations, one can use a regression in order to find out correlations between covariates. Assume we have n items and from item i, we observe k covariates $x_{i1}, ..., x_{ik}$ as well as the goal variable y_i, $i = 1, ..., n$. Using a linear model, we might assume that

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_k x_{ik} + \varepsilon_i, \qquad i = 1, ..., n \tag{1.1}$$

for some $\beta_0, ..., \beta_k, \varepsilon_1, ..., \varepsilon_n$. Clearly, when we determine $\beta_0, ..., \beta_k$, then the errors $\varepsilon_1, ..., \varepsilon_n$ can be computed. In order to obtain an estimate for $\beta := (\beta_0, ..., \beta_k)$, we aim at minimizing the performance measure[2,3]

$$RSS(\beta) := \sum_{i=1}^{n} \varepsilon_i^2 = \sum_{i=1}^{n} (y_i - x_{i\cdot}\beta)^2 = (y - x\beta)^\top (y - x\beta) = y^\top y - 2y^\top x\beta + \beta^\top x^\top x\beta. \tag{1.2}$$

Theorem 1.3 (Multiple Regression). *If $x^\top x$ is invertible, there is the unique minimum of $\beta \mapsto RSS(\beta)$ at*

$$\hat{\beta} = (x^\top x)^{-1} x^\top y.$$

For the prediction

$$\hat{y} := x\hat{\beta} (= x(x^\top x)^{-1} x^\top y),$$

we have[4]

$$y - \hat{y} = (I - x(x^\top x)^{-1} x^\top)\varepsilon.$$

Moreover, $y - \hat{y}$ is perpendicular on both, $\hat{y}$, and the columns of x.

Remark 1.4 (Minimal RSS). The value of the minimum of the *Residual Sum of Squares* is

$$RSS := RSS(\hat{\beta}) = \sum_{i=1}^{n} (y_i - x_{i\cdot}\hat{\beta})^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = (y - \hat{y})^\top (y - \hat{y}).$$

Proof of Theorem 1.3. We use some results from Calculus; see also Appendix A. A necessary condition for $\hat{\beta}$ being a minimum of $\beta \mapsto RSS(\beta)$ is

$$0 = \tfrac{1}{2} \nabla RSS(\beta) = -y^\top x + \beta^\top x^\top x = (x^\top x\beta - x^\top y)^\top,$$

so $\beta \mapsto RSS(\beta)$ has an extremal point at

$$\hat{\beta} = (x^\top x)^{-1} x^\top y.$$

[2] *Residual Sum of Squares*

[3] Here and in the sequel, z is a column vector, and $z^\top$ is a row vector.

[4] Here, I is the unit matrix.

Then, the Hessian of $\beta \mapsto RSS(\beta)$ is $\nabla^2 RSS(\beta) = x^\top x$, independent of β. This is a positive definite matrix since $x^\top x$ is invertible (so has full rank). For the second assertion we find

$$\begin{aligned} y - \hat{y} &= (I - x(x^\top x)^{-1}x^\top)y = (I - x(x^\top x)^{-1}x^\top)(x\beta + \varepsilon) \\ &= x\beta + \varepsilon - x\beta - x(x^\top x)^{-1}x^\top\varepsilon = (I - x(x^\top x)^{-1}x^\top)\varepsilon. \end{aligned}$$

Furthermore, we write

$$\begin{aligned} (y - \hat{y})^\top \hat{y} &= y^\top x(x^\top x)^{-1}x^\top y - y^\top x(x^\top x)^{-1}x^\top x(x^\top x)^{-1}x^\top y = 0, \\ (y - \hat{y})^\top x &= y^\top x - y^\top x(x^\top x)^{-1}x^\top x = 0, \end{aligned}$$

and orthogonality follows. □

The performance measure in (1.2) was an ad hoc choice. It becomes clearer after introducing a statistical view and introducing a statistical model.

Definition 1.5 (Statistical model for regression). *Let $x \in \mathbb{R}^{n\times k}$, $\mathbb{S} = \mathbb{R}^n$ and $\mathbb{O} = \mathbb{R}^k$. For some $\sigma^2 > 0$, let $\mathbf{P}_\beta$ be such that [5] (with I the unit matrix)*

$$\mathbf{P}_\beta(Y \in .) = N(x\beta, \sigma^2 I),$$

i.e. $Y = (Y_1, ..., Y_k)$ is normally distributed with $\mathbf{E}[Y_i] = x_i\beta, \mathbf{V}[Y_i] = \sigma^2$, and $Y_1, ..., Y_k$ are independent.

Remark 1.6 (Maximum-Likelihood estimator). For a statistical model $(X, (\mathbf{P}_\vartheta)_{\vartheta\in\mathbb{O}})$, we introduce the *likelihood function*. It is given by

$$\mathbf{L} : (\vartheta, x) \mapsto \mathbf{P}_\vartheta(X \in dx).$$

For $x \in \mathbb{S}$, we call any maximizer

$$\widehat{\vartheta}^* := \text{argmax}_{\vartheta\in\mathbb{O}}\mathbf{L}(x, \vartheta)$$

a *Maximum-Likelihood estimator* for ϑ. Sometimes, we call $\ell : (\vartheta, x) \mapsto \log \mathbf{L}(\vartheta, x)$ the *log-likelihood function*. Since log is strictly monotone, any maximizer for $\vartheta \mapsto \mathbf{L}(\vartheta, x)$ is a maximizer of $\vartheta \mapsto \ell(\vartheta, x)$ and vice versa.

Proposition 1.7 (Maximum Likelihood estimator for the regression model). *Under the model in Definition 1.5, assuming that $x^\top x$ is invertible, the maximum-likelihood estimator of β is unique and given by*

$$\hat{\beta} = (x^\top x)^{-1}x^\top Y.$$

Proof. The log-likelihood is given by

$$2\sigma^2\ell(\beta, y) = C_{\sigma^2} - \sum_{i=1}^{n}(y_i - x_{i\cdot}\beta)^2$$

for some C_{σ^2} not depending on β. Hence, the maximum-likelihood estimator of β is the unique minimizer of RSS from (1.2). Hence, the result follows from Theorem 1.3. □

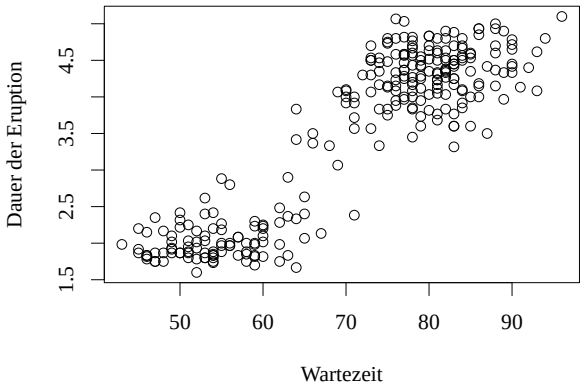


Figure 1.1: The example from the `faithful` dataset, which is included in `R`.

Remark 1.8. From Proposition 1.7, we see that the choice of RSS as the performance measure in linear regression is not ad hoc. Rather, we see the assumption that $Y \sim N(x\beta, \sigma^2 I)$ in fact leads to this choice. In particular, we see that we have implicitely assumed that all Y_i's have the same variance.

Example 1.9 (Regression with `R`). We are using the dataset `faithful` from the 1980s, which is implemented in `R` [15]. The first lines can be read via[6]

```
> head(faithful)
```

which results in

```
  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
4     2.283      62
5     4.533      85
6     2.883      55
```

The covariate `waiting` is the waiting time until the next eruption of the *Old Faithful Gaysier*, located in the Yellowstone National Park, and `eruptions` are the respective durations. In order to get a first impression of the data, one could plot the datapoints by

```
> duration = faithful$eruptions
> waiting = faithful$waiting
> plot(waiting, duration, xlab="Waiting time until next eruption",
  ylab="Duration of the eruption")
```

Obviously there is a connection between waiting time and its duration. The corresponding regression line can be computed in `R` by

[5] We recall the multivariate normal distribution in Appendix B.

[6] The command `head` only gives the first few lines of the dataset. If you want to view the whole dataset, you can type in the name of the dataset `faithful` ein.

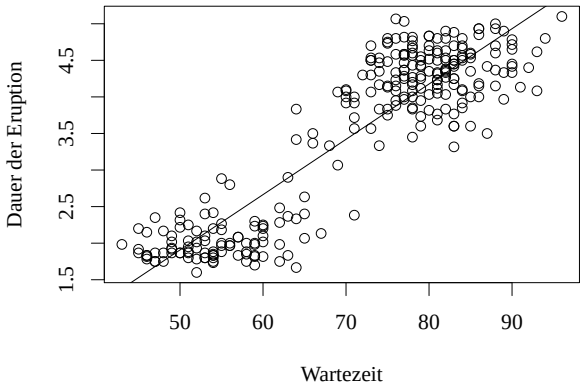


Figure 1.2: The `faithful` dataset and the computed reression line.

```
> lm(eruptions ~ waiting, data=faithful)
```

This gives the output:

```
Coefficients:
(Intercept)      waiting
   -1.87402      0.07563
```

This means that `R` has estimated the linear function

$$\hat{Y} = -1.87402 + 0.07563x$$

for the waiting time Y depending on the eruption duration x; see also Figure 1.5. Here, I have used

```
> coeffs=coefficients(lm(eruptions ~ waiting, data=faithful))
> coeffs=as.vector(coeffs)
> abline(coeffs)
```

for plotting the line. In order to store the figure into a `pdf`, I have used

```
> pdf(file = "fig1.pdf", width=7, height=5, family="Times", onefile=FALSE)
> par(mar=c(5,4,1,1), cex=1.5)
```

before the `plot`-command. (`par` changes the margins.) After the `plot`-command, do not forget to use

```
> dev.off()
```

for closing the `pdf` correctly.

1.2 Logistic regression

Consider again the regression model from Definition 1.5 (again with a sample size of n and k covariates), but assume that Y only takes values in a discrete set (of class labels), e.g. $\mathbb{S} = \{\text{black}, \text{white}\}$, or $\mathbb{S} = \{0, ..., 9\}$. Here, linear regression is not useful. In this case, we speak of a *classification problem*, since we aim at classifying $i = 1, ..., n$ into some $y_i \in \mathbb{S}$ based on x_i. In this section, we learn about one of the simplest method for classification, logistic regression. The model is as follows:

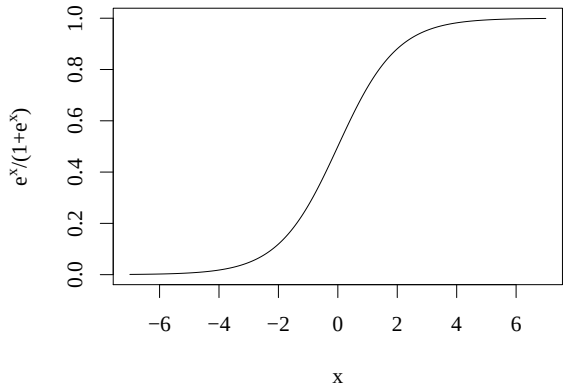


Figure 1.3: The softmax function from Remark 1.11.1.

Definition 1.10 (Logistic regression). *Let $k, n \in \mathbb{N}$, $\mathbb{S}$ be finite, $x \in \mathbb{R}^{n\times k}$ and $\Theta = \mathbb{R}^{k\times\mathbb{S}}$. For the $\mathbb{S}^n$-valued random variable Y and $\beta = (\beta_{jc})_{j=1,\dots,k,c\in\mathbb{S}} \in \Theta$, let $\mathbf{P}_\beta$ be such that*

$$\mathbf{P}_\beta(Y = y) = \prod_{i=1}^{n} \frac{e^{x_{i\cdot}\beta_{\cdot y_i}}}{\sum_c e^{x_{i\cdot}\beta_{\cdot c}}}, \qquad y \in \mathbb{S}^n,$$

i.e. $Y = (Y_1, ..., Y_n)$ is independent and [7]

$$\mathbf{P}(Y_i = y_i) \sim e^{x_{i\cdot}\beta_{\cdot y_i}}.$$

Remark 1.11 (Softmax, binary logistic regression, parameter redundancy).

1. In machine learning, the function

$$\sigma : \begin{cases} \mathbb{R}^k & \to (0,1)^k \\ \eta = (\eta_i)_{i=1,\dots,k} & \mapsto \left(\frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}}\right)_{i=1,\dots,k} \end{cases}$$

   is often called the *softmax* function, and its shape *sigmoidal*. See Figure 1.3 for an illustration.

2. The case $|\mathbb{S}| = 2$ deserves special attention. Wlog (i.e. without loss of generality), let $\mathbb{S} = \{0, 1\}$, and use

$$\mathbf{P}(Y_i = 0) = \frac{e^{x_{i\cdot}\beta_{\cdot 0}}}{e^{x_{i\cdot}\beta_{\cdot 0}} + e^{x_{i\cdot}\beta_{\cdot 1}}} = \frac{1}{1 + e^{x_{i\cdot}(\beta_{\cdot 1} - \beta_{\cdot 0})}},$$

   to see that the likelihood depends on β only via $\alpha := \beta_{\cdot 1} - \beta_{\cdot 0}$.

3. For non-binary classification, this also shows that we can wlog assume that $\beta_{\cdot c^*} = 0$ for one $c^* \in \mathbb{S}$, since – with $\alpha_{\cdot c} = \beta_{\cdot c} - \beta_{\cdot c^*}$ for $c \neq c^*$ –

$$\frac{e^{x_{i\cdot}\beta_{\cdot y_i}}}{\sum_c e^{x_{i\cdot}\beta_{\cdot c}}} = \frac{e^{x_{i\cdot}(\beta_{\cdot y_i} - \beta_{\cdot c^*})}}{\sum_c e^{x_{i\cdot}(\beta_{\cdot c} - \beta_{\cdot c^*})}} = \begin{cases} \dfrac{1}{1 + \sum_{c\neq c^*} e^{x_{i\cdot}\alpha_{\cdot c}}}, & y_i = c^*, \\ \dfrac{e^{x_{i\cdot}\alpha_{\cdot y_i}}}{1 + \sum_{c\neq c^*} e^{x_{i\cdot}\alpha_{\cdot c}}}, & y_i \neq c^*. \end{cases}$$

[7] Recall $\sim$ from Remark B.2.

Let us turn to the task of estimating β in the logistic regression model. Here, the minimizer of the (log-)likelihood can not be computed analytically. However, we will still be able to show uniqueness of the maximizer. In order to proceed, we need a lemma.

Lemma 1.12. *The Hessian of the function*

$$k : \begin{cases} \mathbb{R}^{\mathbb{S}} & \to \mathbb{R}, \\ t & \mapsto \log\Big(\sum_d e^{t_d}\Big) \end{cases}$$

is positive semi-definite. Moreover, for $c^* \in \mathbb{S}$, it is positive definite on $\{t_c \in \mathbb{R}^{\mathbb{S}} : t_{c^*} = 0\}$.

Proof. We compute the first two derivatives

$$\nabla k(t) = \Big(\frac{e^{t_c}}{\sum_d e^{t_d}}\Big)_{c\in\mathbb{S}} =: \frac{e^t}{\sum_d e^{t_d}},$$
$$\nabla^2 k(t) = \frac{\operatorname{diag}(e^t)}{\sum_c e^{t_c}} - \frac{e^t (e^t)^\top}{(\sum_c e^{t_c})^2} = \operatorname{diag}(p) - pp^\top$$

with $p = (p_c)_{c\in\mathbb{S}}$ and $p_c := \frac{e^{t_c}}{\sum_d e^{t_d}}$. Now, let $u \in \mathbb{R}^{\mathbb{S}}$. Then, for $u \in \mathbb{R}^{\mathbb{S}}$,

$$u^\top \nabla^2 k(t) u = \sum_{cd} (\delta_{cd} p_c - p_c p_d) u_c u_d = \sum_c p_c u_c^2 - \Big(\sum_c p_c u_c\Big)^2 \geq 0$$

by the Cauchy-Schwartz inequality – see Lemma B.1. Note that the last inequality is strict unless $c \mapsto u_c$ is constant. This shows positive semi-definiteness of the Hessian. Moreover, on $\{t_c \in \mathbb{R}^{\mathbb{S}} : t_{c^*} = 0\}$, we may only use u with $u_{c_*} = 0$. This also shows positive definiteness on this set. □

Proposition 1.13 (ML-estimator for logistic regression). *In the logistic regression model, any ML-estimator for β satisfies*

$$\sum_{i=1}^n x_{i\cdot}\Big(1_{y_i=c} - \mathbf{P}_\beta(Y_i = c)\Big) = 0, \qquad c \in \mathbb{S}. \tag{1.3}$$

Moreover, for $c^* \in \mathbb{S}$, the ML-estimator is unique in $\{\beta \in \mathbb{R}^{k\times\mathbb{S}} : \beta_{\cdot c^*} = 0\}$.

Remark 1.14 (One-hot encoding). Often, in machine learning analysis, one writes $Y \in \mathbb{S}^n$ as a *one-hot encoding*. This means that we transform Y_i into a vector $h(Y_i) \in \{0,1\}^{\mathbb{S}}$ with[8] $(h(Y_i))_j = \delta_{Y_i j}$. Using componentwise application, we write $h(Y) := (h(Y_i))_{i=1,\dots,n} \in \{0,1\}^{n\times\mathcal{C}}$. With this notation, (1.3) becomes (writing $\mathbf{P}_\beta(Y = .) := \mathbf{P}_\beta(Y_i = c))_{i=1,\dots,n,c\in\mathbb{S}}$)

$$0 = \sum_{i=1}^n x_{i\cdot}(h(Y)_{i\cdot} - \mathbf{P}_\beta(Y_i = .)) = (h(Y) - \mathbf{P}_\beta(Y = .))^\top x.$$

Proof. The log-likelihood reads

$$\ell(\beta, y) = \sum_{i=1}^n x_{i\cdot}\beta_{\cdot y_i} - \log\Big(\sum_c e^{x_{i\cdot}\beta_{\cdot c}}\Big).$$

[8] Recall $\delta_{ij} = 1$ if $i = j$ and 0 otherwise.

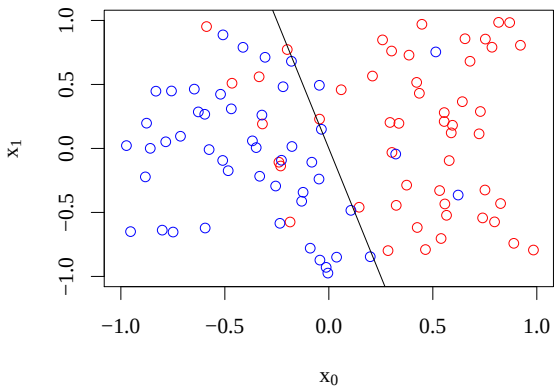


Figure 1.4: An example for binary logistic regression with $k = 2$ covariates; see Example 1.15. Blue points are in class 1, and red points are in class 0.

From this, we obtain the gradient

$$\nabla_\beta \ell(\beta, y) = \Big(\sum_{i=1}^n x_{i\cdot} 1_{y_i = c} - \frac{x_{i\cdot} e^{x_{i\cdot}.\beta_{\cdot c}}}{\sum_c e^{x_{i\cdot}.\beta_{\cdot c}}} \Big)_{c \in \mathbb{S}} = \Big(\sum_{i=1}^n x_{i\cdot} \Big(1_{y_i = c} - \mathbf{P}_\beta(Y_i = c) \Big) \Big)_{c \in \mathbb{S}}.$$

For the Hessian, we write with $t_{ic}(\beta) := x_{i\cdot}.\beta_{\cdot c}$

$$\begin{aligned} \nabla^2_\beta \ell(\beta, y) &= -\sum_{i=1}^n \nabla^2_\beta \log \Big(\sum_d e^{t_{id}(\beta)} \Big) = \sum_{i=1}^n x_{i\cdot} x_{i\cdot} \Big(\nabla^2_t \log \Big(\sum_d e^{t_{id}(\beta)} \Big) \Big) \\ &= -\sum_{i=1}^n x_{i\cdot} x_{i\cdot} \big(\operatorname{diag}(p_i) - p_i p_i^\top \big) \end{aligned}$$

with $p_i = \Big(\frac{e^{x_{i\cdot}.\beta_{\cdot c}}}{\sum_d e^{x_{i\cdot}.\beta_{\cdot d}}} \Big)_{c \in \mathbb{S}}$. This shows as in the proof of Lemma 1.12 that ℓ is negative definite, i.e. any solution of (1.3) maximizes ℓ. □

Example 1.15. Let us simulate some data. For binary logistic regression, we classify i into class 1 if $\mathbf{P}(Y_i = 1) \geq \frac{1}{2}$ or $e^{x_{i\cdot}\beta} \leq 1$. This is equivalent to $x_{i\cdot}\beta \leq 0$, which is a separating hyperplane in $\mathbb{S}$. For $\beta_0 = 4, \beta_1 = 1$, we obtain the line $x_1 = -4x_0$. For the R-implementation, we first define all variables:

```
beta0 = 4
beta1 = 1

x0 = 2*runif(100)-1
x1 = 2*runif(100)-1
p = 1 / (1+ exp(beta0 * x0 + beta1 * x1))
y = (runif(100) < p) + 0 # +0 transforms in an int

plot(c(-1,1), c(-1,1), type="n", xlab=expression(x[0]), ylab=expression(x[1]))
```

The last line produces an exmpty plot. For the output of points and decision boundary, we have

```
points(x0[y==0], x1[y==0], col="red")
points(x0[y==1], x1[y==1], col="blue")
abline(0, -4)
```

1.3 Statistical errors in machine learning

Let us end this introduction by some general thoughts, again comaring concepts from statistics with concepts from machine learning. In particular, we will discuss the important topic of *training error* and *test error*. Before we come to these, we introduce statistical decision theory, which is based on statistical models.

Definition 1.16 (Decision theory). *Let $\mathbb{S}, \mathbb{O}, \mathbb{O}'$ be as in Definition 1.1 and consider a statistical model $(X, (\mathbf{P}_\vartheta)_{\vartheta\in\Theta})$. We call any set $\mathbb{\Gamma}$ a* decision space. *Every map $d : \mathbb{S} \to \mathbb{\Gamma}$ is called a* decision. *A* loss *is a function $l : \mathbb{O}' \times \mathbb{\Gamma} \to \mathbb{R}_+$. The* risk *of a decision d is*

$$r_d(\theta) := \mathbf{E}_\theta\big[l(\theta', d(X))\big].$$

Example 1.17 (Regression). For estimating β in regression, we have (recall that x is fixed, and Y is random) $\mathbb{\Gamma} = \mathbb{R}^k$

$$d(Y) := \hat\beta := (x^\top x)^{-1}x^\top Y,$$

as well as the loss $l(\beta, \hat\beta) := (\beta - \hat\beta)^\top(\beta - \hat\beta)$. This leads to the risk

$$r_d(\beta) = \mathbf{E}_\theta\big[l(\beta, \hat\beta)\big] = \sigma^2\mathrm{tr}((x^\top x)^{-1}).$$

In particular, the risk vanishes if $\mathrm{tr}((x^\top x)^{-1}) \xrightarrow{n\to\infty} 0$.
In order to see this, note that $\mathbf{E}_\beta[Y] = x\beta$, hence

$$\mathbf{E}_\beta[\hat\beta] = (x^\top x)^{-1}x^\top(x\beta) = \beta,$$

so (writing $\mathbf{COV}_\beta[\hat\beta, \hat\beta] := (\mathbf{COV}_\beta[\hat\beta_i, \hat\beta_j])_{ij}$),

$$\begin{aligned}\mathbf{COV}_\beta[\hat\beta, \hat\beta] &= ((x^\top x)^{-1}x^\top)\mathbf{COV}_\beta[Y, Y]x(x^\top x)^{-1}) \\ &= ((x^\top x)^{-1}x^\top)\mathbf{COV}_\beta[\varepsilon, \varepsilon]x(x^\top x)^{-1}) \\ &= ((x^\top x)^{-1}x^\top)\sigma^2 Ix(x^\top x)^{-1}) = \sigma^2(x^\top x)^{-1}.\end{aligned}$$

This leads to

$$\mathbf{E}_\theta[l(\beta, \hat\beta)] = \sum_{i=1}^n \mathbf{E}[(\beta_i - \hat\beta_i)^2] = \sum_{i=1}^n \mathbf{V}[(\hat\beta_i)] = \sigma^2\sum_{i=1}^n (x^\top x)^{-1}_{ii} = \sigma^2\mathrm{tr}((x^\top x)^{-1}).$$

□

Rather than comparing loss and risk functions of different methods, in machine learning, one distinguishes between the *training error* and the *test error* of any method, both of which are also based on some loss function. The main idea, however, is that there is some underlying (statistical) model, which needs to be trained and subsequently is applied to independent data, usually called test data. In other words, when computing the error, the loss (or something related) can be computed either on the data which was used in the statistical model (training data), or on independent data which was not used for estimating β (test data). Clearly, these are two different things. But how does this fit into the framework of decision theory? We will answer this question only for the regression model at the moment.

Remark 1.18 (Training and test error in linear regression). For some data (x, Y), we have now estimated β via Theorem 1.3. Usually, the loss function is quadratic; see Remark 1.17. However, to be honest, the main point of a regression is not to see if the estimated parameter $\hat{\beta}$ fits well with the true parameter β. Rather, regression is used in order to obtain useful predictions Y for some new, independent x we have observed. So, let (x_T, Y_T) be some test data with $x_T \in \mathbb{R}^k$ and $Y_T \sim N(x_T\beta, \sigma^2)$, which is independent from (x, Y) (referred to as training data). Then, using Proposition B.5,

$$\hat{Y} := x_T(x^\top x)^{-1}x^\top Y \sim N(x_T\beta, \sigma^2 x_T(x^\top x)^{-1}x_T^\top),$$

and thus we can compute the expected test error

$$\mathbf{E}[(Y_T - \hat{Y}_T)^2 = \mathbf{V}[Y_T] + \mathbf{V}[\hat{Y}_T] = \sigma^2(1 + x_T(x^\top x)^{-1}x_T^\top).$$

In contrast, let t be one index of the training set and $\hat{Y}_t := x_{t\cdot}\hat{\beta} = x_{t\cdot}(x^\top x)^{-1}x^\top Y$, then

$$\begin{aligned}(e_t^\top - x_{t\cdot}(x^\top x)^{-1}x^\top)(e_t - x(x^\top x)^{-1}x_{t\cdot}^\top) &= \big((I - x(x^\top x)^{-1}x^\top)(I - x(x^\top x)^{-1}x^\top)\big)_{tt} \\ &= (I - x(x^\top x)^{-1}x^\top)_{tt},\end{aligned}$$

hence

$$Y_t - \hat{Y}_t = (e_t - x_{t\cdot}(x^\top x)^{-1}x^\top)Y \sim N(0, (1 - x_{t\cdot}(x^\top x)^{-1}x_{t\cdot}^\top)\sigma^2),$$

so,

$$\mathbf{E}[(Y_t - \hat{Y}_t)^2 = (1 - x_{t\cdot}(x^\top x)^{-1}x_{t\cdot}^\top)\sigma^2.$$

Hence, this shows that the training error is usually smaller than the test error (at least on average).

Example 1.19 (Training and test error in regression). In order to illustrate our calculations on linear regression, we simulate data (x, Y), as well as test data (x_T, Y_T), and compute training and test error. Here, we use quadratic regression and data with $Y = x + x^2 + \varepsilon$ with $\varepsilon \sim N(0, 0.1)$ and $n = 12$. When estimating $\beta = (0, 1, 1)$, we can allow for polynomials of any higher order k as well. When displaying the order of the allowed polynomial against the training and test error in Figure 1.5, we see that the training error is reduced (recall there are only $n = 12$ datapoints in the training set). However, the test error (which uses independent data) increases dramatically starting with $k = 6$. Also, we note that $k = 1$ is in fact a too simple regression model, with large training and test errors. We can also compare (for $k = 5$) the fit of Y to the estimated function of x. We see that Y is more accurately approximating the data only in the area where some data is present, but outside of this area the behavior of true and predicted function is quite different.

2 Optimization

Parts of this chapter are based on [7].

In linear regression, we found an estimator of the regression parameter β by minimizing

$$\beta \mapsto RSS(\beta) := \sum_{i=1}^{n}(Y_i - x_{i\cdot}\beta)^2 = (Y - x\beta)^\top(Y - x\beta) = Y^\top Y - 2Y^\top x\beta + \beta^\top x^\top x\beta.$$

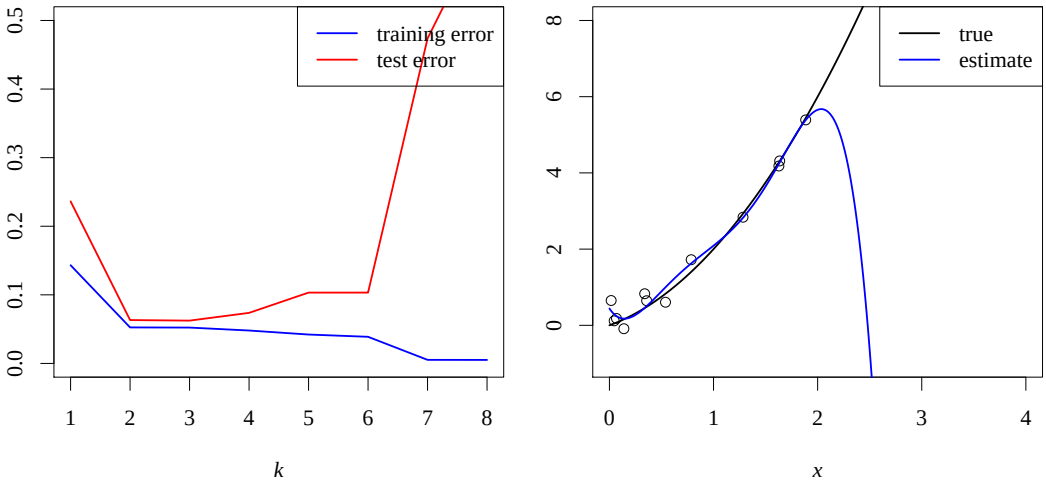


Figure 1.5: Training and test error for simulated regression data with $Y = x + x^2 + \varepsilon$ and $\varepsilon \sim N(0, 0.1)$. The k in the left plot refers to the allowed order of the polynomial (in x), which is used in the regression.

This was a special case when the corresponding minimum could be computed analytically. In this chapter, we will develop ideas for minimization if an analytical solution is out of reach. So, generally, consider the problem of finding a minimum of $f : E \subseteq \mathbb{R}^d \to \mathbb{R}$, and compute it using some computer program. We will treat here the simplest case of a function f, which is smooth and strongly convex. In applications, a frequent choice of optimization algorithms is stochastic gradient descent, which we will discuss in Section 2.4.

2.1 Convexity

It turns out that maxima (or minima) are unique if f is strictly convex. Let us now come to the precise convextity setting we will use.

Definition 2.1 (Convexity). *1. A set $\Theta \subseteq \mathbb{R}^d$ is* convex *if*

$$(x, y \in \Theta, \lambda \in [0, 1]) \Longrightarrow (\lambda x + (1 - \lambda)y \in \Theta).$$

2. Let $\Theta \subseteq \mathbb{R}^d$ be convex and $f : \Theta \to \mathbb{R}$. Then, f is convex, *if*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y), \qquad x, y \in \Theta, \lambda \in (0, 1).$$

If even "$<$" holds for any choice of $x, y \in \Theta$ and $\lambda \in (0, 1)$, f is strictly convex.

We start with the simple one-dimensional case.

Proposition 2.2 (Convexity for $d = 1$). *1. Some $\Theta \subseteq \mathbb{R}$ is convex iff it is an interval. Let $\Theta \subseteq \mathbb{R}$ be an interval.*

2. Some $f : \Theta \to \mathbb{R}$ is (strictly) convex iff $y \mapsto \frac{f(y)-f(x)}{y-x}$ is non-decreasing (increasing) for all $x \in \Theta$.

3. Some $f \in \mathcal{C}^1(\Theta)$ is (strictly) convex iff f' is non-decreasing (increasing).

4. *Some $f \in \mathcal{C}^2(\Theta)$ is (strictly) convex $f'' \geq 0$ ($f'' > 0$).*

Proof. 1. is clear. For 2.–4., we only show the assertions for convex functions (but not for strictly convex functions). 2. $\Rightarrow$: Assume that there are $x, y, z \in \Theta$ with $z < y$ and $\frac{f(z)-f(x)}{z-x} > \frac{f(y)-f(x)}{y-x}$. By symmetry, assume that $x < z$. (If this is not the case, change the roles of x and z.) Take $\lambda \in (0,1)$ with $z = \lambda x + (1-\lambda)y$. Then, the inequality holds iff

$$\frac{f(\lambda x + (1-\lambda)y) - f(x)}{1-\lambda} > f(y) - f(x) \text{ or iff } f(\lambda x + (1-\lambda)y) > \lambda f(x) + (1-\lambda)f(y).$$

This shows that f cannot be convex. $\Leftarrow$: Follow the steps of $\Rightarrow$ in reverse order.
3. If $f \in \mathcal{C}^1(\Theta)$ is convex and $y > x$, by 2.,

$$f'(y) = \lim_{h \to 0} \frac{f(y) - f(y-h)}{h} \geq \frac{f(y) - f(x)}{y - x} \geq \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = f'(x),$$

i.e. f' is increasing. Reversely, if f' is increasing and $x < z < y$,

$$\frac{f(y) - f(x)}{y - x} = \frac{\int_x^y f'(w)dw}{y - x} \geq \frac{\int_z^y f'(w)dw}{z - x} = \frac{f(z) - f(x)}{z - x},$$

and f is convex by 2.
4. follows from 3. since a differentiable function is non-decreasing iff its derivative is non-negative. □

Theorem 2.3 (Convex functions in $\mathbb{R}^d$). *Let $\Theta \subseteq \mathbb{R}^d$ be* convex *and $f : \Theta \to \mathbb{R}$. Then, the following are equivalent:*

1. *f is convex.*
2. *For each $x \in \Theta$ and $z \in \mathbb{R}^d$, the function $g : t \mapsto f(x + tz)$ (defined on $\{t \in \mathbb{R} : x + tz \in \Theta\}$) is convex.*

If $f \in \mathcal{C}^1(\Theta)$, the following is also equivalent:

3. *$f(y) \geq f(x) + \nabla f(x)^\top (y - x)$ for all $x, y \in \Theta$.*
4. *$(\nabla f(y) - \nabla f(x))^\top (y - x) \geq 0$ for all $x, y \in \Theta$.*

If $f \in \mathcal{C}^2(\Theta)$, the following is also equivalent:

5. *$\nabla^2 f(x)$ is positive semi-definite for all $x \in \Theta$.*

Proof. 1. $\Rightarrow$ 2.: We write for $\lambda \in [0,1]$

$$f(x + (\lambda s + (1-\lambda)t)z) = f(\lambda(x + sz) + (1-\lambda)(x + tz)) \leq \lambda f(x + sz) + (1-\lambda) f(x + tz),$$

which shows the convexity of $t \mapsto f(x + tz)$. 2. $\Rightarrow$ 1. Assume that there are $x, y \in \Theta$ and $\lambda \in (0,1)$ with $f(\lambda x + (1-\lambda)y) > \lambda f(x) + (1-\lambda) f(y)$ and that $t \mapsto f(x + ty)$ is convex. Take $z = x - y$ and write

$$\lambda f(x) + (1-\lambda) f(y) < f(y + \lambda z) = f(y + (\lambda 1 + (1-\lambda)0)z) \leq \lambda f(x) + (1-\lambda) f(y),$$

a contradiction.
1. $\Rightarrow$ 3.: We write, using the defining property of the gradient (A.1) in the first equality

$$0 = \lim_{\lambda \to 0} \frac{f(\lambda y + (1-\lambda)x) - f(x)}{\lambda} - \nabla f(x)^\top (y-x) \leq f(y) - f(x) - \nabla f(x)^\top (y-x).$$

3. $\Rightarrow$ 4.: By symmetry, we have that both,

$$f(y) - f(x) - \nabla f(x)^\top (y-x) \geq 0 \text{ and } f(x) - f(y) - \nabla f(y)^\top (x-y) \geq 0.$$

Adding both inequalities gives the result.
4. $\Rightarrow$ 2.: For $x \in \mathbb{\Theta}$ and $z \in \mathbb{R}^d$, let $y := x + z$ which is in $\mathbb{\Theta}$ wlog. Define $g(t) := f(x + tz)$. Now, for $0 \leq s < t \leq 1$, we have[9]

$$g'(t) - g'(s) = \nabla f(x+tz)^\top z - \nabla f(x+sz)^\top z = \frac{1}{t-s}\Big(\nabla f(x+tz)^\top - \nabla f(x+sz)^\top\Big)(t-s)z \geq 0,$$

which shows that g' is increasing, hence g is convex by Proposition 2.2.
2. $\Leftrightarrow$ 5.: Since $f \in \mathcal{C}^2(\mathbb{\Theta})$, we also have that g'' exists and is continuous, and can be computed by the chain rule. We find

$$g'(t) = \sum_{i=1}^d \frac{\partial f}{\partial z_i}(x+tz) z_i = \nabla f(x+tz)^\top z,$$

$$g''(t) = \sum_{i,j=1}^d \frac{\partial^2 f}{\partial z_i \partial z_j}(x+tz) z_i z_j z^\top = z^\top \nabla^2 f(x+tz)^\top z.$$

Then, if g is convex (for all x, z), we find that $g'' \geq 0$, which implies that $\nabla^2 f(x)$ is positive semi-definite for all $x \in \mathbb{\Theta}$. Conversely, if $\nabla^2 f(x)$ is positive semi-definite for all $x \in \mathbb{\Theta}$, we see from this equality that $g'' \geq 0$ and g is convex by Proposition 2.2. □

Corollary 2.4. *Let $\mathbb{\Theta} \subseteq \mathbb{R}^d$ be open and convex and $f \in \mathcal{C}^1(\mathbb{\Theta})$ be convex. For $x \in \mathbb{\Theta}$, the following are equivalent:*

1. *x is local minimum of f.*
2. *x is global minimum of f.*
3. *$\nabla f(x) = 0$.*

Proof. 2. $\Rightarrow$ 1. $\Rightarrow$ 3.: clear. For 3. $\Rightarrow$ 2., recall from that Theorem 2.3.3 that $f(y) \geq f(x)$ for all $y \in \mathbb{\Theta}$, which implies that x is the global minimum. □

We now come to a notion which is stronger the strict convextiy and needed in the analysis of gradient descent optimization algorithms.

Definition 2.5. *Let $\mathbb{\Theta} \subseteq \mathbb{R}^d$ be convex and $m > 0$. Then,[10] $f \in \mathcal{C}^1(\mathbb{\Theta})$ is* strongly convex (with parameter m)*, if*

$$f(y) \geq f(x) + \nabla f(x)^\top (y-x) + \tfrac{m}{2}||y-x||^2, \qquad x, y \in \mathbb{\Theta}.$$

Corollary 2.6. *Let $\mathbb{\Theta} \subseteq \mathbb{R}^d$ be open and convex and $f \in \mathcal{C}^1(\mathbb{\Theta})$. If f is strongly convex with parameter $m > 0$, then f is strictly convex.*

Proof. Exercise. □

[9] Here, we use the chain rule from the first year courses.
[10] In the sequel, we will use the euclidean norm $||.|| : x \mapsto (x_1^2 + \cdots + x_d^2)^{1/2}$.

2.2 Smoothness

For convergence of numerical methods, we need to restrict how much the gradient can change. For this, we need the notion of smoothness.

Definition 2.7. *Let $\mathbb{O} \subseteq \mathbb{R}^d$. Then $f \in \mathcal{C}^1(\mathbb{O})$ is called* s-smooth*, if*

$$||\nabla f(y) - \nabla f(x)|| \leq s||x - y||.$$

Lemma 2.8. *If f is s-smooth, then*

1. *If $f \in \mathcal{C}^2(\mathbb{O})$, $z^\top \nabla^2 f(x) z \leq s||z||^2$,* $x \in \mathbb{O}, z \in \mathbb{R}^d$,
2. $f(y) \leq f(x) + \nabla f(x)^\top (y - x) + \frac{s}{2}||y - x||^2$, $x, y \in \mathbb{O}$,
3. $f(x - \frac{1}{s}\nabla f(x)) \leq f(x) - \frac{1}{2s}||\nabla f(x)||^2$, $x \in \mathbb{O}$.

If x^* is the unique minimizer of f, then

4. $f(x) - f(x^*) \geq \frac{1}{2s}||\nabla f(x)||^2$, $x \in \mathbb{O}$.

Proof. 1. We write using the Cauchy-Schwartz inequality

$$\int_0^t z^\top \nabla^2 f(x + sz) z ds = (\nabla f(x) - \nabla f(x + tz))^\top z \leq ||\nabla f(x) - \nabla f(x + tz)|| \cdot ||z|| \leq st||z||^2,$$

Dividing by t and letting $t \to 0$ gives the assertion.
2. Similarly, with $z = y - x$,

$$\begin{aligned} f(y) &= f(x) + \int_0^1 \nabla f(x + tz)^\top z dt \\ &= f(x) + \nabla f(x)^\top z + \int_0^1 (\nabla f(x + tz) - \nabla f(x))^\top z dt \\ &\leq f(x) + \nabla f(x)^\top z + \int_0^1 ||\nabla f(x + tz) - \nabla f(x)|| \cdot ||z|| dt \\ &\leq f(x) + \nabla f(x)^\top z + s \int_0^1 t||z||^2 dt = f(x) + \nabla f(x)^\top z + \tfrac{s}{2}||y - x||^2, \end{aligned}$$

which is the assertion. 3. follows from 2. by using $y = x - \frac{1}{s}\nabla f(x)$. Finally, 4. follows from 3. and $f(x^*) \leq f(x - \frac{1}{s}\nabla f(x))$. □

2.3 Deterministic gradient descent

For some $\mathbb{O} \subseteq \mathbb{R}^d$ open, consider the problem of finding a minimum of $f \in \mathcal{C}^1(\mathbb{O})$, i.e.

$$x^* = \text{argmin}_{x \in \mathbb{O}} f(x).$$

For finding x^*, we consider the iterative procedure starting at $x_0 \in \mathbb{O}$ and updates given by

$$x_{t+1} = x_t + \eta_t d_t$$

with $\eta_t \in \mathbb{R}$ and d_t such that $x_{t+1} \in \Theta$ for all $t = 0, 1, 2, ...$ Here, η_t is called the *learning rate*, and d_t the *descent direction* in step $t+1$, which may depend on f and $x_0, ..., x_t$. A usual choice is $d_t = -\nabla f(x_t)$, since then for small $\eta_t > 0$

$$f(x_{t+1}) \approx f(x_t) + \eta_t \nabla f(x_t) d_t = f(x_t) - \eta_t ||\nabla f(x_t)||^2,$$

which shows that $f(x_{t+1}) < f(x_t)$ unless $\nabla f(x_t) = 0$, i.e. x_t is a critical point for f. See Algorithm 1.

Algorithm 1 DETERMINISTIC GRADIENT DESCENT

INPUT: Input $f \in \mathcal{C}^1(\Theta), \nabla f, x_0, \eta_0, \eta_1, ..., \varepsilon > 0$
OUTPUT: x putative global minimum of f

```
t = 1
while t = 1 or |x_t − x_{t−1}| > ε do
    x_{t+1} = x_t − η_t ∇f(x_t)
    t = t + 1
end while
return x_{t+1}
```

We now show convergence of deterministic gradient descent for smooth and strongly convex f, if we choose a constant learning rate. As we will see, convergence is exponentially fast, leading to a small number of iterations before convergence.

Theorem 2.9 (Convergence of deterministic gradient descent for constant learning rate). *Let $\Theta \subseteq \mathbb{R}^d$ be open and convex, and $f \in \mathcal{C}^1(\Theta)$ be s-smooth and m-strongly convex with unique minimizer $x^* \in \Theta$. For $x_0 \in \Theta$ and $\frac{1}{s} \geq \eta > 0$, define the iteration*

$$x_{t+1} = x_t - \eta \nabla f(x_t).$$

If $x_0, ..., x_t \in \Theta$,

$$||x_t - x^*||^2 \leq (1 - \eta m)^t ||x_0 - x^*||^2.$$

Proof. We write, using the definition of strong convexity in the second and Lemma 2.8.4 in the third line

$$\begin{aligned}
||x_{t+1} - x^*||^2 = ||x_t - x^* - \eta \nabla f(x_t)||^2 &= ||x_t - x^*||^2 - 2\eta \nabla f(x_t)^\top (x_t - x^*) + \eta^2 ||\nabla f(x_t)||^2 \\
&\leq (1 - \eta m)||x_t - x^*||^2 - 2\eta(f(x_t) - f(x^*)) + \eta^2 ||\nabla f(x_t)||^2 \\
&\leq (1 - \eta m)||x_t - x^*||^2 - 2\eta(f(x_t) - f(x^*)) + 2\eta^2 s(f(x_t) - f(x^*)) \\
&= (1 - \eta m)||x_t - x^*||^2 - 2\eta(1 - \eta s)(f(x_t) - f(x^*)) \\
&\leq (1 - \eta m)||x_t - x^*||^2.
\end{aligned}$$

□

2.4 Stochastic gradient descent

When searching for a minimizer of $x \mapsto f(x)$, one wants to avoid too many computations. In particular, $f(x)$ usually depends on all items in a dataset, which might be large. Frequently, one way out is a stochastic approach by using only a few items in the iteration scheme rather

than all. Precisely, assume that there exists (for some finite $\mathbb{I}$) some $\mathbb{I}$-valued random variable Y and a family $(f_y)_{y\in\mathbb{I}}$ in $\mathcal{C}^1(\mathbb{O})$ with

$$f(x) = \mathbf{E}[f_Y(x)]. \tag{2.1}$$

For ∇f, we assume that ∇f_Y is such that

$$\nabla f(x) = \mathbf{E}[\nabla f_Y(x)].$$

(This is usually the case since it only requires that $\mathbf{E}[.]$ and ∇ commute by linearity of $\mathbf{E}[.]$.)

Example 2.10 (Sums and the mini-batch size). The most frequent example is an f of the form

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{2.2}$$

for appropriate $f_1, ..., f_n$. Here, we can use $\mathbb{I} = [1:n] := \{1, ..., n\}$ and let[11] $Y \sim U([1:n])$ and note that $f_Y(x)$ equals $f_i(x)$ iff $Y = i$. Then,

$$\mathbf{E}[f_Y(x)] = \tfrac{1}{n}\sum_{i=1}^{n} f_i(x) = f(x)$$

and $\mathbf{E}[\nabla f_Y(x)] = \nabla f(x)$ follows by linearity. Note that Y can also be chosen differently. Let Y be uniformy distributed on the subsets of $[1:n]$ of size $1 \le k \le n$ and

$$f_Y(x) = \frac{1}{k}\sum_{i\in Y} f_i(x).$$

This also satisfies (2.1), as will be shown in an exercise. One speaks of a mini-batch size of size k here.

In order to describe stochastic gradient descent, let $Y_1, Y_2, ...$ iid such that (2.2) holds. For finding the minimizer x^* of f, we consider the iterative procedure $X_0 = x_0 \in \mathbb{O}$ and updates given by

$$X_{t+1} = X_t - \eta_t \nabla f_{Y_t}(X_t).$$

This is given in Algorithm 2.
We now show convergence of stochastic gradient descent for strongly convex f, but have to assume that the learning rate decreases over time.

Theorem 2.11 (Convergence of stochastic gradient descent). *Let $\mathbb{O} \subseteq \mathbb{R}^d$ be open and convex, and $f = \frac{1}{n}\sum_{i=1}^n f_i \in \mathcal{C}^1(\mathbb{O})$, where $f_1, ..., f_n$ are s-smooth and m-strongly convex with $\mathbf{V}[\nabla f_Y(x)] \le r$, and f has unique minimizer $x^* \in \mathbb{O}$. For $X_0 = x_0 \in \mathbb{O}$ and $\frac{1}{s} \ge \eta_t > 0$, define the iteration*

$$X_{t+1} = X_t - \eta_t \nabla f_{Y_t}(X_t).$$

If $X_0, ..., X_t \in \mathbb{O}$, then

$$\mathbf{E}[||X_{t+1} - x^*||^2] \le (1 - \eta_t m)\mathbf{E}[||X_t - x^*||^2] + \eta_t^2 r.$$

In particular, let $t^* \ge 1$ be such that $\eta_t = \frac{2}{m(t+t^*)} \le \frac{1}{s}$ for all t. Then,

$$\mathbf{E}[||X_t - x^*||^2] \le \frac{\max\{t^*||x_0 - x^*||^2, 4r/m^2\}}{t + t^*}.$$

[11] Here, $U([1:n])$ is the uniform distribution on $[1:n]$.

Algorithm 2 Stochastic gradient descent for sums, mini-batch size 1

INPUT: $f = \frac{1}{n}\sum_{i=1}^n f_i \in \mathcal{C}^1(\mathbb{O}), \nabla f_1, ..., \nabla f_n, x_0, Y_0, Y_1, ...$ iid, $\sim U([1:n]), \eta_0, \eta_1, ..., \varepsilon > 0$
OUTPUT: x putative global minimum of f

```
t = 1
while t = 1 or |x_t − x_{t−1}| > ε do
    x_{t+1} = x_t − η_t ∇f_{Y_t}(x_t)
    t = t + 1
end while
return x_{t+1}
```

Proof. Before we start, observe that

$$\mathbf{E}[(\nabla f_Y(x))^2] = \tfrac{1}{n}\sum_{i=1}^n (\nabla f_i(x))^2 = \mathbf{V}[\nabla f_Y(x)] + ||\nabla f(x)||^2.$$

We start as in the proof of Theorem 2.9 and write, again using the definition of strong convexity and Lemma 2.8.4 (for f) in the third line

$$\begin{aligned}
\mathbf{E}[||X_{t+1} - x^*||^2] &= \mathbf{E}[||X_t - x^* - \eta_t \nabla f_{Y_t}(X_t)||^2] \\
&= \mathbf{E}[||X_t - x^*||^2] - 2\eta_t \mathbf{E}[\nabla f_{Y_t}(X_t)^\top (X_t - x^*)] + \eta_t^2 \mathbf{E}[(\nabla f_{Y_t}(X_t))^2] \\
&\leq (1 - \eta_t m)\mathbf{E}[||X_t - x^*||^2] - 2\eta_t \mathbf{E}[f_{Y_t}(X_t) - f_{Y_t}(x^*)] + \eta_t^2 \mathbf{E}[(\nabla f_{Y_t}(X_t))^2] \\
&\leq (1 - \eta_t m)\mathbf{E}[||X_t - x^*||^2] - 2\eta_t (1 - \eta_t s)\mathbf{E}[f_{Y_t}(X_t) - f_{Y_t}(x^*)] + \eta_t^2 \mathbf{E}\big[\mathbf{V}[\nabla f_{Y_t}(X_t)|X_t]\big] \\
&\leq (1 - \eta_t m)\mathbf{E}[||X_t - x^*||^2] + \eta_t^2 r.
\end{aligned}$$

For the convergence rate in the case $\eta_t = \frac{2}{m(t+t^*)}$, we set $a := \max\{t^*||x_0 - x^*||^2, 4r/m^2\}$ and proceed by induction. For $t = 0$, the assertion is clear. Assume it holds for some t. Then, by the first result,

$$\begin{aligned}
\mathbf{E}[||X_{t+1} - x^*||^2] &\leq (1 - \tfrac{2}{t+t^*})\mathbf{E}[||X_t - x^*||^2] + \frac{4r}{m^2(t+t^*)^2} \\
&\leq (1 - \tfrac{2}{t+t^*})\frac{a}{t+t^*} + \frac{a}{(t+t^*)^2} = a\Big(\frac{1}{t+t^*} - \frac{1}{(t+t^*)^2}\Big) \leq \frac{a}{t+t^*+1},
\end{aligned}$$

where we have used that $(z+1)\Big(\frac{1}{z} - \frac{1}{z^2}\Big) = 1 + \frac{1}{z} - \frac{z+1}{z^2} = 1 - \frac{1}{z^2} < 1$ for all z in the last step. □

Remark 2.12 (Stochastic gradient descent with momentum). Consider a ball rolling down a hill. If it hits some hill, it will not turn immediately, but carry its *momentum* forward but change its direction a bit. This is the idea of stochastic gradient descent with momentum. Here, the update rule is

$$X_{t+1} = X_t - \eta_t \nabla f_{Y_t}(X_t) + \beta_t (X_t - X_{t-1}),$$

for some $\eta_1, \eta_2, ..., \beta_1, \beta_2, ...$ Many of the modern methods use this kind of heuristics. In many cases, a typical value is $\beta_t = 0.9$.

Remark 2.13 (Quadratic programming). Yet another way of finding the minimizer of some $f \in \mathcal{C}^2(\Theta)$ (for $\Theta \subseteq \mathbb{R}^d$) is as follows. If x^* is a local minimum of f, then, using a Taylor series expansion of f,

$$f(y) = f(x) + \nabla f(x)^\top (y - x) + \tfrac{1}{2}(y - x)^\top \nabla^2 f(x)(y - x).$$

In fact, the minimum of the right hand side (as a function of y) can be computed explicitely. Here, a necessary condition is

$$0 = \nabla f(x) + \nabla^2 f(x)(y - x) \text{ or } y = x - (\nabla^2 f(x))^{-1} \nabla f(x)^\top.$$

This suggests an iterative scheme, starting in some $x_0 \in \Theta$ with

$$x_{n+1} = x_n - (\nabla^2 f(x_n))^{-1} \nabla f(x_n)^\top.$$

2.5 Examples from logistic regression

See the accompanying R markdown file on pages 57ff.

3 Methods for classification

> Parts of this chapter are worked out using [2]. Theorem 3.24 is from [3]. More calculations for estimating the asymptotic errors for nearest neighbor classifiers in Section 3.5 are from [5].

We will now treat methods for classification. The important technique of logistic regression was already given in Section 1.2.

3.1 Classification and decision theory

Definition 3.1 (Classification problem). *Let $\mathbb{S}$ be finite and (X, Y) be a pair of random variables with value in $\mathbb{E} \times \mathbb{S}$ with $\mathbb{E} \subseteq \mathbb{R}^k$. Any $\psi : \mathbb{E} \to \mathbb{S}$ is called a classifier. The classification error for ψ is given by*

$$e_\psi := \mathbf{P}(\psi(X) \neq Y). \tag{3.1}$$

We will use this definition without further mention in the rest of this section.

Remark 3.2 (Classification and decision theory). Note that the above definition fits into the scheme of decision theory from Definition 1.16. Here, $\mathbb{S}' = \mathbb{R}^k \times \mathbb{S}$ and $\Theta = \Gamma = \mathbb{S}$, we set

$$d(X, Y) := \psi(X), \qquad l(Y, \psi(X)) = 1_{Y \neq \psi(X)}.$$

The risk of ψ is then the probability of misclassification since

$$\mathbf{E}[l(Y, \psi(X))] = e_\psi.$$

Definition 3.3 (Bayes classifier and its error). *Any minimizer ψ^* of $\psi \mapsto e_\psi$ from* (3.1) *is called a* Bayes classifier*. Its probability of misclassification e_{ψ^*} is called the* Bayes error*.*

The reason for the name *Bayes classifier* becomes clear with the next lemma. It states that a Bayes classifier is given by maximizing the conditional expectation of the class given the features.

Lemma 3.4. *A Bayes classifier is given by*

$$\psi^*(x) := argmax_{y\in\mathbb{S}}\mathbf{P}(Y = y|X = x).$$

If the maximum is not unique, any choice for argmax leads to a Bayes classifier.

Proof. Let $\psi : \mathbb{R}^k \to \mathbb{S}$ be an arbitrary classifier. We may write

$$\begin{aligned}\mathbf{P}(\psi^*(X) = Y|X = x) &= \sum_y \mathbf{P}(Y = y|X = x)1_{\psi^*(x)=y} \\ &\geq \sum_y \mathbf{P}(Y = y|X = x)1_{\psi(x)=y} = \mathbf{P}(\psi(X) = Y|X = x)\end{aligned}$$

and therefore

$$\begin{aligned}e_\psi &= \mathbf{P}(\psi(X) \neq Y) = \mathbf{E}[\mathbf{P}(\psi(X) \neq Y|X)] = 1 - \mathbf{E}[\mathbf{P}(\psi(X) = Y|X)] \\ &\geq 1 - \mathbf{E}[\mathbf{P}(\psi^*(X) = Y|X)] = e_{\psi^*}.\end{aligned}$$

□

Remark 3.5. The Bayes classifier ψ^* from Lemma 3.4 can be written as

$$\psi^*(X) := \operatorname{argmax}_{y\in\mathbb{S}}\delta_y(x), \qquad \delta_y(X) := \log \mathbf{P}(X = x|Y = y) + \log \mathbf{P}(Y = y).$$

Indeed: We have

$$\begin{aligned}\psi^*(x) = z &\iff \mathbf{P}(Y = z|X = x) \geq \mathbf{P}(Y = y|X = x), y \in \mathbb{S} \\ &\iff \frac{\mathbf{P}(X = x|Z = z)\mathbf{P}(Z = z)}{\mathbf{P}(X = x)} \frac{\mathbf{P}(X = x|Z = y)\mathbf{P}(Z = y)}{\mathbf{P}(X = x)}, y \in \mathbb{S}. \\ &\iff \delta_z(x) \geq \delta_y(x), y \in \mathbb{S}.\end{aligned}$$

We now turn to the simplest case of $|\mathbb{S}| = 2$.

Lemma 3.6. *Let $\mathbb{S} = \{0, 1\}$ and ψ^* a Bayes classifier as in Lemma 3.4. With*

$$\eta(X) := \mathbf{P}(Y = 1|X),$$

we then have

$$e_{\psi^*} = \mathbf{E}[\eta(X) \wedge (1 - \eta(X))].$$

In particular, $e_{\psi^*} \leq 1/2$.

Proof. We write

$$\begin{aligned}e_{\psi^*} &= \mathbf{P}(\psi^*(X) \neq Y) = \mathbf{P}(Y = 1, \eta(X) < 1/2) + \mathbf{P}(Y = 0, \eta(X) \geq 1/2) \\ &= \mathbf{E}[\mathbf{P}(Y = 1|X), \eta(X) < 1/2] + \mathbf{E}[\mathbf{P}(Y = 0|X), \eta(X) \geq 1/2] \\ &= \mathbf{E}[\eta(X), \eta(X) < 1/2] + \mathbf{E}[1 - \eta(X), \eta(X) \geq 1/2] = \mathbf{E}[\eta(X) \wedge (1 - \eta(X))].\end{aligned}$$

The last assertion follows since $x \wedge (1 - x) \leq 1/2$ for all $x \in [0, 1]$. □

Remark 3.7. The Bayes classifier from Lemma 3.6 for $|\mathbb{S}| = 2$ can also be written as

$$\eta(X) = 1(\delta(X) > k), \qquad \delta(x) = \frac{\mathbf{P}(X = x|Y = 1)}{\mathbf{P}(X = x|Y = 0)}, \qquad k = \frac{\mathbf{P}(Y = 0)}{\mathbf{P}(Y = 1)}.$$

Indeed, with this definition,

$$\begin{aligned}\mathbf{P}(Y = 1|X = x) > &\mathbf{P}(Y = 0|X = x) \iff \mathbf{P}(Y = 1, X = x) > \mathbf{P}(Y = 0, X = x)\\ &\iff \mathbf{P}(X = x|Y = 1)\mathbf{P}(Y = 1) > \mathbf{P}(X = x, Y = 0)\mathbf{P}(Y = 0)\\ &\iff \delta(x) > k.\end{aligned}$$

Remark 3.8 (Empirical risk minimization). The theory around the classification error and Bayes classifier is of little practical use since we do not know the true distribution $\mathbf{P}$ of the data we want to analyse. Therefore, we will be looking for classifiers which only depend on some $(X_1, Y_1), (X_2, Y_2), ...$, which have the same distribution as the (test) data (X, Y) we want to analyse. This leads to a sequence of classifiers $\psi_1, \psi_2, ...$ in some set $\mathcal{H}$ (e.g. classifiers linear in X), where ψ_n depends on $(X_1, Y_1), ..., (X_n, Y_n), X$. Then, rather than finding a Bayes classifier, we may ask for

$$\mathcal{H} \ni \psi_n : (\mathbb{E} \times \mathbb{S})^n \times \mathbb{E} \to \mathbb{S} \text{ which minimizes } \frac{1}{n}\sum_{i=1}^{n} 1_{\psi((X_i,Y_i)_{i=1,...,n},X_i)\neq Y_i}.$$

This is the *empirical risk* or *training error* of ψ_n.

As for an estimation task, classifiers can be asymptotically optimal.

Definition 3.9. *A sequence of classifiers $\psi_1, \psi_2, ...$ (which might be random) is* consistent *if $\mathbf{P}(e_{\psi_n} - e_{\psi^*} > \varepsilon) \xrightarrow{n\to\infty} 0$ for all $\varepsilon > 0$.*

3.2 Nearest centroid classifier

In plots for classification (see Section 3.6), one often sees distinct areas of the feature space which are prevalent for some class. Then, one could normalize each class by representing the class by its centroid. Then, if we are given new data, we classify it to the nearest centroid. Before we come to a formal definition, we study a main example which we treat here in a general case.

Example 3.10 (Gaussian model). We consider the case of Gaussian random variables as features. Precisely, conditional on $Y = y$, we have $X \sim N(\mu_y, \Sigma_y)$, $y \in \mathbb{S}$. Below, we will use the *Mahalanobis distance*

$$||x||^2_\Sigma := x^\top \Sigma^{-1} x.$$

1. *The general heteroskedastic case:* Define

$$\delta_y(x) := -||x - \mu_y||^2_{\Sigma_y} - \log|\det \Sigma_y| + 2\log \mathbf{P}(Y = y).$$

From Remark 3.5, we can readily compute the Bayes classifier since the definition of δ_y implies that $\frac{1}{2}\delta_y(x) = \log \mathbf{P}(X = x|Y = y) + \mathbf{P}(Y = y)$. It is given by

$$\psi^*(x) = \text{argmax}_{y\in\mathbb{S}}\delta_y(x). \tag{3.2}$$

Since $x \mapsto \delta_y(x)$ is quadratic, decision boundaries are quadratic functions in this case.

2. *The homoskedastic case:* However, if we restrict to the case $\Sigma := \Sigma_y$ for all $y \in \mathbb{S}$, i.e. the covariance is the same in all classes, we find a linear decision boundary (since the $x^\top \Sigma^{-1} x$-term is the same in all $\delta_y(x)$'s). With

$$\gamma_y(x) = -\mu_y^\top \Sigma^{-1} \mu_y + 2\mu_y^\top \Sigma^{-1} x + 2 \log \mathbf{P}(Y = y)$$

we have the Bayes classifier

$$\psi^*(x) = \text{argmax}_{y \in \mathbb{S}} \gamma_y(x). \tag{3.3}$$

3. *The homoskedastic case for $|\mathbb{S}| = 2$:* If we only have two classes, $\mathbb{S} = \{0, 1\}$, and using Remark 3.7, it is possible to compute the Bayes error by

$$a = 2\Sigma^{-1}(\mu_1 - \mu_0), \qquad b = (\mu_0 - \mu_1)^\top \Sigma^{-1}(\mu_0 + \mu_1), \qquad c = 2 \log \frac{\mathbf{P}(Y = 1)}{\mathbf{P}(Y = 0)}.$$

Here, we can write

$$\psi^*(x) = \begin{cases} 1, & a^\top x + b + c > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

For $Z \sim N(0, 1)$, let $\Phi : x \mapsto \mathbf{P}(Z \leq x)$. The Bayes error is then given by

$$\varepsilon_{\psi^*} = \mathbf{P}(Y = 1)\Phi\Big(\frac{-c - \frac{1}{2}||\mu_1 - \mu_0||_\Sigma^2}{||\mu_1 - \mu_0||_\Sigma}\Big) + \mathbf{P}(Y = 0)\Phi\Big(\frac{c - \frac{1}{2}||\mu_1 - \mu_0||_\Sigma^2}{||\mu_1 - \mu_0||_\Sigma}\Big).$$

In particular, for $c = 0$,

$$= \Phi\Big(-\tfrac{1}{2}||\mu_1 - \mu_0||_\Sigma\Big).$$

In order to see this, note that

$$\begin{aligned} a^\top \mu_0 + b &= (\mu_1 - \mu_0)^\top \Sigma\Big(\mu_0 - \frac{\mu_0 + \mu_1}{2}\Big) = -\tfrac{1}{2}||\mu_1 - \mu_0||_\Sigma^2, \\ a^\top \mu_1 + b &= (\mu_1 - \mu_0)^\top \Sigma\Big(\mu_1 - \frac{\mu_0 + \mu_1}{2}\Big) = \tfrac{1}{2}||\mu_1 - \mu_0||_\Sigma^2, \\ a^\top \Sigma a &= (\mu_1 - \mu_0)^\top \Sigma(\mu_1 - \mu_0) = ||\mu_1 - \mu_0||_\Sigma^2. \end{aligned}$$

So, for $X_i \sim N(\mu_i, \Sigma)$, we find with Proposition B.5 that

$$a^\top X_0 + b \sim N(-\tfrac{1}{2}||\mu_1 - \mu_0||_\Sigma^2, ||\mu_1 - \mu_0||_\Sigma^2) \qquad \text{or} \qquad \frac{a^\top X_0 + b + \frac{1}{2}||\mu_1 - \mu_0||_\Sigma^2}{||\mu_1 - \mu_0||_\Sigma} \sim N(0, 1),$$

$$a^\top X_1 + b \sim N(\tfrac{1}{2}||\mu_1 - \mu_0||_\Sigma^2, ||\mu_1 - \mu_0||_\Sigma^2) \qquad \text{or} \qquad \frac{a^\top X_1 + b - \frac{1}{2}||\mu_1 - \mu_0||_\Sigma^2}{||\mu_1 - \mu_0||_\Sigma} \sim N(0, 1).$$

Hence,

$$\begin{aligned} \varepsilon_{\psi^*} &= \mathbf{P}(\psi^*(X) \neq Y) = \mathbf{P}(Y = 1)\mathbf{P}(\psi^*(X_1) = 0) + \mathbf{P}(Y = 0)\mathbf{P}(\psi^*(X_0) = 1) \\ &= \mathbf{P}(Y = 1)\mathbf{P}(a^\top X_1 + b + c < 0) + \mathbf{P}(Y = 0)\mathbf{P}(a^\top X_0 + b + c > 0) \\ &= \mathbf{P}(Y = 1)\mathbf{P}\Big(Z < -\frac{c}{||\mu_1 - \mu_0||_\Sigma} - \tfrac{1}{2}||\mu_1 - \mu_0||_\Sigma\Big) \\ &\qquad + \mathbf{P}(Y = 0)\mathbf{P}\Big(Z > -\frac{c}{||\mu_1 - \mu_0||_\Sigma} + \tfrac{1}{2}||\mu_1 - \mu_0||_\Sigma\Big). \end{aligned}$$

Definition 3.11 ((Empirical) nearest centroid classifier). *Assume we have training data $(X_i, Y_i)_{i=1,...,n}$, which are independent and identically distributed with values in $\mathbb{E} \times \mathbb{S}$ for some $\mathbb{E} \subseteq \mathbb{R}^k$ open, and that X_1 has some density on $\mathbb{E}$. Define the centroids*

$$\widehat{\mu}_y := \frac{1}{N_y} \sum_{i:Y_i=y} X_i \quad \textit{with} \quad N_y := |\{i : Y_i = y\}|.$$

Then, the classifier

$$\psi^{centr}((X_i, Y_i)_{i=1,...,n}, X) := \textit{argmin}_{y \in \mathbb{S}} ||X - \widehat{\mu}_y||$$

is called the nearest centroid classifier.

Example 3.12 (Homoskedastic Gaussian case). For $|\mathbb{S}| = 2$, let us consider $(X_1, Y_1), ..., (X_n, Y_n), (X, Y)$ distributed as in Example 3.10.3 with $\Sigma = I$ (the unit matrix) and $\mathbf{P}(Y = 0) = \mathbf{P}(Y = 1)$. Since we can write

$$\psi^{centr}((X_i, Y_i)_{i=1,...,n}, X) = \begin{cases} 1, & ||X - \widehat{\mu}_1|| < ||X - \widehat{\mu}_0||, \\ 0, & \text{otherwise.} \end{cases},$$

this is the same as – considering (3.4) –

$$= \begin{cases} 1, & \widehat{a}_n^\top x + \widehat{b}_n > 0, \\ 0, & \text{otherwise} \end{cases}$$

with

$$\widehat{a}_n = \widehat{\mu}_1 - \widehat{\mu}_0, \qquad 2\widehat{b}_n = (\widehat{\mu}_1 - \widehat{\mu}_0)^\top (\widehat{\mu}_1 + \widehat{\mu}_0).$$

Conditional on $(X_1, Y_1), ..., (X_n, Y_n)$, we find the empirical misclassification error for $X \sim N(\mu_i, I)$ given $Y = i$ and $\mathbf{P}(Y = i) = \frac{1}{2}, i = 0, 1$

$$\begin{aligned} e_{\psi_n^{\text{centr}}} &= \frac{1}{2}\Big(\mathbf{P}(\widehat{a}_n^\top X + \widehat{b} > 0 | Y = 1) + \mathbf{P}(\widehat{a}_n^\top X + \widehat{b} a 0 | Y = 0)\Big) \\ &= \frac{1}{2}\Big(\Phi\Big(\frac{\widehat{a}_n^\top \mu_0 + \widehat{b}_n}{||\widehat{a}_n||}\Big) + \Phi\Big(-\frac{\widehat{a}_n^\top \mu_0 + \widehat{b}_n}{||\widehat{a}_n||}\Big)\Big). \end{aligned}$$

In fact, the sequence $\psi_1^{\text{centr}}, \psi_2^{\text{centr}}, ...$ is consistent, since $\widehat{\mu}_0 \xrightarrow{n\to\infty} \mu_0, \widehat{\mu}_1 \xrightarrow{n\to\infty} \mu_1$.

3.3 Discriminant analysis

The goal of discriminant analysis is to provide a way to compute class boundaries. Actually, there are two different names connected to this kind of analysis. There is Gaussian discriminant analysis (which comes in two falvors, quadratic and linear), and Fisher discriminant analysis (which leads to linear classification boundaries). All give classifiers, which we introduce and explain in this section. Actually, we have treated Gaussian discriminant analysis already in Example 3.10.

Definition 3.13 (Gaussian discriminant analysis). *Let $\mathbb{S}$ be finite.*

1. *The classifier*
$$\psi(x) = argmax_{y\in\mathbb{S}}\widehat{\delta}_y(x)$$
with $\widehat{\delta}_y(x)$ *as in* (3.2) *with* μ_y *and* Σ_y *replaced by*
$$N_y := |\{i : Y_i = y\}|, \qquad \widehat{\mu}_y = \tfrac{1}{N_y}\sum_{i:Y_i=y} X_i, \qquad \widehat{\Sigma}_y = \frac{1}{N_y - 1}\sum_{i:Y_i=y}(X_i - \widehat{\mu}_y)(X_i - \widehat{\mu}_y)^\top$$
and $\mathbf{P}(Y = y)$ *by* N_y/n *for all* $y \in \mathbb{S}$ *is called classifier for the* quadratic discriminant analysis*.*

2. *The classifier*
$$\psi(x) = argmax_{y\in\mathbb{S}}(\widehat{\gamma}_y(x))$$
with $\widehat{\gamma}_y(x)$ *as in* (3.3) *with* μ_y *and* $\mathbb{P}(Y = y)$ *replaced as above and* Σ *replaced by*
$$\widehat{\Sigma} := \frac{\sum_y (N_y - 1)\widehat{\Sigma}_y}{n - |\mathbb{S}|}$$
is called classifier for the linear discriminant analysis*.*

Remark 3.14 (Connection to logistic regression; geneartive and discriminatory models). In logistic regression, we have used the model – for some $\beta \in \mathbb{R}^{k\times\mathbb{S}}$ with $\beta_{\cdot c^*} = 0$ for one $c^* \in \mathbb{S}$ –
$$\log\frac{\mathbf{P}(Y = y|x)}{\mathbf{P}(Y = z|x)} = x_{i\cdot}(\beta_{\cdot y} - \beta_{\cdot z}),$$
i.e. logistic regression also leads to the classifier
$$\psi(x) = \text{argmax}_{y\in\mathbb{S}}(x_{i\cdot}\widehat{\beta}_{\cdot y})$$
with the ML-estimator $\widehat{\beta}$ of β, and therefore to a linear classification boundary. (Note that we may have $x_{i0} = 1$ for all i, explaining the constant term in $x \mapsto x_{i\cdot}\widehat{\beta}_{\cdot y}$.) So, the classifiers of linear discriminant analysis and logistic regression both lead to linear classification boundaries and to comparable classifiers. However, there are two important differences:

1. The way we estimate β in logistic regression is different from the estimation of the model parameters in linear discriminant analysis.

2. Logistic regression does not impose a distribution of the features X. Rather, they are treated as given, and only the class labels are assigned stochastically in the corresponding statistical model. In linear discriminant analysis, we assume a distribution on the feature vector X given the class label Y. We say that discriminant analysis uses a generative model, while logistic regression uses a discriminative model.

Now, we come to Fishers linear discriminant analysis, which eventually comes to the same classifier as in Gaussian linear discriminant analysis. We treat it here only in the case $\mathbb{S} = 2$.

Remark 3.15 (Fisher's discriminant analysis 1). For $\mathbb{S} = \{0, 1\}$ and test data (X, Y), distributed as in Example 3.10.3 (with invertible Σ), Fisher's idea is to use a linear classifier
$$\psi(x) = 1_{w^\top x>\kappa}$$

for some κ. Here, w is chosen in order to maximize (note that $\mathbf{V}[w^\top X] = w^\top \Sigma w$)

$$\begin{aligned} J(w) &= \frac{(\mathbf{E}[w^\top X|Y=1] - \mathbf{E}[w^\top X|Y=0])^2}{w^\top \Sigma w} \\ &= \frac{w^\top (\mu_1 - \mu_0)(\mu_1 - \mu_0)^\top w}{w^\top \Sigma w} \end{aligned}$$

Lemma 3.16. *Maximizers of $w \mapsto J(w)$ have the form $w^* := a\Sigma^{-1}(\mu_1 - \mu_0)$ for some $a \in \mathbb{R} \setminus \{0\}$.*

Proof. Note that $J(aw) = J(w)$ for all $a \in \mathbb{R} \setminus \{0\}$. So, we can choose w maximizing $w \mapsto w^\top (\mu_1 - \mu_0)(\mu_1 - \mu_0)^\top w$ subject to $w^\top \Sigma w = 1$. Using Lagrange multipliers, we arrive at the equation

$$(\mu_1 - \mu_0)(\mu_1 - \mu_0)^\top w = \lambda \Sigma w,$$

so we need to find eigenvectors of $\Sigma^{-1}(\mu_1 - \mu_0)(\mu_1 - \mu_0)^\top$. However, since $(\mu_1 - \mu_0)(\mu_1 - \mu_0)^\top v$ goes in the direction of $(\mu_1 - \mu_0)$ for all v, we know that $\Sigma^{-1}(\mu_1 - \mu_0)(\mu_1 - \mu_0)^\top v$ goes in the direction of $\Sigma^{-1}(\mu_1 - \mu_0)$ for all v. This means that all eigenvectors to non-negative eigenvalues are of the desired form. □

Remark 3.17 (Classifier for Fisher's Linear Discriminant Analysis). From Lemma 3.16, we can readily obtain the classifier from Remark 3.15. Choosing $w = \Sigma^{-1}(\mu_1 - \mu_0)$, we find that

$$\mathbf{E}[w^\top X|Y=1] - \mathbf{E}[w^\top X|Y=0] = \Sigma^{-1}||\mu_1 - \mu_0||^2 > 0,$$

so we find

$$\psi(x) = \begin{cases} 1, & (\mu^1 - \mu^0)^\top \Sigma^{-1} x > \kappa, \\ 0, & \text{otherwise.} \end{cases}$$

This is the same as in Example 3.10.3 if

$$\kappa = \tfrac{1}{2}(\mu_1 - \mu_0)^\top \Sigma^{-1}(\mu_1 + \mu_0) + \log \frac{\mathbf{P}(Y=0)}{\mathbf{P}(Y=1)}.$$

3.4 Naive Bayes classification

We will now introduce another general classification method. The *naive* means that we assume that all coordinates in the feature vector X are independent, and *Bayes* is related to the Bayes Theorem on conditional probabilities. Different from logistic regression, but similar to all other methods from Section 3, we assume a probability distribution on the features. Again assuming a set $\mathbb{S}$ of different classes, the idea is that – for a given item – the distribution of features $X \in \mathbb{R}^k$ depends on the class label $Y \in \mathbb{S}$, i.e. we start with a distribution (with $\vartheta \in \mathbb{R}^{\mathbb{S} \times k}$) of the form

$$\mathbf{P}_{\vartheta_{y\cdot}}(X \in dx|Y=y) = \prod_{i=1}^{k} \mathbf{P}_{\vartheta_{yi}}(X_i \in dx_i|Y=y), \tag{3.5}$$

depending on some $\vartheta = (\vartheta_{y\cdot})_{y\in\mathbb{S}} = (\vartheta_{yi})_{y\in\mathbb{S},i=1,\dots,k}$. This is similar to the Gaussian case from Example 3.10, but the right hand side assumes independence of features. There are two main options, which apply if X is categorical or continuous, respectively.

Definition 3.18 (Naive Bayes statistical model). *Let $\mathbb{E} \subseteq \mathbb{D}^k$ with $\mathbb{D}$ discrete, or $\mathbb{E} \subseteq \mathbb{R}^k$ open and $\mathbb{S}$ be finite and (X, Y) be an $\mathbb{E} \times \mathbb{S}$-valued random variable.*

1. *If $\mathbb{E} \subseteq \mathbb{D}^k$ with $\mathbb{D}$ discrete, let*

$$\mathbb{\Delta}_{\mathbb{D}} := \{\vartheta \in [0,1]^{\mathbb{D}} : \sum_d \vartheta_d = 1\}$$

be the set of probability distributions on $\mathbb{D}$ and $\Theta = (\mathbb{\Delta}_{\mathbb{D}})^{\mathbb{S}\times k}$. For $\vartheta = (\vartheta_{ydi})_{y\in\mathbb{S},d\in\mathbb{D},i=1,\dots,k}$, define the joint distribution by

$$\mathbf{P}_\vartheta(X = x, Y = y) = \mathbf{P}(Y = y) \prod_{i=1}^k \vartheta_{yx_i i},$$

i.e. given an item belongs to class y, then ϑ_{ydj} is the probability that the jth feature shows $d \in \mathbb{D}$, all items and features being independent.

2. *Let $\mathbb{E} \subseteq \mathbb{R}^k$ open and $\Theta = (\mathbb{R} \times \mathbb{R}_{>0})^{\mathbb{S}\times k}$. For $\vartheta = (\mu_{yi}, \sigma^2_{yi})_{y\in\mathbb{S},i=1,\dots,k}$, set*

$$\mathbf{P}_\vartheta(X \in dx, Y = y) = \mathbb{P}(Y = y) \prod_{i=1}^k \frac{1}{\sqrt{2\sigma^2_{yi}\pi}} \exp\Big(-\frac{(x_i - \mu_{yi})^2}{2\sigma^2_{yi}}\Big),$$

i.e. given an items belongs to class y, we have $X_i \sim N(\mu_{yj}, \sigma^2_{yj})$-distributed, all items and features being independent.

Actually, obtaining the Bayes classifier in the naive Bayes model is not hard.

Remark 3.19 (Bayes classifier for naive Bayes). 1. Consider the multivariate Bernoulli naive Bayes model. Then, the Bayes classifier is given by

$$\psi(x) = \mathrm{argmax}_{y\in\mathbb{S}}\delta_y(x), \qquad \delta_y(x) = \sum_{i=1}^k \log \vartheta_{yx_i i} + \log \mathbf{P}(Y = y).$$

2. Consider the Gaussian naive Bayes model. Then, the Bayes classifier in given by (recall also from Example 3.10)

$$\begin{aligned}\psi(x) &= \mathrm{argmax}_{y\in\mathbb{S}}\delta_y(x),\\ \delta_y(x) &= -\Big(\sum_{i=1}^k \Big(\frac{x_i - \mu_{yi}}{\sigma_{yi}}\Big)^2 + \log\sigma^2_{yi}\Big) + 2\log \mathbf{P}(Y = y).\end{aligned}$$

If we have some training data available, we will replace μ_{yi}, σ^2_{yi} in the Gaussian case by their maximum likelihood estimators. In the multinomial case, estimating ϑ_{ydi} by maximum likelihoods might lead to instabilities. The reason is that the maximum likelihood estimator of ϑ_{ydi} is zero if among all training data in class y feature i does not show d. As a result, if the test data has $x_i = d$, the chance it is classified into class y vanishes, not matter how well the other features match. Therefore, we do not maximize the likelihood, but some a posteriori probability. For this, we need the Dirichlet distribution.

Definition 3.20 (Dirichlet distribution). *Let $\mathbb{D}$ be finite and $\alpha \in \mathbb{R}^{\mathbb{D}}_{>0}$. A $\mathbb{\Delta}_{\mathbb{D}}$-valued random variable with distribution*

$$\mathbf{P}(\Theta \in d\vartheta) = 1_{\vartheta \in \mathbb{\Delta}_{\mathbb{D}}} \frac{\Gamma(\sum_{d'} \alpha_{d'})}{\prod_{d'} \Gamma(\alpha_{d'})} \prod_{d \in \mathbb{D}} \vartheta_d^{\alpha_d - 1}$$

is called Dirichlet distributed and we write $\Theta \sim Dir(\alpha)$.

Lemma 3.21. *1. Let $\Theta \sim Dir(\alpha)$. Then,*

$$\mathbf{E}[\Theta_d] = \frac{\alpha_d}{\sum_{d'} \alpha_{d'}}.$$

2. Let $X, X_1, X_2, ...$ be iid distributed on some finite $\mathbb{D}$ according to some $\vartheta \sim Dir(\alpha)$ with $\alpha \in \mathbb{R}^{\mathbb{D}}_{+}$. Then,

$$\mathbf{P}(X = d | X_1, ..., X_n) = \frac{|\{i : X_i = d\}| + \alpha_d}{n + \sum_{d'} \alpha_{d'}}$$

Proof. We write, using $\Gamma(x+1) = x\Gamma(x)$ for all $x > 0$

$$\begin{aligned}
\mathbf{E}[\Theta_d] &= \frac{\Gamma(\sum_{d'} \alpha_{d'})}{\prod_{d'} \Gamma(\alpha_{d'})} \int_{\mathbb{\Delta}_{\mathbb{D}}} \vartheta_d \prod_{d' \in \mathbb{D}} \vartheta_{d'}^{\alpha_{d'} - 1} d\vartheta \\
&= \frac{\Gamma(\sum_{d'} \alpha_{d'})}{\Gamma(1 + \sum_{d'} \alpha_{d'})} \frac{\prod_{d'} \Gamma(\delta_{dd'} + \alpha_{d'})}{\prod_{d'} \Gamma(\alpha_{d'})} \frac{\Gamma(1 + \sum_{d'} \alpha_{d'})}{\prod_{d'} \Gamma(\delta_{d+d'} + \alpha_{d'})} \int_{\mathbb{\Delta}_{\mathbb{D}}} \vartheta_d \prod_{d' \in \mathbb{D}} \vartheta_{d'}^{\alpha_{d'} - 1} d\vartheta \\
&= \frac{\alpha_d}{\sum_{d'} \alpha_{d'}}.
\end{aligned}$$

Given $X_1, ..., X_n$, we find that $\Theta \sim \mathrm{Dir}(\widetilde{\alpha})$ with $\widetilde{\alpha}_d = \alpha_d + |\{i : X_i = d\}|$. From this, we see that

$$\begin{aligned}
\mathbf{P}(X = d | X_1, ..., X_n) &= \mathbf{E}[\mathbf{P}(X = d | \Theta, X_1, ..., X_n) | X_1, ..., X_n] = \mathbf{E}[\Theta_d | X_1, ..., X_n] \\
&= \frac{|\{i : X_i = d\}| + \alpha_d}{n + \sum_{d'} \alpha_{d'}}
\end{aligned}$$

by 1. □

Definition 3.22 (Multivariate Bernoulli naive Bayes classifier with Dirichlet prior). *Let $\alpha \in \mathbb{R}^{\mathbb{D}}_{>0}$ and training data $(X_1, Y_1), ..., (X_n, Y_n)$ (with $X_i = (X_{ij})_{j=1,...,k}$) are available. Then, the classifier (with $x = (x_j)_{j=1,...,k} \in \mathbb{D}^k$)*

$$\psi(x) = argmax_{y \in \mathbb{S}} \widehat{\delta}_y(x), \qquad \widehat{\delta}_y(x) = \sum_{j=1}^{k} \log(\alpha_{x_j} + |\{i : Y_i = y, X_{ij} = x_j\}|)$$

is called the multivariate Bernoulli naive Bayes classifier with Dirichlet prior.

3.5 Nearest neighbor classifier

The examples we have treated so far were all *parametric*. This means that the underlying statistical model was *simple* is the sense that it only had finitely many dimensions as given by the model parameters. Now, we come to yet another classifier, which is not based on a certain distribution of the data. In other words, the underlying statistical model is non-parametric. The resulting class boundaries can have any form.

Definition 3.23 (Nearest neighbor classifier). *Assume we have training data $(X_i, Y_i)_{i=1,...,n}$, which are independent and identically distributed with values in $\mathbb{E} \times \mathbb{S}$ for some $\mathbb{E} \subseteq \mathbb{R}^k$ open, and that X_1 has some density on $\mathbb{E}$. Then, the classifier*

$$\psi^n_{near}((X_i, Y_i)_{i=1,...,n}, X) := Y_i \text{ iff } ||X_i - X|| = \min_j ||X_j - X||$$

is called the nearest neighbor classifier.

Recall

$$\eta(X) := \mathbf{P}(Y = 1|X).$$

We now provide a nice result on neirest neighbor classification in the binary case. However, the result shows that the nearest neighbor classifier is not consistent, since the Bayes error is (recall from Lemma 3.6) $e_{\psi^*} := \mathbf{E}[\eta(X) \wedge (1 - \eta(X))]$.

Theorem 3.24 (Cover-Hart-Theorem). *Let $\mathbb{S} = \{0, 1\}$, $\eta(X) := \mathbf{P}(Y = 1|X)$ and ψ^n_{near} be the nearest neighbor classifier. Then,*

$$e_{\psi_{near}} := \lim_{n\to\infty} \mathbf{E}[e^n_{\psi_{near}((X_i,Y_i),X)}] = \mathbf{E}[2\eta(X)(1 - \eta(X))].$$

In particular, for the Bayes error e_{ψ^*},

$$e_{\psi_{near}} \leq 2e_{\psi^*}.$$

Proof. First, since X has a density, we make the (trivial) observation

$$\mathbf{P}(p(X) > 0) = \int_{\mathbb{E}} 1_{p(x)>0} p(x) = \int_{\mathbb{E}} p(x) = 1.$$

In other words, the density is positive at X with probability 1. From this, we conclude that $p > 0$ in some neighborhood of X. From this, for any $\varepsilon > 0$

$$\mathbf{P}(||X_1 - X|| > \varepsilon|X) = \int_{\mathbb{E}} 1_{||x_1 - X||>\varepsilon} p(x_1)dx_1 < 1.$$

Therefore, using $(1) := \text{argmin}_i\{||X_i - X|| : i = 1, ..., n\}$,

$$\begin{aligned}\mathbf{P}(||X_{(1)} - X|| > \varepsilon) &= \mathbf{P}(\min_{j=1,...,n} ||X_j - X|| > \varepsilon) = \mathbf{E}[\mathbf{P}(\min_{j=1,...,n} ||X_j - X|| > \varepsilon|X)] \\ &= \mathbf{E}\Big[\mathbf{P}(||X_1 - X|| > \varepsilon|X)^n\Big] \xrightarrow{n\to\infty} 0.\end{aligned}$$

For the test pair (X, Y), let $J \in \{1, ..., n\}$ be such that $||X_J - X|| = \min_j ||X_j - X||$, such that $\psi(X) = Y_J$. We then write

$$\begin{aligned}\mathbf{P}(\psi(X) \neq Y | X, (X_1, Y_1), (X_2, Y_2), ...) &= \mathbf{P}(Y_J \neq Y | X, X_J) \\ &= \mathbf{P}(Y_J = 1, Y = 0 | X, X_J) + \mathbf{P}(Y_J = 0, Y = 1 | X, X_J) \\ &= \mathbf{P}(Y = 0|X)\mathbf{P}(Y_J = 1|X_J) + \mathbf{P}(Y = 1|X)\mathbf{P}(Y_J = 0|X_J) \\ &= (1 - \eta(X))\eta(X_J) + \eta(X)(1 - \eta(X_J)) \xrightarrow{n\to\infty} 2(1 - \eta(X))\eta(X).\end{aligned}$$

Now, the first assertion follows from

$$\mathbf{P}(\psi(X) \neq Y) = \mathbf{E}[\mathbf{P}(\psi(X) \neq Y)] \xrightarrow{n\to\infty} \mathbf{E}[2(1 - \eta(X))\eta(X)].$$

For the second assertion, note that $x(1 - x) \leq x \wedge (1 - x)$ for all $x \in [0, 1]$, hence

$$e_{\psi_{near}} = 2\mathbf{E}[\eta(X)(1 - \eta(X))] \leq 2\mathbf{E}[\eta(X) \wedge (1 - \eta(X))] = e_{\psi^*}.$$

□

As it will next turn out, we can improve the error of we change the classifier to use the k nearest neighbors.

Definition 3.25 (k-nearest neighbor classifier). *Let $k \in \mathbb{N}$ odd and consider the same situation as in Definition 3.23. For $x \in \mathbb{E}$ and $n \geq k$, let*

$$\psi^n_{k-near}((X_i, Y_i)_{i=1,...,n}, X) := y \text{ iff } |\{(i) : Y_{(i)} = y\}| > |\{(i) : Y_{(i)} = z\}| \text{ for all } z \neq y,$$

where $(1), (2), ...$ are the indices in $(X_i)_{i=1,...,n}$ closest, second closest,... to X.

Again, since X has a density, the indices $(1), (2), ...$ are well-defined. (If this is not the case, we would have to introduce an exception rule for a tie.) We now extend Theorem 3.24 for k-nearest neighbors.

Theorem 3.26 (Cover-Hart-Theorem for k-nearest neighbors). *For the situation as in Theorem 3.24*

$$\begin{aligned}e_{\psi_{k-near}} &:= \lim_{n\to\infty} \mathbf{E}[e_{\psi^n_{k-near}((X_i,Y_i),X)}] \\ &= \mathbf{E}\Big[\sum_{j=0}^{k} \binom{k}{j} \eta(X)^j (1 - \eta(X))^{k-j} (\eta(X) 1_{j<k/2} + (1 - \eta(X)) 1_{j>k/2})\Big].\end{aligned}$$

Proof. As in the proof of Theorem 3.24, we can show that $\mathbf{P}(||X_{(j)} - X|| > \varepsilon) \xrightarrow{n\to\infty} 0$ for all $j = 1, 2, ...$ Continuing as in the above proof, let $J_1, ..., J_k$ be the indices $(1), ..., (k)$ (which

are random since they depend on X). We now write (conditioning on the training data)

$$\begin{aligned}
\mathbf{P}(\psi^n_{k-\text{near}}(X) \neq Y|X) &= \mathbf{P}\Big(\sum_{l=1}^k Y_{J_l} < k/2, Y = 1|X\Big) + \mathbf{P}\Big(\sum_{l=1}^k Y_{J_l} > k/2, Y = 0|X\Big) \\
&= \sum_{j=0}^k \mathbf{P}\Big(\sum_{l=1}^k Y_{J_l} = j, Y = 1|X\Big) 1_{j<k/2} + \mathbf{P}\Big(\sum_{l=1}^k Y_{J_l} = j, Y = 0|X\Big) 1_{j>k/2} \\
&= \sum_{j=0}^k \mathbf{P}\Big(\sum_{l=1}^k Y_{J_l} = j|X, Y = 1\Big)\mathbf{P}(Y = 1|X) 1_{j<k/2} \\
&\qquad + \mathbf{P}\Big(\sum_{l=1}^k Y_{J_l} = j|X, Y = 0, X\Big)\mathbf{P}(Y = 0|X) 1_{j>k/2} \\
&\xrightarrow{n\to\infty} \sum_{j=0}^k \binom{k}{j} \eta(X)^j (1-\eta(X))^{k-j} \big(\eta(X) 1_{j<k/2} + (1-\eta(X)) 1_{j>k/2}\big)
\end{aligned}$$

Integrating with respect to X gives the result. □

Our next goal is to show consistency of k-nearest neighbor classifiers, at least for large k. Since we have already shown that convergence of the error for $n \to \infty$ to $e_{\psi_{k-\text{near}}}$, we will now show $e_{\psi_{k-\text{near}}} \xrightarrow{k\to\infty} e_{\psi^*}$. For this, recall

$$e_{\psi^*} = \mathbf{E}[\eta(X) \wedge (1 - \eta(X))]$$

from Lemma 3.6 and note the next lemma.

Lemma 3.27. *Let $p \leq 1/2$ and $X \sim B(k, p)$. Then*

$$\mathbf{P}(X > k/2) \leq \frac{2\sqrt{p(1-p)}}{\sqrt{k}(1-2p)}.$$

Proof. We use the Markov and Cauchy-Schwartz inequalities and write

$$\mathbf{P}(X > k/2) \leq \mathbf{P}(|X - pk| > k(\tfrac{1}{2} - p)) \leq \frac{\mathbf{E}[|X - pk|]}{k(\frac{1}{2} - p)} \leq \frac{\sqrt{\mathbf{V}[X]}}{k(\frac{1}{2} - p)} = \frac{2\sqrt{p(1-p)}}{\sqrt{k}(1-2p)}.$$

□

Theorem 3.28. *For the situation as in Theorem 3.26*

$$e_{\psi_{k-near}} - e_{\psi^*} \leq \sqrt{\frac{2e_{\psi_{near}}}{k}}.$$

In particular, $e_{\psi_{k-near}} - e_{\psi^*} \xrightarrow{k\to\infty} 0$.

Proof. From Theorem 3.26, we see the following: Conditionally on X and some $[0,1]$-valued function a, let $Z_{a(X)} \sim B(k, a(X))$. Then, we can write

$$\begin{aligned}
e_{\psi_{k-\text{near}}} &= \mathbf{E}\Big[\eta(X)\mathbf{P}(Z_{\eta(X)} < k/2|X) + (1-\eta(X))\mathbf{P}(Z_{\eta(X)} > k/2|X)\Big] \\
&= \mathbf{E}\Big[\eta(X)(1-\mathbf{P}(Z_{\eta(X)} > k/2|X)) + (1-\eta(X))\mathbf{P}(Z_{\eta(X)} > k/2|X), \eta(X) \le 1/2\Big] \\
&\quad + \mathbf{E}\Big[(1-(1-\eta(X)))\mathbf{P}(Z_{\eta(X)} < k/2|X)) + (1-\eta(X))(1-\mathbf{P}(Z_{\eta(X)} < k/2|X)), \eta(X) > 1/2\Big] \\
&= \mathbf{E}[\eta(X) + (1-2\eta(X))\mathbf{P}(Z_{\eta(X)} > k/2|X), \eta(X) \le 1/2] \\
&\quad + \mathbf{E}[1-\eta(X) + (1-2(1-\eta(X)))\mathbf{P}(Z_{1-\eta(X)} > k/2|X), \eta(X) > 1/2] \\
&= \mathbf{E}[\eta(X) \wedge (1-\eta(X)) + (1-2\eta(X) \wedge (1-\eta(X)))\mathbf{P}(Z_{\eta(X)\wedge(1-\eta(X))} > k/2|X)].
\end{aligned}$$

Together with Lemma 3.27, this gives

$$e_{\psi_{k-\text{near}}} \le \mathbf{E}[\eta(X) \wedge (1-\eta(X))] + \mathbf{E}\Big[\frac{2\sqrt{\eta(X)(1-\eta(X))}}{\sqrt{k}}\Big]$$

and the result follows. □

3.6 Examples

See the `Rmarkdown`-file on pages 71ff for an example of a naive Bayes classifier. See the `Rmarkdown`-file on pages 65ff for an example of nearest-neighbor classification.

4 Neural networks

> Some parts of the material presented in this section are adapted from [14] and [8]. The universal approximation theorem (for sigmoidal functions) was discovered in [4].

In recent years, neural networks have become a powerful tool in machine learning. In this section, we will try to explain what these networks are. Reasons why they are widely used today include the availability of cheap GPUs, which are processing units which can perform fast matrix algebra, and the growth of useful and big datasets where new methods could be applied.

4.1 Introduction

Consider logistic regression from Section 1.2. Here, we have come up with a function $f : x \mapsto \mathbf{P}(Y = 1) = \rho(x^\top \beta)$ with $\rho(t) = \frac{e^t}{1+e^t}$, where x is a vector of features and β is a vector of weights of these features; see Remark 1.11 and Figure 1.4. (Below, ρ will be called activation function and β will be called the vector of weights.) By this structure, for some new data x from a new item, we have classified the item into class 1 if $f(x) \ge 1/2$ and to class 0 otherwise. By this structure, there is a hyperplane with $x^\top \beta = 0$ which determines the border between class 0 and class 1. However, in real situations, i.e. if f is more complex, this border should allow for more complex structures. We have seen examples for this in the (non-parametric) nearest neighbor classifier in Section 4.1. Neural networks, as treated here, are still parametric

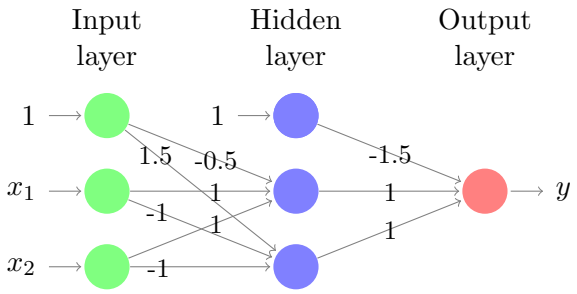


Figure 4.1: The XOR neural network with one hidden layer.

models (i.e. they come with a finite number of model parameters), but allow for arbitrary classification borders.

Here comes a famous example of a (relatively simple) function f, which cannot be learned by any linear approach. With $x = (x_1, x_2) \in \{0,1\}^2$, we consider the activation function $\rho(x) = 1_{x>0}$, and the XOR-function[12]. Identifying $0 \equiv$ `FALSE` and $1 \equiv$ `TRUE`, this function is given by[13]

$$f(x_1, x_2) := x_1 \veebar x_2 := (x_1 \vee x_2) \wedge (\neg(x_1 \wedge x_2)).$$

Before we come to this function, let us deal with other unary and binary operations. We write with $x^\top = (1, x_1, x_2)^\top$ (noting also the trivial identity $x_1 = \rho(x_1)$)

$$\begin{aligned}
\neg x_1 \equiv -x_1 + 1 = \rho(-x_1 + 0.5) &= \rho(x^\top \beta) \text{ with } \beta = (0.5, -1, 0),\\
x_1 \wedge x_2 \equiv \rho(x_1 + x_2 - 1.5) &= \rho(x^\top \beta) \text{ with } \beta = (-1.5, 1, 1),\\
x_1 \vee x_2 \equiv \rho(x_1 + x_2 - 0.5) &= \rho(x^\top \beta) \text{ with } \beta = (-0.5, 1, 1),\\
\neg(x_1 \wedge x_2) \equiv \rho(-x_1 - x_2 + 1.5) &= \rho(x^\top \beta) \text{ with } \beta = (1.5, -1, -1),\\
\neg x_1 \wedge x_2 \equiv \rho(-x_1 + x_2 - 0.5) &= \rho(x^\top \beta) \text{ with } \beta = (-0.5, -1, 1).
\end{aligned}$$

Moreover, it is clear that $x_1 \veebar x_2$ cannot be written as $\rho(x^\top \beta)$ for some appropriate β, since it would have to be $\beta = (\beta_0, 1, 1)$ for some β_0 due to the symmetric positive impact of x_1 and x_2. However, for such a β, we cannot choose β_0 such that both, $x_1 = x_2 = 0$ and $x_1 = x_2 = 1$ are covered.

However, it is in fact possible to write

$$\begin{aligned}
x_1 \veebar x_2 &= (x_1 \vee x_2) \wedge (\neg(x_1 \wedge x_2))\\
&\equiv \rho((1, x_1 \vee x_2, (\neg(x_1 \wedge x_2)))(-1.5, 1, 1))\\
&= \rho((1, \rho(x^\top \beta_1), \rho(x^\top \beta_2)), \beta_3)
\end{aligned}$$

with

$$\beta_1 = (-.5, 1, 1), \qquad \beta_2 = (1.5, -1, -1), \qquad \beta_3 = (-1.5, 1, 1),$$

i.e. as a composition of two layers of $x \mapsto \rho(x^\top \beta)$ for appropriate β. The latter, however, is more easily displayed in an illustration of a neural network as given in Figure 4.1.

[12] This stands for *exclusive or*.

[13] Recall the unary $\neg$ operator, as well as the binary $\wedge$ and $\vee$ operator. They could be defined as $\neq x \equiv 1 - x$, $x_1 \wedge x_2 = x_1 x_2$ and $x_1 \vee x_2 = 1 - (1 - x_1)(1 - x_2)$.

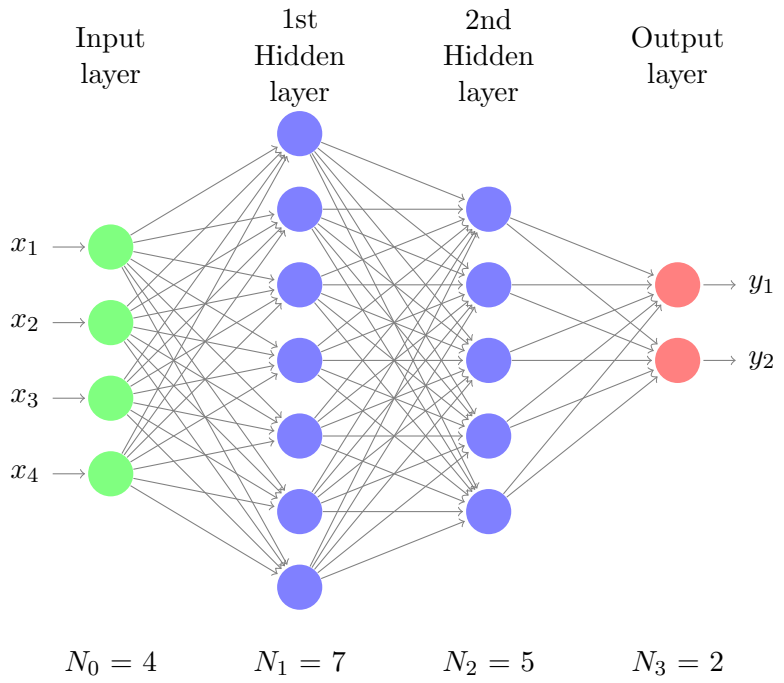


Figure 4.2: A general multilayer perceptron (MLP) with two hidden layers. Note, however, that neither the weights connecting nodes in different layers, nor the activation function, are displayed.

Generalizing this example to an arbitrary activation function (taking more than two values), an arbitrary number of features k, an arbitrary number of outputs l, an arbitrary number of hidden layers, we end up with the multilayer perceptron; see Figure 4.2 for an illustration.

Definition 4.1 (Multilayer perceptron). *Let* $m, k := j_0, j_1, ..., j_m, l := j_{m+1}, \in \mathbb{N}, \beta_i \in \mathbb{R}^{j_i \times j_{i-1}}, \gamma_i \in \mathbb{R}^{j_i}$, *and* $\tau_{\beta,\gamma}(x) = \beta x + \gamma, i = 1, ..., m+1$ *(i.e.* $\tau_{\beta_i,\gamma_i} : \mathbb{R}^{j_{i-1}} \to \mathbb{R}^{j_i}$*). Then, for* $\rho : \mathbb{R} \to \mathbb{R}$, *(writing*[14] $\rho : \mathbb{R}^k \to \mathbb{R}^k$ *given by* $\rho(x_1, ..., x_k) = (\rho(x_1), ..., \rho(x_k))$,*) the function*

$$g_{\beta,\gamma} : \begin{cases} \mathbb{R}^k & \to \mathbb{R}^l \\ x & \mapsto \tau_{\beta_{m+1},\gamma_{m+1}}(\rho(\tau_{\beta_m,\gamma_m}(\cdots \rho(\tau_{\beta_1,\gamma_1}(x)) \cdots))) \end{cases}$$

is called multilayer perceptron *with* m *hidden layers and activation function* ρ. *For* ρ, k, l, m *as above, we denote*

$$MLP(\rho, k, l, m) := \{g_{\beta,\gamma} = \tau_{\beta_{m+1},\gamma_{m+1}}(\rho(\tau_{\beta_m,\gamma_m}(\cdots \rho(\tau_{\beta_1,\gamma_1}(x)) \cdots))) : \tau_{\beta,\gamma}(x) := \beta x + \gamma \text{ for some } \beta_1, \gamma_1, ..., \beta_{m+1}, \gamma_{m+1}\}.$$

Remark 4.2 (Recursive scheme). In practise, we compute $g_{\beta,\gamma} \in \text{MLP}(\rho, k, l, m)$ recursively using the following scheme (in vector notation)

$$\begin{aligned} x_0 &:= x \in \mathbb{R}^{j_0}, \\ z_i &:= \beta_i x_{i-1} + \gamma_i \in \mathbb{R}^{j_i}, && i = 1, ..., l+1, \\ x_i &:= \rho(z_i) \in \mathbb{R}^{j_i}, && i = 1, ..., l, \end{aligned}$$

[14] This is an abuse of notation.

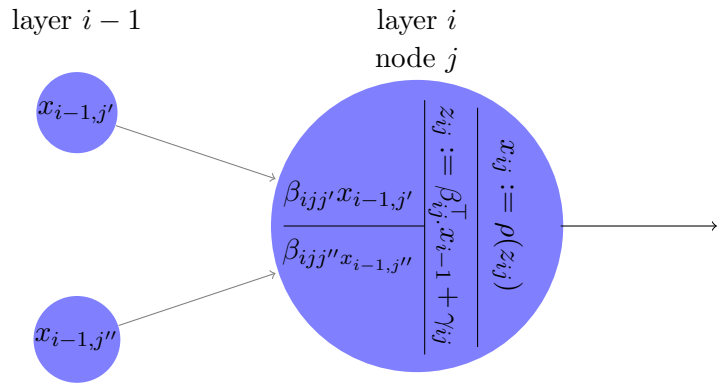


Figure 4.3: Some details of one node in the neural network. Input from layer $i-1$ is processed in layer i by weighting the inputs and applying the activation function ρ in order to produce the output for the next layer.

i.e. z_{ij} is the input into node j in layer $i = 1, ..., l+1$ as a weighted average of all outputs from the nodes in layer $i-1$, which is then evaluated using ρ in order to give the output x_{ij} of node j in layer i.
A more algorithmic way of writing this is

INPUT: Input $x, \rho, \beta_1, \gamma_1, ..., \beta_{m+1}, \gamma_{m+1}$ as above
OUTPUT: $g(x)$, value of the MLP at x

1: **for** $i = 1, ..., m$ **do**
2: $\quad z := \beta_i x + \gamma_i$
3: $\quad x := \rho(z)$
4: **end for**
5: **return** $\beta_{m+1}x + \gamma_{m+1}$

Remark 4.3 ($m = l = 1$). Later, we will use $m = l = 1$. We note that $g \in \mathrm{MLP}(\rho, k, 1, 1)$ has the form

$$g(x) = \beta'(\rho(\beta x + \gamma)) + \gamma'$$

with $\beta \in \mathbb{R}^{j_1 \times k}, \gamma \in \mathbb{R}^k$ (hence $\beta_1 x \in \mathbb{R}^{j_1}$) and $\beta' \in \mathbb{R}^{1 \times j_1}, \gamma' \in \mathbb{R}$. With $\beta' = e_i$ and $\gamma' = 0$, we find that $g(x) = \rho(\beta_i . x)$, hence

$$\{x \mapsto \rho(\beta x) : \beta \in \mathbb{R}^k\} \subseteq \mathrm{MLP}(\rho, k, 1, 1),$$

and $\mathrm{MLP}(\rho, k, 1, 1)$ is the linear space spanned by $\{x \mapsto \rho(\beta x) : \beta \in \mathbb{R}^k\} \cup \{x \mapsto \gamma : \gamma \in \mathbb{R}\}$.

Remark 4.4.

1. As in linear regression, we could omit the constant terms, and write βx rather than $\beta x + \gamma$. The reason is that we can extend x by one coordinate which equals 1 in all cases, and see the equality between these two expressions.

2. If ρ is linear, i.e. $\rho(x) = \alpha x$ for some $\alpha \in \mathbb{R}$, the multilayer perceptron (for any choice of $\beta_i, \gamma_i, i = 1, ..., m+1$ is linear.

3. More generally, if ρ is a polynomial of degree at most a, the multilayer perceptron f is a polynomial of degree at most am.

4.2 The universal approximation theorem: formulation

Starting with the observation in Section 4.1 that (i) only linear functions f can be learned without a hidden layer, and (ii) two layers suffice to learn all bi-variate boolean functions, we may ask which functions can be learned using one hidden layer for more general (not necessarily boolean) spaces, when we allow for an arbitrary number of nodes in the hidden layer. In this section, we are going to prove the surprising result that (provided the underlying space is compact) all continuous functions can be approximated by a MLP with one hidden layer, provided the activation function is discriminatory – see Definition 4.8. However, before we can even state the result – see Theorem 4.9 – we have to introduce some notions from topology, measure theory and functional analysis. (The latter is dealing withnextensions of linear algebra to linear spaces of functions).

Definition 4.5 (Some topology and measure theory). *1. For $x \in \mathbb{R}^k$, we define the open ball with radius r around x via*

$$B_r(x) := \{y \in \mathbb{R}^k : ||y - x|| < r\}.$$

2. Some $O \subseteq \mathbb{R}^k$ is open, *if $(x \in O \Rightarrow \exists \varepsilon > 0 : B_\varepsilon(x) \subseteq O)$.*
The system

$$\mathcal{O} := \{O \subseteq \mathbb{R}^k \text{ open}\}$$

is the euclidean topology on $\mathbb{R}^k$. For $\mathbb{K} \subseteq \mathbb{R}^k$, the set $\mathcal{O}_\mathbb{K} := \{O \cap \mathbb{K} : O \in \mathcal{O}\}$ is the trace topology on $\mathbb{K}$.

3. The Borel σ-algebra on $\mathbb{K} \subseteq \mathbb{R}^k$ is the smallest system of sets $\mathcal{F}_\mathbb{K}$ (of subsets of $\mathbb{K}$) such that $\mathcal{O}_\mathbb{K} \subseteq \mathcal{F}_\mathbb{K}$ and

- *$A \in \mathcal{F}_\mathbb{K} \Rightarrow K \setminus A \in \mathcal{F}_\mathbb{K}$;*
- *$A_1, A_2, ... \in \mathcal{F}_\mathbb{K} \Rightarrow A_1 \cup A_2 \cup \cdots \in \mathcal{F}_\mathbb{K}$.*

4. A finite signed measure on a Borel σ-algebra $\mathcal{F}$ is a map $\mu : \mathcal{F} \to \mathbb{R}$ such that

$$A_1, A_2, ... \in \mathcal{F} \text{ disjoint} \Rightarrow \mu\Big(\bigcup_{n=1}^{\infty} A_n\Big) = \sum_{n=1}^{\infty} \mu(A_n).$$

We denote the set of Borel measures by

$$\mathcal{M}_\mathbb{K} := \{\mu : \mathcal{F}_\mathbb{K} \to \mathbb{R} \text{ finite signed Borel measure}\}.$$

5. We call every function $f : \mathbb{K} \to \mathbb{R}$ of the form

$$f = \sum_{n=1}^{\infty} a_n 1_{A_n}$$

with $A_1, A_2, ... \in \mathcal{F}_\mathbb{K}$ disjoint and $a_1, a_2, ... \in \mathbb{R}$ simple.

6. Let $\mathbb{K} \subseteq \mathbb{R}^k$ be compact, $f \in \mathcal{C}(\mathbb{K})$ and μ a finite signed measure on $\mathcal{F}_\mathbb{K}$. Then, if

$$\inf\Big\{\sum_{n=1}^{\infty} a_n \mu(A_n) : \sum_{n=1}^{\infty} a_n 1_{A_n} \geq f \text{ simple}\Big\} = \sup\Big\{\sum_{n=1}^{\infty} a_n \mu(A_n) : \sum_{n=1}^{\infty} a_n 1_{A_n} \leq f \text{ simple}\Big\}$$

denote this by

$$\int f(x)\mu(dx).$$

Example 4.6. The *integral* which you heard about in the first year of your studies, is related to the above notion. There, you probably wrote $\int f(x)dx$ rather than $\int f(x)\mu(dx)$. Here, roughly speaking, μ weights the small strips in the integral by some factor $\mu(dx)$, which might be negative (since μ might be signed).

We will also need normed linear spaces.

Remark 4.7 (Continuous functions on a compact set). Let $\mathbb{K} \subseteq \mathbb{R}^k$ be compact[15] and $\mathbb{V} := \mathcal{C}(\mathbb{K})$. The sup-norm on $\mathbb{V}$ is given by

$$||f|| := \sup_{x \in \mathbb{K}} |f(x)|.$$

We call $\mathbb{W} \subseteq \mathbb{V}$ dense, if for all $f \in \mathbb{V}$ and $\varepsilon > 0$ there exists $g \in \mathbb{W}$ with $||g - f|| < \varepsilon$.

With the notions from above, we can now formulate the universal approximation theorem. For this, we need the notion of a discriminatory function. Below, we will see some examples.

Definition 4.8 (Discriminatory activation function). *Let $\mathbb{K} \subseteq \mathbb{R}^k$ be compact. Some $\rho \in \mathcal{C}(\mathbb{K})$ is called* $\mathbb{K}$-discriminatory *if, for any $\mu \in \mathcal{M}_{\mathbb{K}}$*

$$\Big(\int \rho(\beta x + \gamma) d\mu(x) = 0 \text{ for all } \beta \in \mathbb{R}^k, \gamma \in \mathbb{R} \Big) \Longrightarrow \mu = 0. \tag{4.1}$$

Theorem 4.9 (Universal approximation theorem). *Let $k \in \mathbb{N}$ and $K \subseteq \mathbb{R}^k$ compact, and $\rho : \mathbb{R} \to \mathbb{R}$ be $\mathbb{K}$-discriminatory. Then, MLP$(\rho, k, 1, 1)$ is dense in $\mathcal{C}(\mathbb{K})$.*

Apparently, the discriminitarory property of ρ is key here. We try to shed some light on this notion by making two examples. For these, we need some results in measure theory, which we state without proof:

Remark 4.10 (Some notes on finite signed measures). Let $\mathbb{K} \subseteq \mathbb{R}^k$ be compact.

1. *Absolute value for a signed measure:* For each $\mu \in \mathcal{M}_{\mathbb{K}}$, we find $\mathbb{K}^+, \mathbb{K}^-$ disjoint (and unique up to sets of μ-measure 0) and such that $\mu(A \cap \mathbb{K}^+) \geq 0$ and $\mu(A \cap \mathbb{K}^-) \leq 0$ for all $A \in \mathcal{B}(\mathbb{K})$. We can define uniquely a non-negative measure $|\mu|$ by setting

$$|\mu|(A) := \mu(A \cap \mathbb{K}^+) - \mu(A \cap \mathbb{K}^-).$$

2. *Uniqueness using projections:* If $\mu, \nu \in \mathcal{M}_{\mathbb{K}}$ are such that $\mu(\{x : \beta x > \alpha\}) = \nu(\{x : \beta x > \alpha\})$ for all $\alpha \in \mathbb{R}, \beta \in \mathbb{R}^k$, then $\mu = \nu$.

3. *Dominated convergence:* Let $\mu \in \mathcal{M}_{\mathbb{K}}$. If $f_1, f_2, ... \in \mathcal{C}(\mathbb{K})$ are such that $|f_n| \leq g$, $f_n(x) \xrightarrow{n \to \infty} f(x)$ for all $x \in \mathbb{K}$ and $\int g(x)|\mu|(dx) < \infty$, then

$$\int f_n(x)\mu(dx) \xrightarrow{n \to \infty} \int f(x)\mu(dx).$$

[15] This means that $\mathbb{K}$ is bounded and closed.

Remark 4.11 (Sigmoid functions are discriminatory). Any $\rho : \mathbb{R} \to \mathbb{R}$ with $\rho(x) \xrightarrow{x\to\infty} 1$ and $\rho(x) \xrightarrow{x\to-\infty} 0$ is called *sigmoid*. Such a function is $\mathbb{K}$-discriminatory for every compact $\mathbb{K} \subseteq \mathbb{R}^k$.
We write

$$\rho(n\beta x - n\gamma) + m) \xrightarrow{n\to\infty} 1_{\beta x>\gamma} + \rho(m)1_{\beta x=\gamma} \xrightarrow{m\to-\infty} 1_{\beta x>\gamma}.$$

Now, let $\mu \in \mathcal{M}_{\mathbb{K}}$ satisfy the left hand side of (4.1). Then, we find by dominated convergence that

$$\mu(\{x : \beta x > \gamma\}) = \lim_{m\to-\infty} \lim_{n\to\infty} \int \rho(n\beta x - n\gamma + m)\mu(dx) = 0$$

for all $\gamma \in \mathbb{R}, \beta \in \mathbb{R}^{1+k}$. This means that μ equals the 0-measure on all sets of the form $\{x : \beta x > \gamma\}$. By Remark 4.10.2, this means that $\mu = 0$.

Remark 4.12 (ReLu is discriminatory). The function $x \mapsto \rho(x) := x^+ := \max(x, 0)$ is called *Rectified Linear unit* and is discriminatory.
We proceed as above. Here, we note that, with $x_0 = 1$,

$$\begin{aligned}\rho(n\beta x - n\gamma) - \rho(n\beta x - n(\gamma + \tfrac{1}{n}) &= \max(n(\beta x - \alpha), 0) - \max(n(\beta x - \alpha) - 1, 0)\\ &\xrightarrow{n\to\infty} 1_{\beta x>\alpha}.\end{aligned}$$

From this, again using dominated convergence,

$$\mu(\{x : \beta x > \alpha\}) = \lim_{n\to\infty} \int \rho(n\beta x - n\gamma) - \rho(n\beta x - n(\gamma + \tfrac{1}{n}) = 0$$

for all $\gamma \in \mathbb{R}, \beta \in \mathbb{R}^k$. The rest follows as in Remark 4.11.

4.3 The universal approximation theorem: proof

Definition 4.13 (Normed linear space, dense subset). *A normed linear space $\mathbb{V}$ is a linear (vector) space, equipped with a norm $||.|| : \mathbb{V} \to \mathbb{R}_+$, i.e. a map satisfying*

$$||x|| = 0 \text{ iff } x = 0, \qquad ||ax|| = |a| \cdot ||x||, a \in \mathbb{R}, x \in \mathbb{V}, \qquad ||x + y|| \le ||x|| + ||y||, x, y \in \mathbb{V}.$$

1. *A subset $\mathbb{W} \subseteq \mathbb{V}$ is dense in $\mathbb{V}$ if for all $v \in \mathbb{V}$ and $\varepsilon > 0$ there is $w \in \mathbb{W}$ with $||v-w|| < \varepsilon$.*
2. *For some linear: $f : \mathbb{V} \to \mathbb{R}$, we define the norm of f by*

$$||f|| := \sup_{v\in\mathbb{V}} \frac{|f(v)|}{||v||}.$$

We say that f is bounded if $||f|| < \infty$.

The famous next result characterizes linear functions on $\mathcal{C}(K)$ (for compact $K \subseteq \mathbb{R}^k$). On the one hand – extending the results from your first year – the integral with repsect to any signed bounded measure – the map $f \mapsto \int f(x)\mu(dx)$ – is linear. On the other hand, Riesz representation theorem says that such maps are the only possible linear, bounded functions. In other words, any bounded, linear map is represented by a bounded signed measure. We state the result without proof.

Theorem 4.14 (Riesz representation theorem). *Let $K \subseteq \mathbb{R}^d$ be compact and $f^* : C(K) \to \mathbb{R}$ be linear and bounded. Then, there exists a finite, signed Borel measure μ on $\mathcal{F}_K$ such that*

$$f^*(f) = \int f(x)\mu(dx), \qquad f \in \mathcal{C}(K).$$

Theorem 4.15 (Hahn-Banach Theorem). *Let $\mathbb{V}$ be a linear, normed space and $\mathbb{W} \subseteq \mathbb{V}$ a linear subspace. For each linear and bounded $g : \mathbb{W} \to \mathbb{R}$, there exists $f : \mathbb{V} \to \mathbb{R}$ such that $f|_{\mathbb{W}} = g$ and $||f|| = ||g||$.*

Corollary 4.16. *Let $\mathbb{V}$ be a linear, normed space, $\mathbb{W} \subseteq \mathbb{V}$ a linear subspace and $v \in \mathbb{V}$. Then, for every $\varepsilon > 0$ there exist $w \in \mathbb{W}$ with $||w - v|| < \varepsilon$ if and only if there is no linear bounded $f : \mathbb{V} \to \mathbb{R}$ with $f(w) = 0$ for $w \in \mathbb{W}$ but $f(v) \neq 0$.*

Proof. $\Rightarrow$: Let $f : \mathbb{V} \to V$ linear and bounded with $f(w) = 0$ for $w \in \mathbb{W}$. Then, f is also continuous, and $f(v) = \lim_{n\to\infty} f(w_n) = 0$.
$\Leftarrow$: We proceed by contradiction and assume that there is $\varepsilon > 0$ such that $||w - v|| > \varepsilon$ for all $w \in \mathbb{W}$. We have to construct $f : \mathbb{V} \to \mathbb{R}$ linear and bounded with $f(w) = 0$ for $w \in \mathbb{W}$ and $f(v) \neq 0$. Let $\mathbb{W}'$ be the linear subspace generated by $\mathbb{W}$ and v. Define $g : \mathbb{W}' \to \mathbb{R}$ linear by setting $g(w + \lambda v) = \lambda$ for any $w \in \mathbb{W}$. Since, for $w \in \mathbb{W}$

$$\frac{|g(w + \lambda v)|}{||w + \lambda v||} = \frac{|\lambda|}{|\lambda| \cdot ||\lambda^{-1}w + v||} \leq \frac{1}{\varepsilon},$$

we see that g is bounded. Hence, by Theorem 4.15, there is $f : \mathbb{V} \to \mathbb{R}$ with $f|_{\mathbb{W}} = g|_{\mathbb{W}} = 0$ and $f(v) = g(v) = 1$. □

Proof of Theorem 4.9. Observe that $\mathrm{MLP}(\rho, k, 1, 1)$ is a linear subspace of $\mathcal{C}(\{1\} \times \mathbb{K})$. The proof is by contradiction, so assume there is $f \in \mathcal{C}(\{1\} \times \mathbb{K})$ and $\varepsilon > 0$ such that $||f - g|| > \varepsilon$ for all $g \in \mathrm{MLP}(\rho, k, 1, 1)$. By Corollary 4.16, there is $f^* : \mathcal{C}(\{1\} \times \mathbb{K}) \to \mathbb{R}$ with $f^*(g) = 0$ for $g \in \mathrm{MLP}(\rho, k, 1, 1)$ but $f^*(f) \neq 0$. Using the Riesz representation theorem, there is $\mu \in \mathcal{M}_{\{1\}\times\mathbb{K}}$ such that $f^*(f) = \int f(x)\mu(dx)$ for all $f \in \mathcal{C}(\{1\} \times \mathbb{K})$. However, using Remark 4.3, we find that for $g \in \mathrm{MLP}(\rho, k, 1, 1)$ with $g(x) = \rho(\beta x)$, where $\beta \in \mathbb{R}^k$ that

$$\int g(x)\mu(dx) = \int \rho(\beta x)\mu(dx) = 0.$$

Since ρ is discriminatory, this implies that $\mu = 0$, which contradicts that $0 \neq f^*(f) = \int f(x)\mu(dx)$. Hence we are done. □

4.4 Backpropagation

The universal approximation theorem is an important theoretical result. However, in practise, for $\ell \in \mathcal{C}(\mathbb{K})$, we still have to compute a function $\ell_{\beta,\gamma} \in \mathrm{MLP}(\rho, k, l, m)$ approximating ℓ. Usually, this is done using (stochastic) gradient descent and a loss function (which needs to be minimized) is $(\beta, \gamma) \mapsto \ell(y; g_{\beta,\gamma}(x)) = \frac{1}{n}\sum_{i=1}^n \ell_i(y_i, g_{\beta,\gamma}(x_i))$, where ℓ_i measures the loss on training data (x_i, y_i). Here, we will study the task of computing the gradient, i.e. $\nabla_{\beta,\gamma}\ell(y; g_{\beta,\gamma}(x))$. The backpropagation works backwards from layer $l + 1$, as the name suggests.

Recall (vectors are column vectors, and their transpose is a row vector)

$$\begin{aligned}(\ell \circ g_{\beta,\gamma})(x) &= \ell(z_{m+1}),\\ x_0 &:= x \in \mathbb{R}^{j_0},\\ z_i &:= \beta_i x_{i-1} + \gamma_i \in \mathbb{R}^{j_i}, && i = 1, ..., m+1,\\ x_i &:= \rho(z_i) \in \mathbb{R}^{j_i}, && i = 1, ..., m,\end{aligned}$$

i.e. z_{ij} is the input into node j in layer $i = 1, ..., m+1$ as a weighted average of all outputs from the nodes in layer $i-1$, which is then evaluated using ρ in order to give the output x_{ij} of node j in layer i.

Theorem 4.17 (Backpropagation). *In the setting above, (i.e. $\beta_1, ..., \beta_{m+1}, \gamma_1, ..., \gamma_{m+1}$ are the parameters of layer $1, ..., m+1$ with $\beta_i \in \mathbb{R}^{j_i \times j_{i-1}}$ and $\gamma_i \in \mathbb{R}^{j_i}$), and we have a function $\ell : \mathbb{R}^{j_{m+1}} \to \mathbb{R}$. We use the iterative scheme from Remark 4.2. The partial derivatives*

$$\begin{aligned}\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \beta_i} &:= \Big(\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \beta_{ijj'}}\Big)_{j=1,...,j_i, j'=1,...,j_{i-1}} \in \mathbb{R}^{j_i \times j_{i-1}},\\ \frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \gamma_i} &:= \Big(\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \gamma_{ij}}\Big)_{j=1,...,j_i} \in \mathbb{R}^{1 \times j_i}\end{aligned}$$

can be computed as follows: First, define iteratively[16]*, viewing ℓ as a function of z_{m+1} in the first line,*

$$\begin{aligned}\delta_{m+1} &= \nabla \ell(z_{m+1}) \in \mathbb{R}^{1 \times j_{m+1}},\\ \delta_i &= \delta_{i+1} \beta_{i+1} diag(\rho'(z_i)) \in \mathbb{R}^{1 \times j_i}, \qquad i = m, ..., 1.\end{aligned} \tag{4.2}$$

Then,

$$\begin{aligned}\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \beta_i} &= (\delta_{ij} x_{i-1,j'})_{j=1,...,j_i, j'=1,...,j_{i-1}} = \delta_i x_{i-1}^\top,\\ \frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \gamma_i} &= (\delta_{ij})_{j=1,...,j_i} = \delta_i.\end{aligned}$$

Proof. We set

$$\delta_i := \nabla_{z_i}(\ell \circ g_{\beta,\gamma}) := \Big(\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial z_{ij}}\Big)_{j=1,...,j_i} \in \mathbb{R}^{1 \times j_i}, \qquad i = 1, ..., m+1,$$

i.e. δ_i is the gradient of $\ell \circ g_{\beta,\gamma}$, viewed as a function of the weighted inputs $z_i = (z_{ij})_{j=1,...,j_i}$ at layer i (and evaluated at z_i). We will show that this definition coincides with (4.2). Note that

$$z_i = \beta_i \rho(z_{i-1}) + \gamma_i = \Big(\sum_{j'=1}^{j_{i-1}} \beta_{ijj'} \rho(z_{i-1,j'}) + \gamma_{ij}\Big)_{j=1,...,j_i} \in \mathbb{R}^{j_i},$$

and we can view z_i as a function of z_{i-1} using this equation. Hence,

$$\begin{aligned}\frac{\partial z_i}{\partial z_{i-1}} &= \Big(\frac{\partial z_{ij}}{\partial z_{i-1,j'}}\Big)_{j=1,...,j_i, j'=1,...,j_{i-1}} = \Big(\beta_{ijj'} \rho'(z_{i-1,j'})\Big)_{j=1,...,j_i, j'=1,...,j_{i-1}}\\ &= \beta_i \mathrm{diag}(\rho'(z_{i-1})) \in \mathbb{R}^{j_i \times j_{i-1}}.\end{aligned}$$

[16] We write ∇f as a row vector, generalizing the usual Jacobi matrix.

For (4.2), we write, noting that $g_{\beta,\gamma}(x) = z_{m+1}$

$$\delta_{m+1} := \nabla \ell(z_{m+1}) \in \mathbb{R}^{1 \times i_{m+1}}.$$

Then, using the chain rule from multi-dimensional calculus,

$$\delta_i := \frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial z_{i+1}} \frac{\partial z_{i+1}}{\partial z_i} = \delta_{i+1} \beta_{i+1} \mathrm{diag}(\rho'(z_i)) \in \mathbb{R}^{1 \times j_i}.$$

Finally, we use

$$\begin{aligned} \frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \beta_i} &:= \Big(\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \beta_{ijj'}} \Big)_{j=1,\dots,j_i, j'=1,\dots,j_{i-1}} = \Big(\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial \beta_{ijj'}} \Big)_{j=1,\dots,j_i, j'=1,\dots,j_{i-1}} \\ &= \Big(\delta_{ij} x_{i-1,j'} \Big)_{j=1,\dots,j_i, j'=1,\dots,j_{i-1}} = \delta_i x_{i-1}^\top \in \mathbb{R}^{j_i \times j_{i-1}} \end{aligned}$$

and

$$\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \gamma_i} := \Big(\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial \gamma_{ij}} \Big)_{j=1,\dots,j_i} = \Big(\frac{\partial(\ell \circ g_{\beta,\gamma})}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial \gamma_{ij}} \Big)_{j=1,\dots,j_i} = \delta_i.$$

So, the result follows. □

Remark 4.18. Sometimes, the output layer also uses ρ, i.e. $x_{m+1} = \rho(z_{m+1})$ is reported on the output layer (rather than z_{m+1}. For example, this is the case for classification where $j_{m+1} = |\mathbb{C}|$, i.e. nodes $1, ..., j_{m+1}$ are the classes, and the goal for the output nodes are $g_{\beta,\gamma}(x_i) = e_{y_i}$, where $g_{\beta,\gamma} \in \mathrm{MLP}(\rho, k, l, m)$. In this case, we can write $\widetilde{\ell} = \ell \circ \rho$ and find

$$\nabla \widetilde{\ell}(z_{m+1}) = \nabla \ell(\rho(z_{m+1})) \mathrm{diag}(\rho'(z_{m+1})).$$

So, in this case, we replace the first equality in (4.2) by the right hand side.

4.5 Extensions

We have only covered feed-forward networks with a finite number of hidden layers. As for our theoretical results, e.g. Theorem 4.9, we restricted ourselves to one hidden layer (although it is an easy corollary that the Universal Approximation Theorem continues to hold for more than one hidden layer). Let us briefly mention some other use cases for neural networks. In all cases, the basic structure of using an input layer, an output layer and nodes inbetween, is the same. *Convolutional Neural Networks* are designed to work with image data. Rather than connecting all nodes from layer $i-1$ to all nodes in layer i, nodes in the first hidden layer are only connexted to a small subset of input pixels in order to be able to recognize patterns in this area of the image. In recurrent neural networks, there is no clear order of layers since input from one layer can be the input of any other layer. Such network designs are often said to have memory since the input signal might stay for many computations within the network. Such networks are e.g. used in speech recognition or learning to read handwritten letters. Similar techniques also work when an image is made from a verbal description. Yet another network design are Generative Adversarial Networks, where two neural networks are trained by each other. For example, one network is trained to produce images from a verbal description, while the other network is trained to distinguish real images from images of the other network.

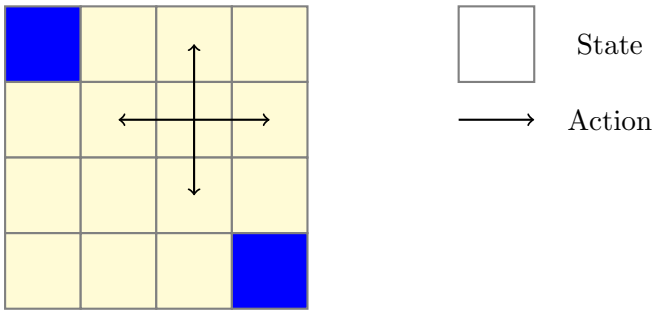


Figure 5.1: A robot moves on a grid. Each square in the grid is a state, and each direction it moves is an action.

4.6 Example

See the `Rmarkdown`-file on pages 78ff.

5 Reinforcement Learning

A standard reference in the field of reinforcement learning is [16]. A (somewhat slow but precise) introduction is [18]. We also used [9] for the preparation.

5.1 Markov Decision Processes

Example 5.1 (Moving robot 1). Consider a robot which has to move on the grid from Figure 5.1 from its current position to one of the two blue squares as fast as possible. This optimization task is easy to solve, but we can make it harder by saying that some actions (i.e. going north, east, south or west) are more expensive than others. So, let us introduce something more general, which are Markov Decision Processes.

Definition 5.2 (Markov Decision Process). *Let $\gamma \in (0,1]$ be a* discount factor*, $\mathbb{S}, \mathbb{A}$ be finite sets, called the set of states and the set of actions. A Markov Decision Process is a time-homogeneous Markov chain $(X_t)_{t=0,1,2,\dots}$ with $X_t = (S_t, A_t)$ and state space $\mathbb{S} \times \mathbb{A}$. It has the following structure: The distribution of S_{t+1} only depends on $X_t = (S_t, A_t)$ and we denote the transition matrix by*

$$P^a_{ss'} := \mathbf{P}(S_{t+1} = s' | S_t = s, A_t = a).$$

The distribution of A_t (the action) only depends on S_t (the state) and we denote by

$$\pi_s(a) = \mathbf{P}(A_t = a | S_t = s).$$

a (decision/action) policy *for the Markov Decision Process. The* reward *is a function $r : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$. In a Markov decision process, we call*

$$G_t = r(X_t) + \gamma r(X_{t+1}) + \dots = \sum_{k=0}^{\infty} \gamma^k r(X_{t+k}).$$

The state value function $v_\pi : \mathbb{S} \to \mathbb{R}$ *for policy* π *is given by*

$$v_\pi(s) := \mathbf{E}[G_0|S_0 = s].$$

The action value function $g_\pi : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ *is given by*

$$g_\pi(s, a) := \mathbf{E}[G_0|S_0 = s, A_0 = a].$$

Remark 5.3. Due to time-homogeneity, the state and action value function can also be defined by exchanging 0 by any $t = 0, 1, 2, ...$ on the right hand sides.

By construction,

$$v_\pi(s) = \mathbf{E}[\mathbf{E}[G_0|S_0 = s, A_0]|S_0 = s] = \mathbf{E}[g_\pi(s, A_0)].$$

We start with a fundamental lemma, which uses a restart argument of the Markov decision process.

Lemma 5.4 (Bellmann equations). *Let* $S = s \in \mathbb{S}$*,* $A \sim \pi_s$*, and* $S' \sim P^A_{s\cdot}$ *given* A*, and* $A' \sim \pi_{S'}$ *given* S'*. Then,* $(S, A, S', A') \sim (S_0, A_0, S_1, A_1)$ *and*

$$v_\pi(s) = \mathbf{E}[r(s, A)] + \gamma \mathbf{E}[v_\pi(S')|S = s], \tag{5.1}$$

$$g_\pi(s, a) = r(s, a) + \gamma \mathbf{E}[v_\pi(S')|S = s, A = a]. \tag{5.2}$$

Proof. Note that $(S, A, S', A') \sim (S_0, A_0, S_1, A_1)$ by construction. We write for the state value function, recalling that $X_k = (S_k, A_k)$,

$$\begin{aligned} v_\pi(s) &= \mathbf{E}\Big[\sum_{k=0}^{\infty} \gamma^k r(X_k)|S_0 = s\Big] = \mathbf{E}\Big[r(X_0) + \gamma\Big(\sum_{k=0}^{\infty} \gamma^k r(X_{k+1})\Big)|S_0 = s\Big] \\ &= \mathbf{E}[r(s, A)] + \gamma \mathbf{E}\Big[\mathbf{E}\Big[\sum_{k=0}^{\infty} \gamma^k r(X_{k+1})|S_1 = S', S_0 = s\Big]\Big] \\ &= \mathbf{E}[r(s, A)] + \gamma \mathbf{E}\Big[\mathbf{E}\Big[\sum_{k=0}^{\infty} \gamma^k r(X_k)|S_0 = S'\Big]\Big] \\ &= \mathbf{E}[r(s, A)] + \gamma \mathbf{E}[v_\pi(S')]. \end{aligned}$$

Similarly, for the action value function,

$$\begin{aligned} g_\pi(s, a) &= \mathbf{E}\Big[\sum_{k=0}^{\infty} \gamma^k r(X_k)|S_0 = s, A_0 = a\Big] \\ &= r(s, a) + \gamma \mathbf{E}\Big[\mathbf{E}\Big[\sum_{k=0}^{\infty} \gamma^k r(X_{k+1})|S_1 = S', S_0 = s, A_0 = a\Big]\Big] \\ &= r(s, a) + \gamma \mathbf{E}[v_\pi(S')|S = s, A = a]. \end{aligned}$$

□

Remark 5.5. In fact, it is possible to compute the state values using the last lemma using the following matrix notation: We can write

$$\mathbf{E}[v_\pi(S')] = \sum_{s',a} v_\pi(s')\mathbf{P}(S' = s', A = a|S = s) = \sum_{s',a} v_\pi(s')\pi_s(a)P^a_{ss'},$$

therefore (5.1) turns into

$$v_\pi(s) = \sum_a \pi_s(a)\Big(r(s,a) + \gamma \sum_{s'} P^a_{ss'} v_\pi(s')\Big). \tag{5.3}$$

This equation shows that Lemma 5.4 gives a system of linear equations for v_π. Moreover, once v_π is known, the right hand side of (5.2) gives g_π by using

$$g_\pi(s,a) = r(s,a) + \gamma \mathbf{E}[v_\pi(S')|S = s, A = a] = r(s,a) + \gamma \sum_{s'} v_\pi(s') P^a_{ss'}.$$

In other words, v_π can be computed by solving a linear system. However, systems with large state space $\mathbb{S}$ require the inversion of a large matrix, which is computationally costly.

5.2 The fixed point theorem

For a more efficient algorithm for computing v_π, note the following: Given a policy (of actions) π (i.e. $\pi_s(a)$ is the action in state s), we can use (5.3) in order to see that v_π is a fixed point of

$$f_\pi : \begin{cases} \mathbb{R}^{\mathbb{S}} & \to \mathbb{R}^{\mathbb{S}} \\ v & \mapsto \Big(\sum_a \pi_s(a)\Big(r(s,a) + \gamma \sum_{s'} P^a_{ss'} v(s')\Big)\Big)_{s \in \mathbb{S}}. \end{cases} \tag{5.4}$$

(Here, a fixed point is any $v \in \mathbb{R}^{\mathbb{S}}$ with $f_\pi(v) = v$.) Such a formulation can be used in order to show existence and uniqueness of solutions, provided f is a contraction.

Definition 5.6 (Contraction). *Let $d = 1, 2, ...$ and $||.||$ be some norm on $\mathbb{R}^d$ and $\mathbb{E} \subseteq \mathbb{R}^d$. Let $\gamma \in [0, 1)$. A function $f : \mathbb{E} \to \mathbb{E}$ is a* γ-contraction *if*

$$||f(v) - f(w)|| \le \gamma ||v - w||, \qquad v, w \in \mathbb{E}.$$

Example 5.7. Let f be as in (5.4) and $||v|| := \sup_s |v_s|$ be the sup-norm in $\mathbb{R}^d$. Then, for $v, w \in \mathbb{R}^{\mathbb{S}}$,

$$||f_\pi(v) - f_\pi(w)|| = \gamma \sup_s \Big| \sum_{a,s'} \pi_s(a) P^a_{ss'}(v_{s'} - w_{s'}) \Big| \le \gamma \sup_s \sum_{a,s'} \pi_s(a) P^a_{ss'} ||v - w|| = \gamma ||v - w||,$$

where we have used that π_s and $P^a_{s\cdot}$ are probability distributions. Hence, f_π is a γ-contraction.

We state without proof the result that every Cauchy sequence (in $\mathbb{R}^d$) converges. (The reverse is true as well.) The reason we do not prove this statement is that this fact helps to define the real numbers, and is tightly connected to the construction of the real numbers.

Theorem 5.8 ($\mathbb{R}^d$ is complete). *Let $x_1, x_2, ... \in \mathbb{R}^d$ be a Cauchy sequence, that is, for all $\varepsilon > 0$ there is some finite N such that $||x_n - x_m|| < \varepsilon$ as long as $m, n > N$. Then, there is $x \in \mathbb{R}^d$ such that $x_n \xrightarrow{n\to\infty} x$.*

Theorem 5.9 (Fixed point theorem). *Let $\gamma \in [0, 1)$, $\mathbb{E} \subseteq \mathbb{R}^d$ be closed and $f : \mathbb{E} \to \mathbb{E}$ a γ-contraction. Then, there exists a unique fixed point of f, i.e. there is a unique $x^* \in \mathbb{R}^d$ with $f(x^*) = x^*$. Moreover, for all $x_0 \in \mathbb{E}$, define $x_{n+1} := f(x_n)$, $n = 0, 1, 2, ...$ Then,*

$$||x_n - x^*|| \le \frac{||x_1 - x_0||}{1 - \gamma} \gamma^n \xrightarrow{n\to\infty} 0,$$

i.e. convergence to x^* is exponentially fast with rate γ.

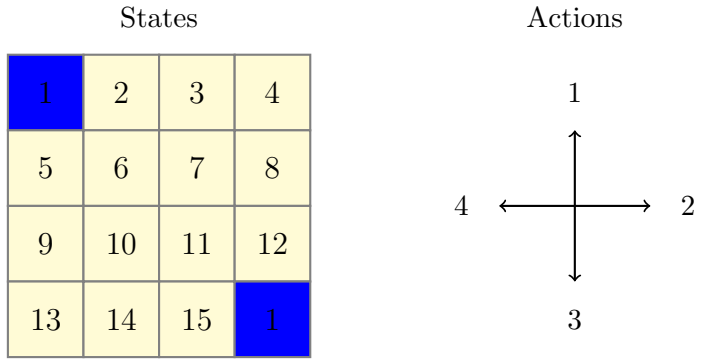


Figure 5.2: This is how we encode states and actions in our code.

Proof. First, note that f must be continuous by the contraction property. We start with proving uniqueness of the fixed point. Let $x^*, y^* \in \mathbb{E}$ be two fixed points. Then,

$$||x^* - y^*|| = ||f(x^*) - f(y^*)|| \leq \gamma ||x^* - y^*||,$$

which can only be true if $||x^* - y^*|| = 0$ or $x^* = y^*$. For existence, let $x_0, x_1, ... \in \mathbb{E}$ be as in the statement of the Theorem. Then, by induction

$$||x_{n+1} - x_n|| = ||f(x_n) - f(x_{n-1})|| \leq \gamma ||x_n - x_{n-1}|| \leq \cdots \leq \gamma^n ||x_1 - x_0||$$

for all $n = 0, 1, ...$ We now show that $x_0, x_1, ...$ is Cauchy. So, let $\varepsilon > 0$. Then, let N be large enough for $\gamma^N < (1-\gamma)\varepsilon / ||x_1 - x_0||$. Now, for $n \geq m > N$

$$||x_n - x_m|| \leq \sum_{k=m}^{n-1} ||x_{k+1} - x_k|| \leq ||x_1 - x_0|| \sum_{k=m}^{\infty} \gamma^k = ||x_1 - x_0|| \frac{\gamma^m}{1-\gamma} \leq ||x_1 - x_0|| \frac{\gamma^N}{1-\gamma} < \varepsilon. \tag{5.5}$$

By Theorem 5.8, there is $x^* \in \mathbb{E}$ with $x_n \xrightarrow{n \to \infty} x^*$. This must actually be a fixed point of f since (by continuity of f)

$$f(x^*) = \lim_{n \to \infty} f(x_n) = \lim_{n \to \infty} x_{n+1} = x^*.$$

Finally, the last statement follows since $||x_m - x^*|| = \lim_{n \to \infty} ||x_m - x_n||$ and the same calculation as in (5.5). □

Corollary 5.10 (Policy evaluation). *Consider f_π from* (5.4)*. Then, v_π is the unique fixed point of f_π. Moreover, for arbitrary $v_0 = v$, define the recursion $v_{k+1} = f_\pi(v_k)$. Then, $v_k \xrightarrow{k \to \infty} v_\pi$ and convergence is exponentially fast with rate at most γ.*

Proof. The result follows directly from Theorem 5.9 and Example 5.7. □

Example 5.11 (Moving robot 2). We continue with Example 5.1 from Figure 5.1 of the moving robot. First, we need to formalise the system, which is done in Figure 5.2. We have 15 states and 4 actions. The accompanying `R-markdown`-file from pages 83.

5.3 Optimal policies

Our goal is to find an optimal policy π^*, i.e. a policy which maximizes the state value function from Definition 5.2. We now see that it is in fact possible to optimize the policy simultaneously for all states. We start with a definition.

Definition 5.12. *On the set $\mathbb{A}_{\mathbb{A}}^{\mathbb{S}}$ of policies, we define the partial order*[17]

$$\pi \leq \pi' \qquad \Longleftrightarrow \qquad v_\pi(s) \leq v_{\pi'}(s) \text{ for all } s \in \mathbb{S}.$$

A policy $\pi^* \in \mathbb{A}_{\mathbb{A}}^{\mathbb{S}}$ is optimal if $\pi \leq \pi^*$ for all $\pi \in \mathbb{A}_{\mathbb{A}}^{\mathbb{S}}$.

Theorem 5.13 (Bellmann Optimality Equation). *1. There is a unique solution $v^* \in \mathbb{R}^{\mathbb{S}}$ for*

$$v^*(s) = \max_{\pi_s \in \mathbb{A}_{\mathbb{A}}} \Big(\sum_a \pi_s(a) \Big(r(s,a) + \gamma \sum_{s'} P_{ss'}^a v^*(s') \Big) \Big), \qquad s \in \mathbb{S}.$$

2. It is the unique fixed point of

$$f : \begin{cases} \mathbb{R}^{\mathbb{S}} & \to \mathbb{R}^{\mathbb{S}} \\ v & \mapsto \Big(\max\limits_{\pi_s \in \mathbb{A}_{\mathbb{A}}} \Big(\sum\limits_a \pi_s(a) \Big(r(s,a) + \gamma \sum\limits_{s'} P_{ss'}^a v(s') \Big) \Big) \Big)_{s \in \mathbb{S}}. \end{cases} \tag{5.6}$$

Here, f is a γ-contraction.

3. Any policy $\pi^* = (\pi_s^*)_{s \in \mathbb{S}}$ with

$$\pi_s^* := argmax_{\pi_s \in \mathbb{A}_{\mathbb{A}}} \Big(\sum_a \pi_s(a) \Big(r(s,a) + \gamma \sum_{s'} P_{ss'}^a v^*(s') \Big) \Big) \in \mathbb{A}_{\mathbb{A}}, \qquad s \in \mathbb{S}.$$

(where argmax can be defined arbitrarily in case of multiple maxima) is optimal and $v_{\pi^*} = v^*$.

Proof. For 1. and 2. it is clear that v^* is a solution in 1. iff $f(v^*) = v^*$ for f as in 2. Hence, it suffices to show that f is a γ-contraction. Then, Theorem 5.9 shows both assertions. Let $v, w \in \mathbb{R}^{\mathbb{S}}$ and $\pi^v, \pi^w \in \mathbb{A}_{\mathbb{A}}^{\mathbb{S}}$ be the maximizers in the definition of f at v and w, respectively. Then,

$$(f(v) - f(w))_s = \sum_a \pi_s^v(a) \Big(r(s,a) + \gamma \sum_{s'} P_{ss'}^a v(s') \Big) - \pi_s^w(a) \Big(r(s,a) + \gamma \sum_{s'} P_{ss'}^a w(s') \Big)$$

and we can write, using π_s^v instead of π_s^w in the second term

$$\begin{aligned} &\leq \gamma \sum_a \pi_s^v(a) \Big(\sum_{s'} P_{ss'}^a (v(s') - w(s')) \Big) \\ &\leq \gamma \sum_a \pi_s^v(a) \Big(\sum_{s'} P_{ss'}^a ||v - w|| \Big) = \gamma ||v - w||, \end{aligned}$$

[17] For a set $\mathbb{B}$, a partial order is some $R \subseteq \mathbb{B} \times \mathbb{B}$ such that (writing $b \leq c$ for $(b,c) \in R$) (i) $b \leq b$ for all $b \in \mathbb{B}$; (ii) if $a \leq b$ and $b \leq a$, then $a = b$ and (iii) if $a \leq b$ and $b \leq c$, then $a \leq c$. A partial order is total if, for all $a, b \in \mathbb{B}$, either $a \leq b$ or $b \leq a$.

but also, using π_s^w instead of π_s^v in the first term

$$\begin{aligned}
&\geq \gamma \sum_a \pi_s^w(a)\Big(P_{ss'}^a(v(s') - w(s'))\Big) \\
&\geq -\gamma \sum_a \pi_s^w(a)\Big(P_{ss'}^a||v - w||\Big) = -\gamma||v - w||.
\end{aligned}$$

By these calculations, we find that f is a γ-contraction. For 3., we start by showing that any value function is smaller than v^*, the unique fixed point of f. So, let $\pi \in \mathbb{\Delta}_{\mathbb{A}}^{\mathbb{S}}$. Then,

$$\begin{aligned}
v^*(s) - v_\pi(s) &= \max_{\tilde{\pi}_s} \Big(\sum_a \tilde{\pi}_s(a)\Big(r(s,a) + \gamma \sum_{s'} P_{ss'}^a v^*(s')\Big)\Big) \\
&\qquad\qquad - \sum_a \pi_s(a)\Big(r(s,a) + \gamma \sum_{s'} P_{ss'}^a v_\pi(s')\Big) \\
&\geq \gamma \sum_a \pi_s(a) \sum_{s'} P_{ss'}^a (v^*(s') - v_\pi(s')) \\
&\geq \gamma^2 \sum_{a,a',s',s''} \pi_s(a)\pi_{s'}(a') P_{ss'}^a P_{s's''}^{a'} (v^*(s'') - v_\pi(s'')) \\
&\geq \cdots \geq \gamma^n \sum_{a_0,\ldots,a_{n-1},s_1,\ldots,s_n} \pi_s(a_0)\cdots\pi_{s_{n-1}}(a_{n-1}) P_{ss_1}^{a_0} \cdots P_{s_{n-1}s_n}^{a_{n-1}} (v^*(s_n) - v_\pi(s_n)) \\
&\xrightarrow{n\to\infty} 0.
\end{aligned}$$

We still need to show that $v^* = v_{\pi^*}$. For this, we already have that v^* satisfies

$$v^* = f(v^*) = f_{\pi^*}(v^*)$$

with f_{π^*} as in (5.4). Therefore, Corollary 5.10 gives $v^* = v_{\pi^*}$. □

In order to apply the above theorem, note that we can compute v^*, which we identified as the value function for some optimal policy. From Theorem 5.13.3 we know how to obtain an optimal policy by evaluating an argmax over all possible $\pi \in \mathbb{\Delta}_{\mathbb{A}}^{\mathbb{S}}$. We now show that we can restrict ourselves to deterministic policies (i.e. $\pi_s^*(a) \in \{0, 1\}$ for all a, s) for finding the optimum.

Proposition 5.14 (Greedy deterministic optimal polycies). *Let v^* be as in Theorem 5.13.1 and*

$$g^*(s,a) = r(s,a) + \gamma \sum_{s'} P_{ss'}^a v^*(s')$$

as well as $a^*(s) = argmax_a g^*(s,a)$. Then,

$$\pi_s^*(a) = \begin{cases} 1, & a = a^*(s), \\ 0, & otherwise \end{cases}$$

is an optimal (and deterministic) policy.

Proof. We have already shown that any policy with

$$\pi_s = \mathrm{argmax}_{\pi_s} \sum_a \pi_s(a)\Big(r(s,a) + \gamma \sum_{s'} P_{ss'}^a v^*(s')\Big) = \mathrm{argmax}_{\pi_s} \sum_a \pi_s(a) q^*(s,a)$$

is optimal. For the policy given in the statement this is clearly the case. □

Remark 5.15 (Algorithm to find an optimal policy). We can now provide an algorithm for finding an optimal (deterministic) policy; see Algorithm 3. This is now just a corollary of Theorem 5.13 and Proposition 5.14.

Note that f in (5.6) is not linear in v. As a consequence, we cannot compute a solution of $v = f(v)$ using matrix inversion. However, since f is a contraction, we can still use the fixed point theorem.

Algorithm 3 FINDING A DETERMINISTIC OPTIMAL POLICY IN A MDP

INPUT: Input $\varepsilon > 0$, $\gamma \in [0, 1)$, $r \in \mathbb{R}^{\mathbb{S}\times\mathbb{A}}$, $P^a_{s\cdot} \in \mathbb{A}_{\mathbb{S}}$ for $s \in \mathbb{S}, a \in \mathbb{A}$
OUTPUT: π^* deterministic optimal policy and its value v^*

$v_{\text{old}} = 0 \in \mathbb{R}^{\mathbb{S}}, g = 0 \in \mathbb{R}^{\mathbb{S}\times\mathbb{A}}$
repeat
  $g(s, a) = r(s, a) + \gamma \sum_{s'} P^a_{ss'} v_{\text{old}}(s')$ for all $s \in \mathbb{S}, a \in \mathbb{A}$
  $v_{\text{new}}(s) = \max_a g(s, a)$ for all $s \in \mathbb{S}$
until $|v_{\text{new}} - v_{\text{old}}| < \varepsilon$
$\pi_s(a^*) = 1$ iff $a^* = \text{argmax}_a g(s, a)$ and $\pi_s(a) = 0$ otherwise
return $\pi^* = \pi, v^* = v_{\text{new}}$

Example 5.16 (Moving robot 3). We continue with the moving robot from Examples 5.1 and 5.1. Again, look at the accompanying `R-markdown` file from pages 83 in order to look at the procedure to compute an optimal strategy.

5.4 Basic Monte Carlo techniques

For evaluating and finding optimal policies, we have assumed that the model is known. By this, we mean that we can access $r(s, a)$ and $P^a_{s\cdot}$ for all values of $s \in \mathbb{S}$ and $a \in \mathbb{A}$ and use these quentities in our calculations. In practise, however, the state and action space needs to be explored before the reward can be known. Reinforcement learning without knowing details of the reward is denoted *model-free reinforcement learning*. We start with this notion here, and will continue also in the next subsection.

As we have seen, policy evaluation is fundamental for improving and optimizing policies, we describe how to evaluate policies in the model-free setting. From (5.2), and if $A \sim \pi_s$, we read

$$v_\pi(s) = \mathbf{E}[G_0|S_0 = s] = \mathbf{E}[\mathbf{E}[G_0|S_0 = s, A_0]|S_0 = s] = \mathbf{E}[g_\pi(s, A)].$$

So, for policy evaluation, we need to approximate $g_\pi(s, a) = \mathbf{E}[G_0|S_0 = s, A = 0 = a]$ for all $s \in \mathbb{S}$ and $a \in \mathbb{A}$. This can be done using the law of large numbers.

Remark 5.17 (Basic Monte Carlo policy evaluation). Since G_0 can be observed along any path of X, we can approximate $\mathbf{E}[g_\pi(s, A)]$ as follows (for a policy π with $\pi_s(a) > 0$ for all $s \in \mathbb{S}, a \in \mathbb{A}$). Assume we have access to independent realizations $X^1, ..., X^n$ (for some large n) of paths of the Markov chain X, starting in all possible $(s, a) \in \mathbb{S} \times \mathbb{A}$. Then, we may write

$$G^i_0 = \sum_{k=0}^{\infty} \gamma^k r(X^i_k)$$

for the cumulative reward of the ith path. and

$$\widehat{g}_\pi(s,a) = \frac{1}{|\mathcal{N}_{(s,a)}|} \sum_{i \in \mathcal{N}_{(s,a)}} G_0^i \quad \text{with} \quad \mathcal{N}_{(s,a)} = \{i : X_0^i = (s,a)\}$$

as a proxy for $g_\pi(s,a)$. The law of large numbers implies that

$$\widehat{g}_\pi(s,a) \to \mathbf{E}[G_0 | S = s, A = a] = g_\pi(s,a)$$

as long as $n \to \infty$.

In the basic Monte Carlo policy evaluation, every path is only used in the estimate of $g_\pi(s,a)$ for a single pair (s,a). This can be done more efficiently as follows:

Remark 5.18 (Sample-efficient Monte Carlo policy evaluation). Given $s \in \mathbb{S}$ and $a \in \mathbb{A}$, we observe $X_t^i = (s,a)$ for various times t and independent copies i. By the Markov property, we can use any such visit for the approximation of $g_\pi(s,a)$. Setting

$$G_t^i = \sum_{k=0}^{\infty} \gamma^k r(X_{t+k}^i)$$

for the cumulative reward of the ith path starting at time r, we find that

$$\widetilde{g}_\pi(s,a) = \frac{1}{|\mathcal{M}_{(s,a)}|} \sum_{(i,t) \in \mathcal{M}_{(s,a)}} G_t^i \quad \text{with} \quad \mathcal{M}_{(s,a)} = \{(i,t) : X_t^i = (s,a)\}$$

as a proxy for $g_\pi(s,a)$. Since every path is used multiple times for policy evaluation, this is more sample efficient than the algorithm from Remark 5.17.

5.5 Temporal difference and Q-learning

The Monte Carlo methods from the last subsection had the drawback that we need the paths $X^1, ..., X^n$ before we can even start to evaluate or optimize our policy. Rather, we would like to optimize the policy as we get more and more paths. This is achieved by temporal difference and Q-learning learning as we will see now. Before we can start, we state a Theorem without proof, which we need in otder to show convergence of the learning algorithms.

Theorem 5.19 (Dvoretzky's extended Theorem). *Let $\mathbb{S}$ be finite and*

$$\Delta_{t+1}(s) = (1 - B_t(s))\Delta_t(s) + C_t E_t(s), \qquad s \in \mathbb{S}$$

for some $0 \le B_t \le C_t$. If, for all $s \in \mathbb{S}$, $(E_t(s))_{t=1,2,...}$ is bounded, $\sum B_t(s) = \infty, \sum B_t(s)^2 < \infty$,

$$\max_s |\mathbf{E}[E_t(s) | \mathcal{H}_t]| \le \gamma \max_s |\Delta_t(s) \tag{5.7}$$

with $\mathcal{H}_t = (\Delta_t, \Delta_{t-1}, ..., E_{t-1}, ..., B_{t-1}, ..., C_{t-1}, ...)$ for some $\gamma \in [0,1)$, then $\mathbf{P}(|\Delta_t| > \varepsilon) \xrightarrow{t \to \infty} 0$ for all $\varepsilon > 0$.

Remark 5.20 (Generalizations). In fact, the assumption of bounded $(E_t)_{t=1,2,...}$ can be relaxed to $\mathbf{V}[E_t(s)|\mathcal{H}_t] \leq c(1 + \max_s |\Delta_t(s)|)^2$ for some $c > 0$ and the convergence is almost sure, i.e. $\Delta_t(s) \xrightarrow{t\to\infty} 0$ almost surely for all s. However, we do not need the relaxed assumption and the formulation of the stronger convergence requires some measure theory.

Example 5.21 (Online estimation of the expectation). Assume we are given real-valued (bounded) $X_1, X_2, ...$ which are independent and identically distributed, and we want to find $\mathbf{E}[X]$. Define iteratively (using e.g. $W_1 = X_1$)

$$W_{t+1} = W_t - \eta_t(W_t - X_{t+1}) = (1 - \eta_t)W_t + \eta_t X_{t+1}.$$

Set $\Delta_t := W_t - \mathbf{E}[X_1]$, we see that

$$\Delta_{t+1} = (1 - \eta_t)\Delta_t + \eta_t(X_{t+1} - \mathbf{E}[X_1])$$

and $W_t \xrightarrow{t\to\infty} \mathbf{E}[X_1]$ iff $\Delta_t \xrightarrow{t\to\infty} 0$. In order to show the latter convergence, we apply Dvoretzky's Theorem with $E_t = X_{t+1} - \mathbf{E}[X_1]$ with $\mathbf{E}[E_t|\mathcal{H}_t] = 0$. Therefore, if we take η_t such that $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$, we find that $W_t \xrightarrow{t\to\infty} \mathbf{E}[X_1]$. A special choice is $\eta_t = \frac{1}{t+1}$ since then,

$$W_{t+1} = \frac{tW_t + X_{t+1}}{t+1}, \text{ leading to } W_t = \frac{X_1 + \cdots + X_t}{t}.$$

See also the `R-markup` file on page 91.

Let us now come to temporal difference learning. This is a recursion which approximates the value function of a policy.

Theorem 5.22 (Temporal difference learning). *Let $(\pi_s)_{s\in\mathbb{S}}$ be a policy such that every state can be visited from any other non-terminal state when following the policy. Let $X = (X_t)_{t=1,2,...}$ be a Markov decision process, which either has no terminal states, or is restarted randomly if it hits a terminal state. Moreover, let $(V_0(s))_{s\in\mathbb{S}}$ be arbitrary and*

$$V_{t+1}(s) = V_t(s) - \eta_t 1_{S_t=s}\Big(V_t(s) - (r(s, A_t) + \gamma V_t(S_{t+1}))\Big), \qquad s \in \mathbb{S}.$$

Then, for any $\varepsilon > 0$, if $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$,

$$\mathbf{P}(|V_t(s) - v_\pi(s)| > \varepsilon) \xrightarrow{t\to\infty} 0 \text{ for all } s \in \mathbb{S}.$$

Remark 5.23. Note that the update of the value-function happens after each step of the Markov Decision process. We do not need to wait until a terminal state has been reached. This is also called online learning.

Proof. For the application of Dvoretzky's extended Theorem to temporal difference dearning, we define

$$\begin{aligned}
\Delta_t(s) &:= V_t(s) - v_\pi(s), \\
B_t(s) &:= C_t(s) := \eta_t 1_{S_t=s}, \\
E_t(s) &:= 1_{S_t=s}(r(S_t, A_t) + \gamma V_t(S_{t+1}) - v_\pi(s)).
\end{aligned}$$

Temporal difference learning is the recursion

$$\begin{aligned} V_{t+1}(s) &= V_t(s) - \eta_t 1_{S_t=s}\Big(V_t(s) - (r(S_t, A_t) + \gamma V_t(S_{t+1}))\Big) \\ &= (1 - B_t(s))V_t(s) + B_t(s)(r(S_t, A_t) + \gamma V_t(S_{t+1})), \end{aligned}$$

which is equivalent to

$$\Delta_{t+1}(s) = (1 - B_t(s))\Delta_t(s) + B_t(s)E_t(s).$$

We compute, using (5.1),

$$\begin{aligned} \mathbf{E}[E_t(s)|\mathcal{H}_t] &= 1_{S_t=s}(\mathbf{E}[r(S_t, A_t) + \gamma V_t(S_{t+1})|S_t, A_{t-1}, S_{t-1}, ...] - v_\pi(s)) \\ &= 1_{S_t=s} \sum_{a,s'} \pi_s(a)(r(s, a) + \gamma P^a_{ss'} V_t(s') - r(s, a) - \gamma P^a_{ss'} v_\pi(s')) \\ &= 1_{S_t=s}\gamma \sum_{a,s'} \pi_s(a) P^a_{ss'} \Delta_t(s'). \end{aligned}$$

From this, we see that

$$\max_s |\mathbf{E}[E_t(s)|\mathcal{H}_t]| \leq \gamma \max |\Delta_t(s)|$$

and the result is a consequence of Dvoretzky's Theorem. □

Rather than learning the value function, we want to find an optimal policy using online learning. For this, we need to have an approximation of g^* from Proposition 5.14. This is the result of Q-learning.

Theorem 5.24 (Q-learning). *Let $(\pi_s)_{s\in\mathbb{S}}$ be a policy such that every state can be visited from any other non-terminal state when following the policy. Let $X = (X_t)_{t=1,2,...}$ be a Markov decision process, which either has no terminal states, or is restarted randomly if it hits a terminal state. Moreover, let $(G_0(s, a))_{s\in\mathbb{S},a\in\mathbb{A}}$ be arbitrary and*

$$G_{t+1}(s, a) = G_t(s, a) - \eta_t 1_{S_t=s,A_t=a}\Big(G_t(s, a) - (r(S_t, A_t) + \gamma \max_{a'} G_t(S_{t+1}, a'))\Big), \qquad s \in \mathbb{S}, a \in \mathbb{A}.$$

Then, for any $\varepsilon > 0$, if $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$,

$$\mathbf{P}(|G_t(s, a) - g^*(s, a)| > \varepsilon) \xrightarrow{t\to\infty} 0 \text{ for all } s \in \mathbb{S}, a \in \mathbb{A},$$

where g^* is a function such that $s \mapsto \max_a g^*(s, a)$ solves the Bellmann optimality equality from Theorem 5.13.

Remark 5.25.

1. Note that we are following some (non-optimal) π here, but are able to obtain an optimal π from our observations. This is also called off-policy learning.
2. Since the result gives a solution of the Bellmann optimality equation, we can use Proposition 5.14 in order to find an optimal policy.

Proof. The proof has two main arguments. First, we will show using Dvoretzky's extended Theorem that G_t converges to the unique function g^* satisfying

$$g^*(s,a) = r(s,a) + \gamma \mathbf{E}[\max_{a'} g^*(S_{t+1}, a') | S_t = s, A_t = a]. \tag{5.8}$$

(The fact that there is a unique such function is again an application of the fixed point theorem.) Once this is achieved, set $v^* : s \mapsto \max_a g^*(s,a)$, and note that

$$\begin{aligned} v^*(s) &= \max_a \Big(r(s,a) + \gamma \mathbf{E}[\max_{a'} g^*(S_{t+1}, a') | S_t = s, A_t = a] \Big) \\ &= \max_a \Big(r(s,a) + \gamma \sum_{s'} P^a_{ss'} v^*(s') \Big) \end{aligned}$$

and this is the Bellmann optimality equation, noting that we know that there are deterministic optimal strategies. So, it remains to prove convergence to the above g^*. Again, we apply Dvoretzky's extended Theorem and we define

$$\begin{aligned} &\Delta_t(s,a) := G_t(s,a) - g^*(s,a), \\ &B_t(s,a) := C_t(s,a) := \eta_t 1_{S_t = s, A_t = a}, \\ &E_t(s,a) := 1_{S_t = s, A_t = a}(r(s,a) + \gamma \max_{a'}(G_t(S_{t+1}, a')) - g^*(s,a)). \end{aligned}$$

Q-learning is the recursion

$$\begin{aligned} G_{t+1}(s,a) &= G_t(s,a) - \eta_t 1_{S_t = s, A_t = a} \Big(G_t(s,a) - (r(s,a) + \gamma \max_{a'} G_t(S_{t+1}, a')) \Big) \\ &= (1 - B_t(s,a)) G_t(s,a) + B_t(s,a)(r(s,a) + \gamma \max_{a'} G_t(S_{t+1}, a')), \end{aligned}$$

or equivalently,

$$\Delta_{t+1}(s,a) = (1 - B_t(s,a)) V_t(s,a) + B_t(s,a) E_t(s,a).$$

In order to show (5.7), note that for any g_1, g_2

$$|\max_{a'} g_1(a') - \max_{a''} g_2(a'')| \leq \max_{a'} |g_1(a') - g_2(a')| \tag{5.9}$$

In order to see this, assume wlog $\max_{a''} g_2(a'') \leq \max_{a'} g_1(a')$. Then write, for any a',

$$g_1(a') - \max_{a''} g_2(a'') \leq g_1(a') - g_2(a') \leq |g_1(a') - g_2(a')|.$$

Taking the maximum over all a' shows (5.9). We use this to write

$$\begin{aligned} |\mathbf{E}[E_t(s,a) | \mathcal{H}_t]| &= 1_{S_t = s, A_t = a} |r(s,a) + \gamma \mathbf{E}[\max_{a'} G_t(S_{t+1}, a') | S_t, A_t, S_{t-1}, ...] - g^*(s,a)| \\ &= 1_{S_t = s, A_t = a} \gamma |\mathbf{E}[\max_{a'} G_t(S_{t+1}, a') - \max_{a'} g^*(S_{t+1}, a'') | S_t = s, A_t = a]| \\ &\leq 1_{S_t = s, A_t = a} \gamma \mathbf{E}[|\max_{a'} G_t(S_{t+1}, a') - \max_{a'} g^*(S_{t+1}, a'')| \, | S_t = s, A_t = a]| \\ &\leq 1_{S_t = s, A_t = a} \gamma \mathbf{E}[\max_{a'} |\Delta_t(S_{t+1}, a)| \, | S_t = s, A_t = a]| \end{aligned}$$

It remains to take the maximum over all s, a on both sides in order to see that $\max_{s,a} |\mathbf{E}[E_t(s,a) | \mathcal{H}_t]| \leq \gamma \max_{s,a} |\Delta_t(s,a)|$, which finishes the proof. □

Appendix

We use this appendix in order to recall some notions from introductory lectures in mathematics and probability theory.

A Calculus

Let $\Theta \subseteq \mathbb{R}^d$ be open[18] and $f : \Theta \to \mathbb{R}^n$ be continuous. For $x \in \Theta$ we define the *derivative* $Df(x)$, if it exists, as the unique element of $\mathbb{R}^{n\times d}$ such that (note that $Df(x)h \in \mathbb{R}^n$ is the multiplication of a matrix and a vector)

$$\lim_{h\to 0} \frac{f(x+h) - f(x) - Df(x)h}{||h||} = 0. \tag{A.1}$$

In the special case $n = 1$, we call $\nabla f := Df$ the *gradient of* f.

Definition A.1 (Partial derivative, Gradient, Hessian). *Let $\Theta \subseteq \mathbb{R}^d$ be open and $f \in \mathcal{C}(\Theta) := \{f : \Theta \to \mathbb{R}$ continuous$\}$. Then, for $i = 1, ..., d$ and the ith unit vector e_i,*

$$\frac{\partial f}{\partial x_i} f(x) := \lim_{h\to 0} \frac{f(x + he_i) - f(x)}{h},$$

if the limit exists. If the limit exists for all $x \in \Theta$ and is continuous, we say $f \in \mathcal{C}^1(\Theta)$. (Then,

$$\nabla f := \Big(\frac{\partial f}{\partial x_i}\Big)_{i=1,...,d}.\Big)$$

If, in addition, all limits in

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} := \frac{\partial}{\partial x_i}\frac{\partial f(x)}{\partial x_j}$$

exist, we say that $f \in \mathcal{C}^2(\Theta)$ and write

$$\nabla^2 f := \Big(\frac{\partial^2 f}{\partial x_i \partial x_j}\Big)_{i,j=1,...,d}$$

for the Hessian *of f.*

We state some results without proof, so if you don't remember these, look them up.

Theorem A.2 (Schwartz' Theorem, maxima, minima). *Let $\Theta \subseteq \mathbb{R}^d$ be open.*

1. *Schwartz' Theorem: If $f \in \mathcal{C}^2(\Theta)$, then $\nabla^2 f$ is symmetric, i.e. for all $x \in \Theta$, $\nabla^2 f(x)$ is a symmetric matrix or*

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}, \qquad 1 \le i, j \le d.$$

2. *If $f \in \mathcal{C}^1(\Theta)$ has an optimum (maximum or minimum) at $x \in \Theta$, then $\nabla f(x) = 0$.*

[18] $\Theta \subseteq \mathbb{R}^d$ is open if $x \in \Theta \Longrightarrow \exists \varepsilon > 0 : B_\varepsilon(x) \subseteq \Theta$, where $B_\varepsilon(x)$ is the open ball with radius ε around x.

3. *If $x \in \Theta$ is such that $\nabla f(x) = 0$ and $\nabla^2 f(x)$ is positive (negative) definite, i.e. $\xi^\top \nabla^2 f(x)\xi > 0$ (... < 0) for all $0 \neq \xi \in \mathbb{R}^d$ or all eigenvalues of $\nabla^2 f(x)$ are positive (negative), then f has a minimum (maximum) at x.*

Remark A.3 (Multi-dimensional Taylor formula). In some cases, it is good to remember the multi-dimensional Taylor formula for the approximation of some $f \in \mathcal{C}^2(\Theta)$. It reads

$$f(y) = f(x) + \nabla f(x)^\top (y - x) + (y - x)^\top \nabla^2 f(x)(y - x) + R_2 f(y, x),$$

where $R_2 f$ is the remainder. Usually, it is $o(|y - x|^2)$.

B Probability theory

Here, we collect some results on basic probability theory, hopefully covered in a first course in the subject.

Lemma B.1 (Cauchy–Schwartz Ungleichung). *Let X, Y be real-valued random variables with finite second moments. Then,*

$$\mathbf{Cov}[X, Y]^2 \leq \mathbf{E}[|(X - \mathbf{E}[X])(Y - \mathbf{E}[Y])|]^2 \leq \mathbf{Var}[X]\mathbf{Var}[Y].$$

Remark B.2 (Notation). For some X and its distribution $\mathbf{P}$, we write $\mathbf{P}(X \in dx) \sim f(x)$ iff $\mathbf{P}(X \in dx) = f(x)/\sum_z f(z)$ or $\mathbf{P}(X \in dx) = f(x)/\int f(z)dz$.

The multivariate normal distribution

In several calculations, we will make use of the multivariate normal distribution. Here, we recall some of its properties. The proofs can e.g. be found in my lecture notes for *Stochastik für Informatiker*.

Definition B.3 (Multivariate normal distribution). *Let $\mu \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n \times n}$ symmetric and positiv definite. If for some $\mathbb{R}^n$-valued random variable X*

$$\mathbf{P}(X \in dx_1, ..., dx_n) = \frac{1}{\sqrt{(2\pi)^n det(\Sigma)}} \exp\Big(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu)\Big) dx_1 \cdots dx_n,$$

then, X is called multi-variat normally distributed with expectation vector μ and covariance matrix Σ. Then, we write $X \sim N(\mu, \Sigma)$. If $\mu = 0$ and $\Sigma = I$ (the n-dimensional unit matrix), X is called multivariante standard normally distributed. Standard-normalverteilt.

Remark B.4 (Alternative definition). [Extension using characteristic functions] If X has values in $\mathbb{R}^n$ and a density, the above definition is in fact equivalent to

$$\mathbf{E}[e^{it^\top X}] = e^{it^\top \mu - \frac{1}{2} t^\top \Sigma t}, \qquad t \in \mathbb{R}^n. \tag{B.1}$$

Taking this equation as a definition (i.e. defining the mutlivariate normal distribution as the unique distribution for which (B.1) holds for all $t \in \mathbb{R}^n$) is even more general, since (B.1) does not imply that X has a density.

Proposition B.5 (Linear transformation of normally distributed random variable). *Let $X \sim N(\mu, \Sigma)$ for $\mu \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n\times n}$ symmetric and positive definite.. Then,*

$$\mathbf{E}[X_i] = \mu_i, \qquad \mathbf{Cov}[X_i, X_j] = \Sigma_{ij}.$$

Furthermore, let $\nu \in \mathbb{R}^k$ and $B \in \mathbb{R}^{k\times n}$ (for some $k \leq n$), such that $B\,\Sigma\,B^\top$ is invertible. Then,

$$Y := \nu + B\,X \sim N(\nu + B\,\mu, B\,\Sigma\,B^\top).$$

Corollary B.6 (Independent and uncorrelated normally distributed random variables). *Let X be multivariate normally distributed. Then, X_i, X_j are independent if they are uncorrelated.*

Proposition B.7 (Conditional normal distribution is normal). *Let $(X, Y) \sim N((\mu, \Sigma)$ be an $\mathbb{R}^{m+n}$-valued random variable with $\mu = (\mu_X, \mu_Y)$ and*

$$\Sigma = \begin{pmatrix} \Sigma_{XX} & \Sigma_{XY} \\ \Sigma_{YX} & \Sigma_{YY} \end{pmatrix}.$$

Then, the distribution of X conditional of Y has a

$$N\big(\mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(Y - \mu_Y), \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX}\big) - \textit{distribution.}$$

References

[1] Anthony, M., P. L. Bartlett, P. L. Bartlett, et al. (1999). *Neural network learning: Theoretical foundations*, Volume 9. cambridge university press Cambridge.

[2] Braga-Neto, U. (2020). *Fundamentals of pattern recognition and machine learning*. Springer.

[3] Cover, T. and P. Hart (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory 13*(1), 21–27.

[4] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems 2*(4), 303–314.

[5] Devroye, L., L. Györfi, and G. Lugosi (2013). *A probabilistic theory of pattern recognition*, Volume 31. Springer Science & Business Media.

[6] Dougherty, G. (2012). *Pattern recognition and classification: an introduction*. Springer Science & Business Media.

[7] Gower, R. M. (2018). Convergence theorems for gradient descent. *Lecture notes for Statistical Optimization*.

[8] Guilhoto, L. F. (2018). An overview of artificial neural networks for mathematicians.

[9] Han, X. (2018). A mathematical introduction to reinforcement learning.

[10] Hastie, T., R. Tibshirani, J. H. Friedman, and J. H. Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*, Volume 2. Springer.

[11] James, G., D. Witten, T. Hastie, and R. Tibshirani (2013). *An introduction to statistical learning*, Volume 112. Springer.

[12] Lehmann, E. L. and J. P. Romano (2022). *Testing statistical hypotheses* (4th ed.). Springer.

[13] Murphy, K. P. (2022). *Probabilistic Machine Learning: An introduction*. MIT Press.

[14] Petersen, P. C. (Version 2022/04/18). Neural network theory.

[15] R Core Team (2022). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.

[16] Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. MIT press.

[17] Vapnik, V. (1999). *The nature of statistical learning theory*. Springer science & business media.

[18] Zhao, S. (Version August 2022, url = https://github.com/MathFoundationRL/Book-Mathmatical-Foundation-of-Reinforcement-Learning). Mathematical foundation of reinforcement learning.

Gradient descent

Peter Pfaffelhuber

2022-11-26

Here, we will use **R Markdown** (see http://rmarkdown.rstudio.com) in order to implement the algorithms for deterministic and stochastic grdient descent from the lecture notes; see Section 2. When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document.

Some tools for gradient descent

We will use the gradient of some function

$$f: \begin{cases} \mathbb{R}^d & \to \mathbb{R} \\ \beta & \mapsto f(\beta) = \frac{1}{n}\sum_{i=1}^n f_i(\beta) \end{cases}.$$

For this, we will use $f = \log \ell$ (as a function of the parameters β) from logistic regression; see Section 2.1. In our implementation, we list $\nabla \log \ell_i$ for the rows which we will need:

In order to set the scene, recall that the ML-estimator for the parameters β in logistic regression is unique. Moreover, assume wlog that $0 \in \mathbb{S}$ and with $\beta_{\cdot 0} = 0 \in \mathbb{R}^k$,

$$\frac{1}{n}\ell(\beta, y) = \frac{1}{n}\sum_{i=1}^n x_{i\cdot}\beta_{\cdot y_i} - \log\Big(1 + \sum_{d\neq 0} e^{x_{i\cdot}\beta_{\cdot d}}\Big),$$

$$\nabla_\beta \frac{1}{n}\ell(\beta, y) = \frac{1}{n}\Big(\sum_{i=1}^n x_{i\cdot}\Big(1_{y_i=c} - \frac{e^{x_{i\cdot}\beta_{\cdot c}}}{1 + \sum_{d\neq 0} e^{x_{i\cdot}\beta_{\cdot d}}}\Big)\Big)_{0\neq c\in\mathbb{S}} \in \mathbb{R}^{k\times\mathbb{S}\setminus\{0\}}.$$

Let us implement the negative gradient of the log-likelihood, since this is what we are going to minimize. Here, rows is a vector with the numbers of items for which the derivative should be computed. We split the computation in these rows from the rest of computing the log-likelihood, since we can then reuse this first function for stochastic gradient descent.

```
nablaell_logreg_row<-function(beta, rows, x, y){
  if(is.vector(beta)){ # the binary case
  beta = matrix(beta, ncol=1)
  }
  res = array(0, dim = c(nrow(beta), ncol(beta), length(rows)))
  for(i in 1:length(rows)) {
    res[,,i] = x[rows[i],] %o% ((y[rows[i]] == 1:ncol(beta)) - exp(x[rows[i],]%*% beta)/
                                  (1 + sum(exp(x[rows[i],]%*% beta))))
  }
  -res
}
```

Deterministic gradient descent

Assume (for some $d \in \mathbb{N}$), we are given some $f \in \mathcal{C}^1(\mathbb{R}^d)$, as well as $\nabla f : \mathbb{R}^d \to \mathbb{R}^d$, $x_0 \in \mathbb{R}^d$, and a constant learning rate η, and some small $\varepsilon > 0$, which determines if we continue with finding the maximum of f. Then, the following code implements deterministic gradient descent:

Deterministic gradient descent is based on using all functions, i.e. we need to use all rows in the function above.

```
nablaell_logreg<-function(beta, x, y){
  n = nrow(x)
  apply(nablaell_logreg_row(beta, 1:n, x, y), 1:2, mean)
}
```

The function we have just defined will serve as nablaf in the actual function for deterministic gradient descent.

```
det_grad_desc<-function(nablaf, beta0, eta, varepsilon, maxStep, ...) {
  t = 1
  step = 0
  beta = beta_all = beta0
  cont = TRUE
  while(cont) {
    beta_new = beta - eta * nablaf(beta,...)
    step = step + 1
  if(sum(abs(beta_new - beta)) < varepsilon | step>maxStep) {
    cont = FALSE
  }
  beta=beta_new
  beta_all = cbind(beta_all, beta_new)
  }
  list(argmin = beta, beta_all = beta_all, steps=step)
}
```

Stochastic gradient descent

Actually, implementing stochastic gradient descent is not much different from the implementation of deterministic gradient descent. However, eta here needs to be a function, and b is the mini-batch size.

```
nablaell_logreg_minibatch<-function(beta, b, x, y){
  n = nrow(x)
  rows = sample(n, b)
  apply(nablaell_logreg_row(beta, rows, x, y), 1:2, mean)
}
```

Then, we have the main stochastic gradient descent routine.

```
stoch_grad_desc<-function(nablaf, beta0, eta, b, varepsilon, maxStep, ...) {
  t = 1
  step = 0
  beta = beta_all = beta0
  cont = TRUE
  while(cont) {
    beta_new = beta - eta(t) * nablaf(beta, b,...)
    step = step + 1
    if(sum(abs(beta_new - beta)) < varepsilon | step>maxStep) {
      cont = FALSE
    }
    beta=beta_new
    beta_all = cbind(beta_all, beta_new)
  }
  list(argmin = beta, beta_all = beta_all, steps=step)
}
```

Data example

We will use the above methods on some simulated data.

```
# number of items
n = 100
# true parameter
beta = c(4,1)
# simulate data
set.seed(1)
x = matrix(2*runif(2*n)-1, ncol=2)
p = 1 / (1+ exp(x %*% beta)) # p is the probability to be in class 0
y = (runif(n) > p) + 0
```

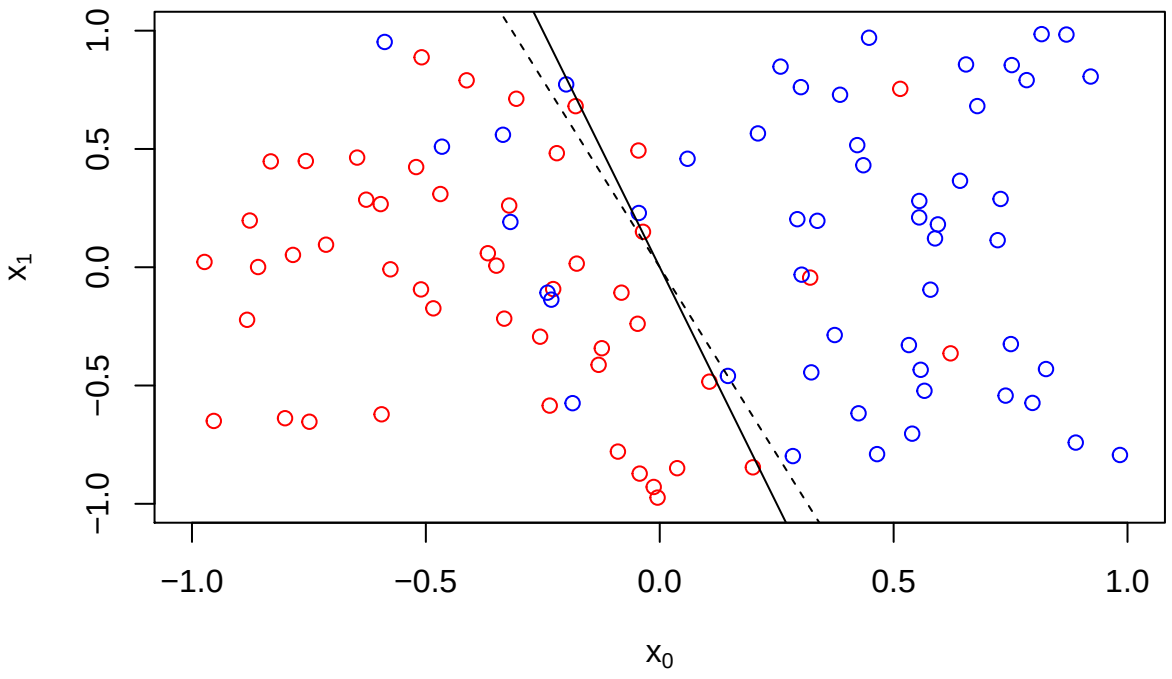
Here is what we do for deterministic gradient descent:

```
# initialization of optimization
beta0 = c(0,0)
eta=1
varepsilon=0.001
maxStep=1000
# Now we estimate beta using deterministic gradient descent
nablaf = nablaell_logreg
a = det_grad_desc(nablaf, beta0, eta, varepsilon, maxStep, x, y)
cat("Number of steps: ", a$steps)
```

```
## Number of steps:  305
```

We plot the resulting estimated class boundaries next.

```
plot(c(-1,1), c(-1,1), type="n", xlab=expression(x[0]), ylab=expression(x[1]))
points(x[y==0,1], x[y==0,2], col="red")
points(x[y==1,1], x[y==1,2], col="blue")
abline(0, -4, col="black", lty=1)
abline(0, -a$argmin[1,1] / a$argmin[2,1], col="black", lty=2)
```



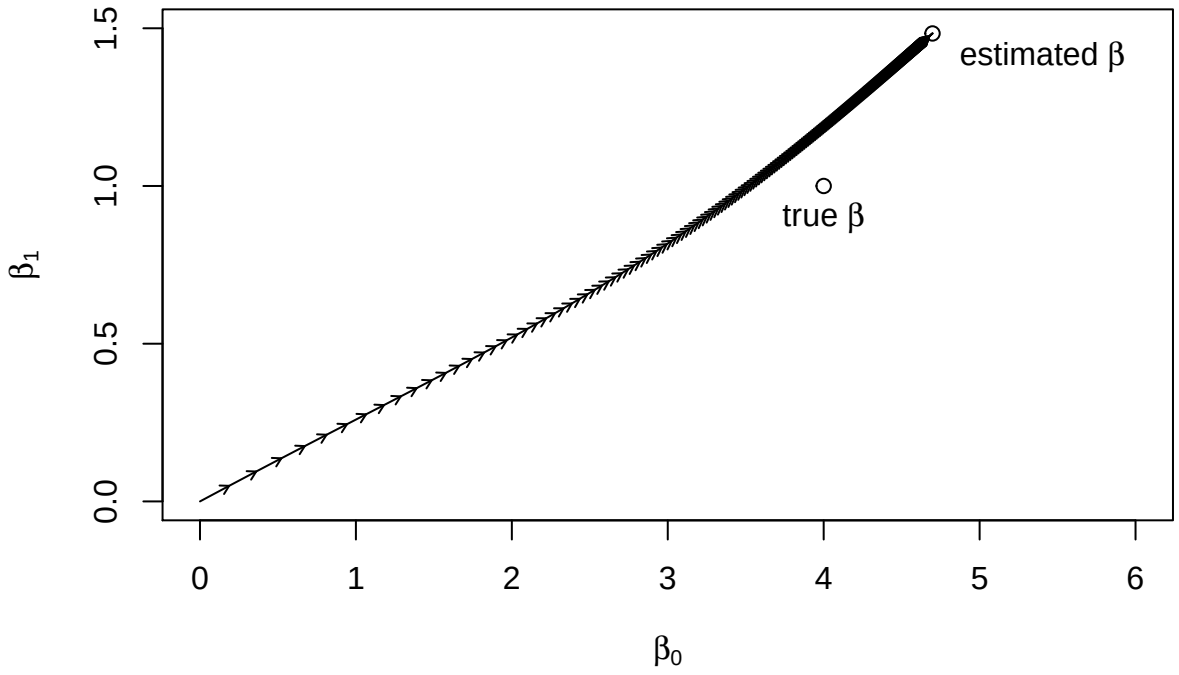
Then, we can also plot the path the algorithm takes in the β-space:

```
plot(c(0,1.5*beta[1]), c(0,1.5*beta[2]), type="n", xlab=expression(beta[0]), ylab=expression(beta[1]))
beta_path = a$beta_all
# only show k steps
#k=120
arrows(beta_path[1,1:(ncol(beta_path)-1)], beta_path[2,1:(ncol(beta_path)-1)], beta_path[1,2:(ncol(beta_
```

```
## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped
```

```
## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped
```

```
#arrows(beta_path[1,1:k], beta_path[2,1:k], beta_path[1,2:(k+1)], beta_path[2,2:(k+1)], length=0.05)
points(beta[1], beta[2])
text(beta[1], 0.9*beta[2], expression(paste("true ", beta)))
beta_est = a$beta_all[,a$steps]
points(beta_est[1], beta_est[2])
text(1.15*beta_est[1], .95*beta_est[2], expression(paste("estimated ", beta)))
```



We can also obtain the training error:

```
p_est = 1 / (1+ exp(x %*% a$argmin[,1]))
cat("For the estimated beta, from", n, " datapoints,", sum((p_est<0.5 & y ==0) | (p_est>0.5) & (y==1)),
```

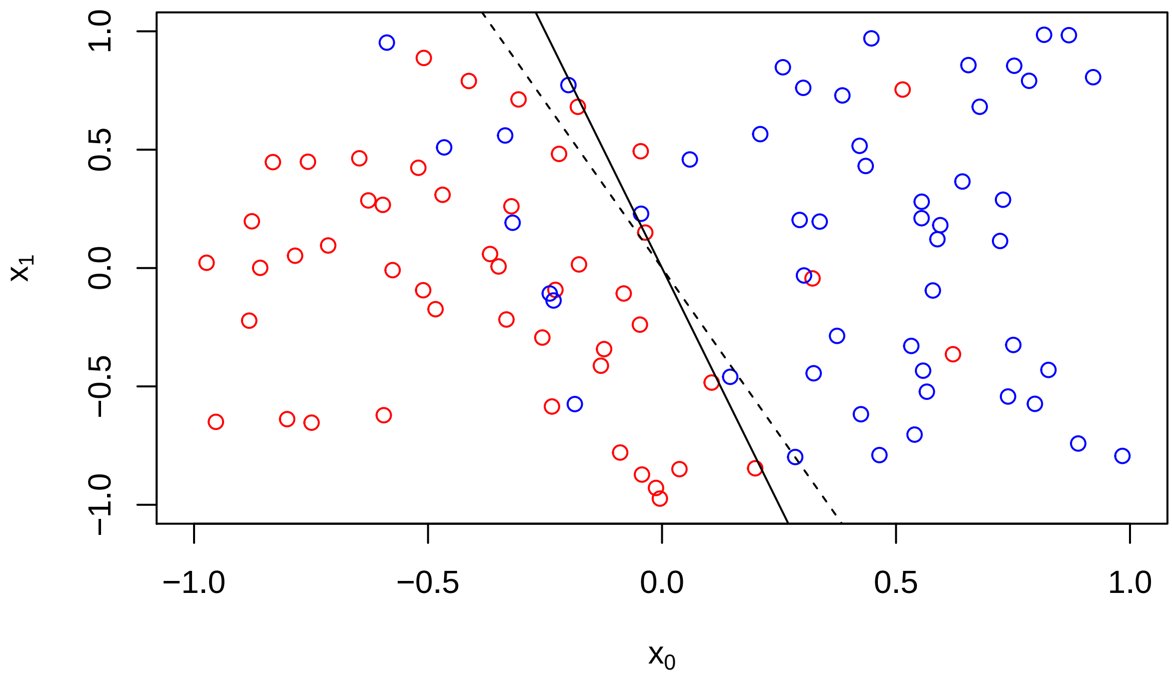
```
## For the estimated beta, from 100  datapoints, 13 are misclassified.
```

Here are the corresponding results for stochastic gradient descent:

```
beta0 = c(0,0)
varepsilon=0.001
maxStep=10000
eta<-function(t, eta0=.1) {
  eta0/t
}
b = 1
# Now we estimate beta using stochastic gradient descent
nablaf = nablaell_logreg_minibatch
a = stoch_grad_desc(nablaf, beta0, eta, b, varepsilon, maxStep, x, y)
```

Again, we can plot the resulting class boundary.

```
plot(c(-1,1), c(-1,1), type="n", xlab=expression(x[0]), ylab=expression(x[1]))
points(x[y==0,1], x[y==0,2], col="red")
points(x[y==1,1], x[y==1,2], col="blue")
abline(0, -4, col="black", lty=1)
abline(0, -a$argmin[1,1] / a$argmin[2,1], col="black", lty=2)
```




```
p_est = 1 / (1+ exp(x %*% a$argmin[,1]))
cat("For the estimated beta, from 100 datapoints,", sum((p_est<0.5 & y ==0) | (p_est>0.5) & (y==1)), "a
```

```
## For the estimated beta, from 100 datapoints, 14 are misclassified.
```

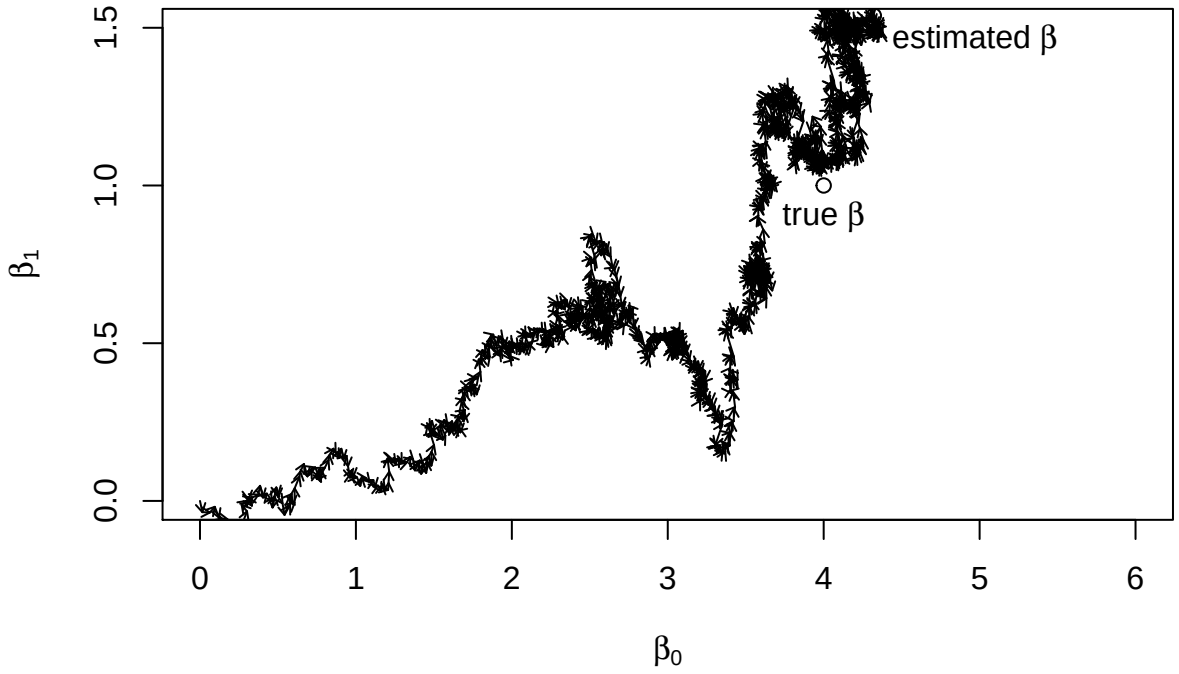
Here, the path to the optimum looks not as smooth.

```
plot(c(0,1.5*beta[1]), c(0,1.5*beta[2]), type="n", xlab=expression(beta[0]), ylab=expression(beta[1]))
beta_path = a$beta_all
arrows(beta_path[1,1:(ncol(beta_path)-1)], beta_path[2,1:(ncol(beta_path)-1)], beta_path[1,2:(ncol(beta_
```

```
## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped
```

```
## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped

## Warning in arrows(beta_path[1, 1:(ncol(beta_path) - 1)], beta_path[2, 1:
## (ncol(beta_path) - : zero-length arrow is of indeterminate angle and so skipped
```

```
points(beta[1], beta[2])
text(beta[1], 0.9*beta[2], expression(paste("true ", beta)))
beta_est = a$beta_all[,a$steps]
points(beta_est[1], beta_est[2])
text(1.15*beta_est[1], .95*beta_est[2], expression(paste("estimated ", beta)))
```



Nearest centroid and nearest neighbor classifier for the Iris dataset

Peter Pfaffelhuber

2022-11-26

The iris dataset

In machine learning, there are some famous datasets which are frequently used as examples. One of them is the Iris dataset, which gives measurements of three different Iris species. It is from the following paper: R. A. Fisher (1936). "The use of multiple measurements in taxonomic problems". Annals of Eugenics. 7 (2): 179–188. doi:10.1111/j.1469-1809.1936.tb02137.x. hdl:2440/15227.

You can load it into R simply by typing:

```
dat = iris
```

You might also want to have a summary.

```
summary(dat)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Alternatively, just look at the first few rows, display the dimensions and the names. Note that dat is a data.frame since it contains different variable types (numerical and strings):

```
# Showing the first few rows
head(dat)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# The dimensions of the dataframe
dim(iris)
```

```
## [1] 150   5
```

```
# Column names
colnames(dat)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

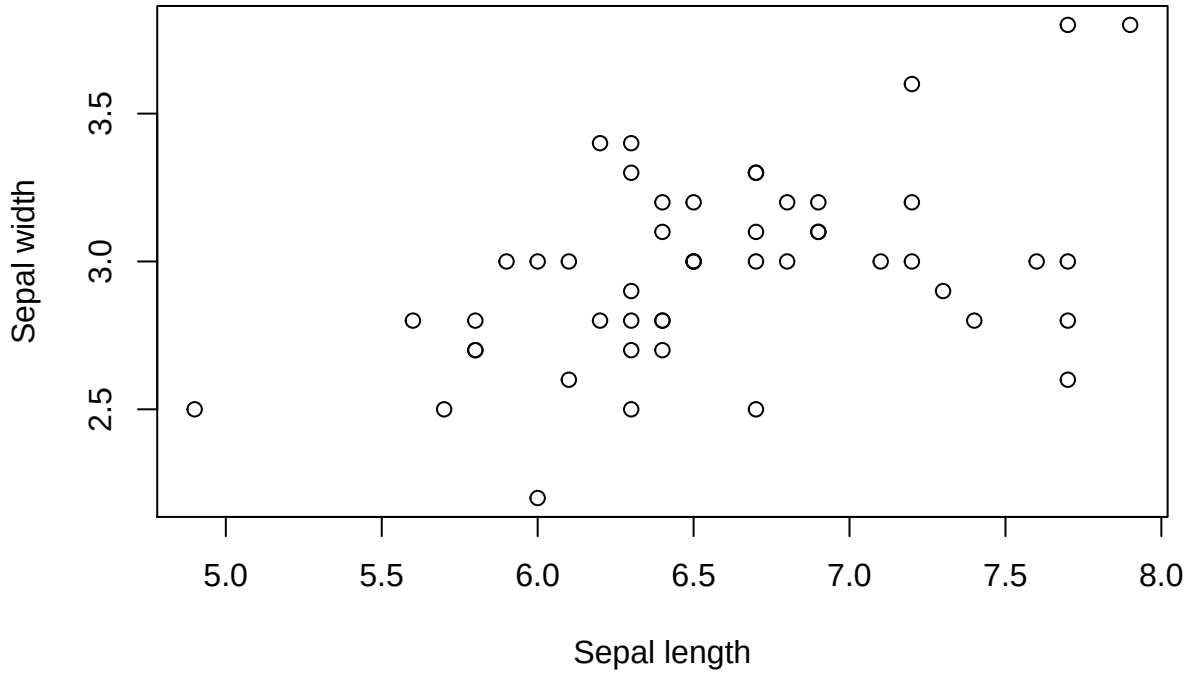
```
# type of datastructure
class(dat)
```

```
## [1] "data.frame"
```

Here, a total of 150 flowers of three different Iris species (setosa, versicolor and virginica) are studied, and their sepal (Kelchblatt) and petal (Blütenblatt) lengths and widths are recorded. We will use these measurements in order to classify the flowers into the three species.

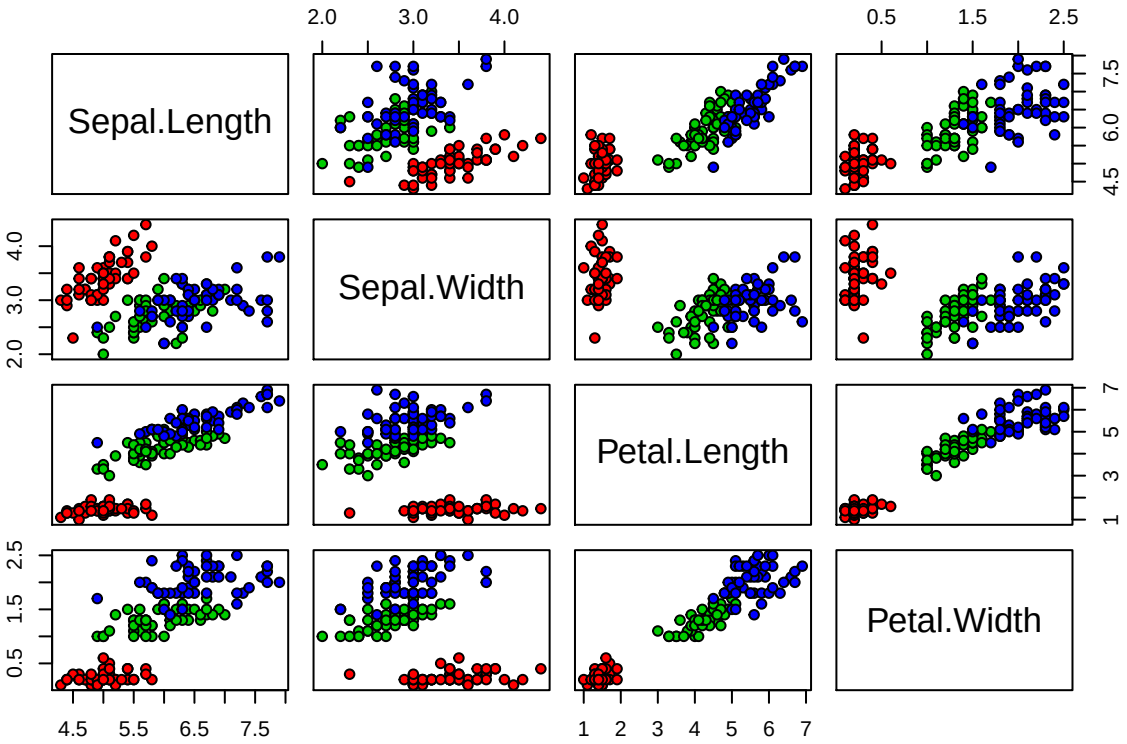
In order to get a feeling for the data, let us try to plot sepals and petals, distinguishing all three species. First, we do this for one pair of variables and only a single species:

```
# Plotting sepal.length against sepal.width for virginica
plot(dat$Sepal.Length[dat$Species == "virginica"], dat$Sepal.Width[dat$Species == "virginica"],
     xlab="Sepal length", ylab = "Sepal width")
```



Alternatively, we can use pairs in order to plot all variables for all species.

```
# Plots for all pairs of variables with different colors for the different species
pairs(dat[1:4], pch = 21, bg = c("red", "green3", "blue")[unclass(dat$Species)])
```



Methods Let us implement the nearest centroid classifier and the k-nearest neighbor classifier.

```
# x is the test data; x.train and y.train are features and labels from the training data
# x.train is assumed to be a matrix
nearest.centroid<-function(x, x.train, y.train) {
  classes = unique(y.train)
  centroids = t(simplify2array(by(x.train, y.train, colMeans)))
  dist_matrix = t(t(as.matrix(centroids))- as.vector(as.matrix(x, nrow=1)))
  dist = rowSums(dist_matrix^2)
  names(which.min(dist))
}

k.nearest.neighbor<-function(x, x.train, y.train, k) {
  classes = as.vector(unique(y.train))
  dist_matrix = t(t(as.matrix(x.train))- as.vector(as.matrix(x, nrow=1)))
  dist = rowSums(dist_matrix^2)
  neighbor_id = sort(dist, index.return=TRUE)$ix[1:k]
  hist.neighbors = table(y.train[neighbor_id])
  names(which.max(hist.neighbors))
}
```

Now, we can apply this to the iris dataset. As training data, we choose one dataset which is not correctly classified.

```
x.train = as.matrix(dat[1:4])
y.train = as.vector(as.matrix(dat[5]))
classes = unique(dat[5])
i=51
cat("i =", i, ": True class is", y.train[i], "\n")
```

```
## i = 51 : True class is versicolor
```

```
x = as.vector(dat[i,1:4])
cat("Nearest centroid is", nearest.centroid(x, x.train, y.train), ".\n")
```

```
## Nearest centroid is virginica .
```

```
cat("Nearest 3-neighbors gives", k.nearest.neighbor(x, x.train, y.train, 3), ".\n")
```

```
## Nearest 3-neighbors gives versicolor .
```

Looking at the whole dataset, we are interested in the training error.

```
# Let us compute the misclassification errors for nearest centroids on the training set
err_nc = 0
for(i in 1:nrow(dat)) {
  x = as.vector(dat[i,1:4])
  nc = nearest.centroid(x, x.train, y.train)
  if(y.train[i] != nc) err_nc = err_nc + 1
}
err_nc = err_nc / nrow(dat)
cat("nc: The misclassification error on the training set is", err_nc, ".\n")
```

```
## nc: The misclassification error on the training set is 0.07333333 .
```

```
# The same for k-nearest neighbors
for(k in 1:30) {
  err_nn = 0
  for(i in 1:nrow(dat)) {
    x = as.vector(dat[i,1:4])
    nn = k.nearest.neighbor(x, x.train, y.train, k)
    if(y.train[i] != nn) err_nn = err_nn + 1
  }
  err_nn = err_nn / nrow(dat)
  cat(k, "-nn: The misclassification error on the training set is", err_nn, ".\n")
}
```

```
## 1 -nn: The misclassification error on the training set is 0 .
## 2 -nn: The misclassification error on the training set is 0.02 .
## 3 -nn: The misclassification error on the training set is 0.04 .
## 4 -nn: The misclassification error on the training set is 0.04 .
## 5 -nn: The misclassification error on the training set is 0.03333333 .
## 6 -nn: The misclassification error on the training set is 0.02666667 .
## 7 -nn: The misclassification error on the training set is 0.02666667 .
## 8 -nn: The misclassification error on the training set is 0.02 .
## 9 -nn: The misclassification error on the training set is 0.02 .
## 10 -nn: The misclassification error on the training set is 0.02 .
## 11 -nn: The misclassification error on the training set is 0.02666667 .
## 12 -nn: The misclassification error on the training set is 0.02 .
## 13 -nn: The misclassification error on the training set is 0.02 .
## 14 -nn: The misclassification error on the training set is 0.02 .
## 15 -nn: The misclassification error on the training set is 0.01333333 .
## 16 -nn: The misclassification error on the training set is 0.01333333 .
## 17 -nn: The misclassification error on the training set is 0.02 .
## 18 -nn: The misclassification error on the training set is 0.02666667 .
## 19 -nn: The misclassification error on the training set is 0.02 .
## 20 -nn: The misclassification error on the training set is 0.02 .
## 21 -nn: The misclassification error on the training set is 0.02 .
## 22 -nn: The misclassification error on the training set is 0.02 .
```

```
## 23 -nn: The misclassification error on the training set is 0.02 .
## 24 -nn: The misclassification error on the training set is 0.02666667 .
## 25 -nn: The misclassification error on the training set is 0.02 .
## 26 -nn: The misclassification error on the training set is 0.02666667 .
## 27 -nn: The misclassification error on the training set is 0.02666667 .
## 28 -nn: The misclassification error on the training set is 0.03333333 .
## 29 -nn: The misclassification error on the training set is 0.02666667 .
## 30 -nn: The misclassification error on the training set is 0.04666667 .
```

Cross-validation

It is certainly not ideal if training and test data overlap. Since the method adopts to specifics only present in the training data, this leads to overfitting, i.e. the estimates are too well adapted to the training data and will not perform well on independent data. If many datasets are available, one way out would be as follows: We split the data at hand in two sets: the training and the test data. The training data is used to train the method, and the test data serves as independent dataset, where we can measure performance.

However, there are not enough datasets, and we will use cross-validation. Here, choosing some $k = 2, ..., n$, we split our dataset in k parts. Then, we perform k times the following: In iteration $i = 1, ..., k$, * use the ith part as testset, * and the rest as training set. Then, we can measure performance (i.e. misclassification error) averaging again over all datasets, but use the results from the k folds. This then gives the k-times cross-validation misclassification error. If $k = n$, the method is also called leave-one-out cross-validation.

Let us study this for the k-nearest-neighbor classifier on the iris dataset. We start with leave-one.out:

```
# leave-one-out-cross-validation for 3-nn
k=3
err_nn = 0
for(i in 1:nrow(dat)) {
  x = as.vector(dat[i,1:4])
  x.train.loc = x.train[-i,]
  y.train.loc = y.train[-i]
  nn = k.nearest.neighbor(x, x.train.loc, y.train.loc, k)
  if(y.train[i] != nn) err_nn = err_nn + 1
}
err_nn = err_nn / nrow(dat)
cat("3-nn: The misclassification error using leave-one-out cross-validation is", err_nn, ".\n")
```

```
## 3-nn: The misclassification error using leave-one-out cross-validation is 0.04 .
```

Here are the result for 5-fold cross-validation.

```
# here for five-fold cross-validation for 3-nn
k = 3
err_nn = 0
n = nrow(dat)
# s determines a random order, and we use batches of n/5
s = sample(150, 150)
for(j in 1:5) {
  rows.for.test = s[(j-1)*n/5 + 1:(n/5)]
  x.train.loc = x.train[-rows.for.test,]
  y.train.loc = y.train[-rows.for.test]
  for(i in rows.for.test) {
    x = x.train[i,]
    y = y.train[i]
    nn = k.nearest.neighbor(x, x.train.loc, y.train.loc, k)
    if(y != nn) err_nn = err_nn + 1
```

```
  }
}
err_nn = err_nn / n
cat("3-nn: The misclassification error using 5-fold cross-validation is", err_nn, ".\n")
```

```
## 3-nn: The misclassification error using 5-fold cross-validation is 0.04 .
```

Using naive Bayes for a spam filter

Peter Pfaffelhuber

2022-12-06

The spam dataset

A classical task of machine learning is to classify an email into *spam* and *no spam* using the occurrence of certain words. Here, we use a dataset that comes within the package *kernlab*, which we have to install first.

```
# Only used for initial running of the script.
# install.packages("kernlab")
```

You can now load the dataset into R by typing:

```
library(kernlab)
data(spam)
dat = spam
```

Again, we look at a summary and the first few lines.

```
summary(dat)
```

```
##       make           address            all             num3d        
##  Min.   :0.0000   Min.   : 0.000   Min.   :0.0000   Min.   : 0.00000  
##  1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000   1st Qu.: 0.00000  
##  Median :0.0000   Median : 0.000   Median :0.0000   Median : 0.00000  
##  Mean   :0.1046   Mean   : 0.213   Mean   :0.2807   Mean   : 0.06542  
##  3rd Qu.:0.0000   3rd Qu.: 0.000   3rd Qu.:0.4200   3rd Qu.: 0.00000  
##  Max.   :4.5400   Max.   :14.280   Max.   :5.1000   Max.   :42.81000  
##       our              over             remove          internet      
##  Min.   : 0.0000   Min.   :0.0000   Min.   :0.0000   Min.   : 0.0000  
##  1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.0000  
##  Median : 0.0000   Median :0.0000   Median :0.0000   Median : 0.0000  
##  Mean   : 0.3122   Mean   :0.0959   Mean   :0.1142   Mean   : 0.1053  
##  3rd Qu.: 0.3800   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.: 0.0000  
##  Max.   :10.0000   Max.   :5.8800   Max.   :7.2700   Max.   :11.1100  
##      order             mail            receive            will        
##  Min.   :0.00000   Min.   : 0.0000   Min.   :0.00000   Min.   :0.0000  
##  1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.00000   1st Qu.:0.0000  
##  Median :0.00000   Median : 0.0000   Median :0.00000   Median :0.1000  
##  Mean   :0.09007   Mean   : 0.2394   Mean   :0.05982   Mean   :0.5417  
##  3rd Qu.:0.00000   3rd Qu.: 0.1600   3rd Qu.:0.00000   3rd Qu.:0.8000  
##  Max.   :5.26000   Max.   :18.1800   Max.   :2.61000   Max.   :9.6700  
##      people           report          addresses           free        
##  Min.   :0.00000   Min.   : 0.00000   Min.   :0.0000   Min.   : 0.0000  
##  1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.: 0.0000  
##  Median :0.00000   Median : 0.00000   Median :0.0000   Median : 0.0000  
##  Mean   :0.09393   Mean   : 0.05863   Mean   :0.0492   Mean   : 0.2488  
##  3rd Qu.:0.00000   3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.: 0.1000  
```

```
##  Max.   :5.55000   Max.   :10.00000   Max.   :4.4100   Max.   :20.0000
##     business          email             you              credit
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   : 0.00000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.: 0.00000
##  Median :0.0000   Median :0.0000   Median : 1.310   Median : 0.00000
##  Mean   :0.1426   Mean   :0.1847   Mean   : 1.662   Mean   : 0.08558
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.: 2.640   3rd Qu.: 0.00000
##  Max.   :7.1400   Max.   :9.0900   Max.   :18.750   Max.   :18.18000
##       your              font             num000           money
##  Min.   : 0.0000   Min.   : 0.0000   Min.   :0.0000   Min.   : 0.00000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.: 0.00000
##  Median : 0.2200   Median : 0.0000   Median :0.0000   Median : 0.00000
##  Mean   : 0.8098   Mean   : 0.1212   Mean   :0.1016   Mean   : 0.09427
##  3rd Qu.: 1.2700   3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.: 0.00000
##  Max.   :11.1100   Max.   :17.1000   Max.   :5.4500   Max.   :12.50000
##        hp               hpl              george           num650
##  Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000   Min.   :0.0000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0000
##  Median : 0.0000   Median : 0.0000   Median : 0.0000   Median :0.0000
##  Mean   : 0.5495   Mean   : 0.2654   Mean   : 0.7673   Mean   :0.1248
##  3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.:0.0000
##  Max.   :20.8300   Max.   :16.6600   Max.   :33.3300   Max.   :9.0900
##       lab               labs             telnet            num857
##  Min.   : 0.00000   Min.   :0.0000   Min.   : 0.00000   Min.   :0.00000
##  1st Qu.: 0.00000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.:0.00000
##  Median : 0.00000   Median :0.0000   Median : 0.00000   Median :0.00000
##  Mean   : 0.09892   Mean   :0.1029   Mean   : 0.06475   Mean   :0.04705
##  3rd Qu.: 0.00000   3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.:0.00000
##  Max.   :14.28000   Max.   :5.8800   Max.   :12.50000   Max.   :4.76000
##       data              num415            num85          technology
##  Min.   : 0.00000   Min.   :0.00000   Min.   : 0.0000   Min.   :0.00000
##  1st Qu.: 0.00000   1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.00000
##  Median : 0.00000   Median :0.00000   Median : 0.0000   Median :0.00000
##  Mean   : 0.09723   Mean   :0.04784   Mean   : 0.1054   Mean   :0.09748
##  3rd Qu.: 0.00000   3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.:0.00000
##  Max.   :18.18000   Max.   :4.76000   Max.   :20.0000   Max.   :7.69000
##     num1999          parts              pm               direct
##  Min.   :0.000   Min.   :0.0000   Min.   : 0.00000   Min.   :0.00000
##  1st Qu.:0.000   1st Qu.:0.0000   1st Qu.: 0.00000   1st Qu.:0.00000
##  Median :0.000   Median :0.0000   Median : 0.00000   Median :0.00000
##  Mean   :0.137   Mean   :0.0132   Mean   : 0.07863   Mean   :0.06483
##  3rd Qu.:0.000   3rd Qu.:0.0000   3rd Qu.: 0.00000   3rd Qu.:0.00000
##  Max.   :6.890   Max.   :8.3300   Max.   :11.11000   Max.   :4.76000
##        cs             meeting          original         project
##  Min.   :0.00000   Min.   : 0.0000   Min.   :0.0000   Min.   : 0.0000
##  1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.: 0.0000
##  Median :0.00000   Median : 0.0000   Median :0.0000   Median : 0.0000
##  Mean   :0.04367   Mean   : 0.1323   Mean   :0.0461   Mean   : 0.0792
##  3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.:0.0000   3rd Qu.: 0.0000
##  Max.   :7.14000   Max.   :14.2800   Max.   :3.5700   Max.   :20.0000
##        re               edu              table           conference
##  Min.   : 0.0000   Min.   : 0.0000   Min.   :0.000000   Min.   : 0.00000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.000000   1st Qu.: 0.00000
##  Median : 0.0000   Median : 0.0000   Median :0.000000   Median : 0.00000
```

```
##  Mean   : 0.3012   Mean   : 0.1798   Mean   :0.005444   Mean   : 0.03187
##  3rd Qu.: 0.1100   3rd Qu.: 0.0000   3rd Qu.:0.000000   3rd Qu.: 0.00000
##  Max.   :21.4200   Max.   :22.0500   Max.   :2.170000   Max.   :10.00000
##  charSemicolon     charRoundbracket charSquarebracket charExclamation
##  Min.   :0.00000   Min.   :0.000    Min.   :0.00000   Min.   : 0.0000
##  1st Qu.:0.00000   1st Qu.:0.000    1st Qu.:0.00000   1st Qu.: 0.0000
##  Median :0.00000   Median :0.065    Median :0.00000   Median : 0.0000
##  Mean   :0.03857   Mean   :0.139    Mean   :0.01698   Mean   : 0.2691
##  3rd Qu.:0.00000   3rd Qu.:0.188    3rd Qu.:0.00000   3rd Qu.: 0.3150
##  Max.   :4.38500   Max.   :9.752    Max.   :4.08100   Max.   :32.4780
##    charDollar        charHash          capitalAve        capitalLong
##  Min.   :0.00000   Min.   : 0.00000   Min.   :   1.000   Min.   :   1.00
##  1st Qu.:0.00000   1st Qu.: 0.00000   1st Qu.:   1.588   1st Qu.:   6.00
##  Median :0.00000   Median : 0.00000   Median :   2.276   Median :  15.00
##  Mean   :0.07581   Mean   : 0.04424   Mean   :   5.191   Mean   :  52.17
##  3rd Qu.:0.05200   3rd Qu.: 0.00000   3rd Qu.:   3.706   3rd Qu.:  43.00
##  Max.   :6.00300   Max.   :19.82900   Max.   :1102.500   Max.   :9989.00
##   capitalTotal          type
##  Min.   :    1.0   nonspam:2788
##  1st Qu.:   35.0   spam   :1813
##  Median :   95.0
##  Mean   :  283.3
##  3rd Qu.:  266.0
##  Max.   :15841.0
```

```
# Showing the first few rows
head(dat)
```

```
##   make address  all num3d  our over remove internet order mail receive will
## 1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00  0.00 0.00    0.00 0.64
## 2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07  0.00 0.94    0.21 0.79
## 3 0.06    0.00 0.71     0 1.23 0.19   0.19     0.12  0.64 0.25    0.38 0.45
## 4 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.31
## 5 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.31
## 6 0.00    0.00 0.00     0 1.85 0.00   0.00     1.85  0.00 0.00    0.00 0.00
##   people report addresses free business email  you credit your font num000
## 1   0.00   0.00      0.00 0.32     0.00  1.29 1.93   0.00 0.96    0   0.00
## 2   0.65   0.21      0.14 0.14     0.07  0.28 3.47   0.00 1.59    0   0.43
## 3   0.12   0.00      1.75 0.06     0.06  1.03 1.36   0.32 0.51    0   1.16
## 4   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0   0.00
## 5   0.31   0.00      0.00 0.31     0.00  0.00 3.18   0.00 0.31    0   0.00
## 6   0.00   0.00      0.00 0.00     0.00  0.00 0.00   0.00 0.00    0   0.00
##   money hp hpl george num650 lab labs telnet num857 data num415 num85
## 1  0.00  0   0      0      0   0    0      0      0    0      0     0
## 2  0.43  0   0      0      0   0    0      0      0    0      0     0
## 3  0.06  0   0      0      0   0    0      0      0    0      0     0
## 4  0.00  0   0      0      0   0    0      0      0    0      0     0
## 5  0.00  0   0      0      0   0    0      0      0    0      0     0
## 6  0.00  0   0      0      0   0    0      0      0    0      0     0
##   technology num1999 parts pm direct cs meeting original project   re  edu
## 1          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 2          0    0.07     0  0   0.00  0       0     0.00       0 0.00 0.00
## 3          0    0.00     0  0   0.06  0       0     0.12       0 0.06 0.06
## 4          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
## 5          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
```

```
## 6          0    0.00     0  0   0.00  0       0     0.00       0 0.00 0.00
##   table conference charSemicolon charRoundbracket charSquarebracket
## 1     0          0          0.00            0.000                 0
## 2     0          0          0.00            0.132                 0
## 3     0          0          0.01            0.143                 0
## 4     0          0          0.00            0.137                 0
## 5     0          0          0.00            0.135                 0
## 6     0          0          0.00            0.223                 0
##   charExclamation charDollar charHash capitalAve capitalLong capitalTotal type
## 1           0.778      0.000    0.000      3.756          61          278 spam
## 2           0.372      0.180    0.048      5.114         101         1028 spam
## 3           0.276      0.184    0.010      9.821         485         2259 spam
## 4           0.137      0.000    0.000      3.537          40          191 spam
## 5           0.135      0.000    0.000      3.537          40          191 spam
## 6           0.000      0.000    0.000      3.000          15           54 spam
```

```
# The dimensions of the dataframe
dim(dat)
```

```
## [1] 4601   58
```

```
# Column names
colnames(dat)
```

```
##  [1] "make"              "address"           "all"
##  [4] "num3d"             "our"               "over"
##  [7] "remove"            "internet"          "order"
## [10] "mail"              "receive"           "will"
## [13] "people"            "report"            "addresses"
## [16] "free"              "business"          "email"
## [19] "you"               "credit"            "your"
## [22] "font"              "num000"            "money"
## [25] "hp"                "hpl"               "george"
## [28] "num650"            "lab"               "labs"
## [31] "telnet"            "num857"            "data"
## [34] "num415"            "num85"             "technology"
## [37] "num1999"           "parts"             "pm"
## [40] "direct"            "cs"                "meeting"
## [43] "original"          "project"           "re"
## [46] "edu"               "table"             "conference"
## [49] "charSemicolon"     "charRoundbracket"  "charSquarebracket"
## [52] "charExclamation"   "charDollar"        "charHash"
## [55] "capitalAve"        "capitalLong"       "capitalTotal"
## [58] "type"
```

```
# type of datastructure
class(dat)
```

```
## [1] "data.frame"
```

Here, we will only use the first 48 columns, since they tell us if certain words appear in an email or not. More exactly, spam[i,j] gives a frequency by which word j occurs in email i. We display the words here, as well as the class spam or nonspam together with its frequencies. Note that we translate the frequencies into pure *occurrrence* or *non-occurrence* of a word.

```
x = as.matrix((spam[, 1:48]>0))
y = as.vector(spam[,58])
n = nrow(x)
```

```
# the words
colnames(x)
```

```
##  [1] "make"       "address"    "all"        "num3d"      "our"
##  [6] "over"       "remove"     "internet"   "order"      "mail"
## [11] "receive"    "will"       "people"     "report"     "addresses"
## [16] "free"       "business"   "email"      "you"        "credit"
## [21] "your"       "font"       "num000"     "money"      "hp"
## [26] "hpl"        "george"     "num650"     "lab"        "labs"
## [31] "telnet"     "num857"     "data"       "num415"     "num85"
## [36] "technology" "num1999"    "parts"      "pm"         "direct"
## [41] "cs"         "meeting"    "original"   "project"    "re"
## [46] "edu"        "table"      "conference"
```

```
# the class frequencies
table(y)
```

```
## y
## nonspam    spam
##    2788    1813
```

Next, we split out dataset into training and testset, the latter we specify to be of size 1000.

```
# split into train and test; test has size 1000
test_size=1000
test_index = sample(1:n, test_size)
x.test = x[test_index,]
y.test = y[test_index]
x.train= x[-test_index,]
y.train = y[-test_index]
```

We will need to compute number of occurrence of all words in both classes. Note that getcounts returns a 3d-array, where the last dimension has T or F, depending on a word occurring or not occurring.

```
getcounts<-function(x.train, y.train) {
  classes = unique(y.train)
  res = array(0, dim = c(length(classes), ncol(x.train), 2), dimnames = list(classes, colnames(x.train)
  for(i in 1:nrow(x.train)) {
    res[y.train[i],,1] = res[y.train[i],,1] + x.train[i,]
    res[y.train[i],,2] = res[y.train[i],,2] + 1-x.train[i,]
  }
  res
}
counts=getcounts(x.train, y.train)
counts
```

```
## , , T
##
##         make address all num3d our over remove internet order mail receive will
## spam     496     493 852    29 872  517    600      474   416  643     432  882
## nonspam  333     216 627     6 492  260     36      163   180  371     114  930
##         people report addresses free business email  you credit your font
## spam       406    179       226  772      540   522 1249    287 1141   71
## nonspam    273    103        40  198      209   287 1264     33  759   17
##         num000 money  hp hpl george num650 lab labs telnet num857 data num415
## spam       449   533  36  21      6     21   8   10      2      2   47      8
```

```
## nonspam     62    43 807 595    598    330 272  337    220    152  280    152
##         num85 technology num1999 parts  pm direct  cs meeting original project
## spam       37         86      70    23  46    145   1      15       63      37
## nonspam   331        370     569    38 249    182 107     248      223     222
##          re edu table conference
## spam    379  49    12         13
## nonspam 617 346    35        142
##
## , , F
##
##         make address  all num3d  our over remove internet order mail receive
## spam     907     910  551  1374  531  886    803      929   987  760     971
## nonspam 1865    1982 1571  2192 1706 1938   2162     2035  2018 1827    2084
##         will people report addresses free business email you credit your font
## spam     521    997   1224      1177  631      863   881 154   1116  262 1332
## nonspam 1268   1925   2095      2158 2000     1989  1911 934   2165 1439 2181
##         num000 money   hp  hpl george num650  lab labs telnet num857 data
## spam       954   870 1367 1382   1397   1382 1395 1393   1401   1401 1356
## nonspam   2136  2155 1391 1603   1600   1868 1926 1861   1978   2046 1918
##         num415 num85 technology num1999 parts   pm direct   cs meeting original
## spam      1395  1366       1317    1333  1380 1357   1258 1402    1388     1340
## nonspam   2046  1867       1828    1629  2160 1949   2016 2091    1950     1975
##         project   re  edu table conference
## spam       1366 1024 1354  1391       1390
## nonspam    1976 1581 1852  2163       2056
```

Let us use the delta-function which needs to be optimized for the naive Bayes classifier. We also define the naive Bayes classifier.

```
# counts is a classes x features matrix.
# x.test is a single vector
delta<-function(x.test, counts){
  classes = rownames(counts)
  res = matrix(0, ncol = ncol(counts), nrow = length(classes), dimnames = list(classes, colnames(counts
  for(class in classes) {
    res[class,] = x.test * counts[class,,1] + (1-x.test) * counts[class,,2]
  }
#  cat("\n", rowSums(log(res)))
  rowSums(log(1+res) - log(2 + counts[,1,1] + counts[,1,2]))
}

naive_Bayes_classifier<-function(x.test, counts) {
  del = delta(x.test, counts)
  names(which.max(del))
}
```

We are now ready to compute the training and test error.

```
# Compute the training error
tr_error = 0
y.train.inferred = y.train
for(i in 1:nrow(x.train)) {
  y.train.inferred[i] = naive_Bayes_classifier(x.train[i,], counts)
  if(y.train[i] != y.train.inferred[i])
    tr_error = tr_error +1
}
```

```
cat("Training error: ", tr_error/(n-test_size), "\n")
```

```
## Training error:  0.1216329
```

```
# Compute the test error
te_error = 0
y.test.inferred = y.test
for(i in 1:nrow(x.test)) {
  y.test.inferred[i] = naive_Bayes_classifier(x.test[i,], counts)
  if(y.test[i] != y.test.inferred[i])
    te_error = te_error +1
}
cat("Test error: ", te_error/test_size, "\n")
```

```
## Test error:  0.126
```

Let us have a closer look. We want to specify which class gives the larger error.

```
# More detailed training error
tr_error_spam = tr_error_nonspam = 0
y.train.inferred = y.train
for(i in 1:nrow(x.train)) {
  y.train.inferred[i] = naive_Bayes_classifier(x.train[i,], counts)
  if(y.train[i] == "spam" & y.train.inferred[i] == "nonspam")
    tr_error_spam = tr_error_spam +1
  if(y.train[i] == "nonspam" & y.train.inferred[i] == "spam")
    tr_error_nonspam = tr_error_nonspam +1
}
cat("Training error for spam: ", tr_error_spam/sum(y.train == "spam"), "\n")
```

```
## Training error for spam:  0.1824661
```

```
cat("Training error for nonspam: ", tr_error_nonspam/sum(y.train == "nonspam"), "\n")
```

```
## Training error for nonspam:  0.08280255
```

```
# More detailed test error
te_error_spam = te_error_nonspam = 0
y.test.inferred = y.test
for(i in 1:nrow(x.test)) {
  y.test.inferred[i] = naive_Bayes_classifier(x.test[i,], counts)
  if(y.test[i] == "spam" & y.test.inferred[i] == "nonspam")
    te_error_spam = te_error_spam +1
  if(y.test[i] == "nonspam" & y.test.inferred[i] == "spam")
    te_error_nonspam = te_error_nonspam +1
}
cat("Test error for spam: ", te_error_spam/sum(y.test == "spam"), "\n")
```

```
## Test error for spam:  0.204878
```

```
cat("Test error for nonspam: ", te_error_nonspam/sum(y.test == "nonspam"), "\n")
```

```
## Test error for nonspam:  0.07118644
```

‘‘‘

A feed-forward neural network for classification in the Iris dataset

Peter Pfaffelhuber

2023-01-08

The iris dataset

We will again use the Iris dataset, which we also used for the nearest-neighbor classifier. Again, our goal is to construct a classifier, based on Sepal and Petal length and width, but now using a feed-forward neural netword with one hidden layer and a sigmoid activation function. Recall that you can load the dataset into R and showing the first few rows by:

```
dat = iris
head(dat)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
nsam=nrow(dat)
```

The neural network is going to approximate some function from $\mathbb{R}^4 \to [0,1]^3$. We are using the convention that setosa $\equiv (1,0,0)$, versicolor $\equiv (0,1,0)$, virginica $\equiv (0,0,1)$.

```
  trans=diag(3)
  rownames(trans) = unique(dat[,5])
  trans
```

```
##            [,1] [,2] [,3]
## setosa        1    0    0
## versicolor    0    1    0
## virginica     0    0    1
```

```
  dat2 = cbind(dat[,1]-mean(dat[,1]), dat[,2]-mean(dat[,2]), dat[,3]-mean(dat[,3]),
               dat[,4]-mean(dat[,4]), trans[dat[,5],])
```

For the neural network, we need to consider some properties of the neural network. We need to fix the number of nodes in the hidden layer, and the activation function. We are using a sigmoidal function here. For the loss function, we use $\ell(z,y) = \sum_{i=1}^3 (\rho(z_{2i}) - y_i)^2$, where y is the true outcome $(1,0,0), (0,1,0)$ or $(0,0,1)$ and $\rho(z_2)$ is the estimated outcome from the neural network.

In the code chunk below, we also implement all details of Theorem 4.17 and Remark 4.18. Here, we only deal with a single dataset. Later, we will use the backpropagate function for learning the parameters based on all datasets.

```
# Sigmoidal activation function
# This implementation also works for vectors x.
rho <- function(z) {
```

```
   1/(1+exp(-z))
}
# Derivative for back-propagation
rho_prime<-function(z){
  exp(-z)/(1+exp(-z))^2
}

# Note that we will evaluate rho on the output layer in order to obtain outputs in [0,1]
evaluate_forward_nn<-function(x0, beta1, beta2, gamma1, gamma2){
  z1 = beta1 %*% x0 + gamma1
  x1 = rho(z1)
  z2 = beta2 %*% x1 + gamma2
  list(x0=x0, z1=z1, x1=x1, z2=z2)
}

# We concatenate squared loss with rho, in order to get the loss function; see Remark 4.18.
ell <- function(z2, y) {
  sum((rho(z2)-y)^2)
}

nabla_ell<-function(z2, y) {
  matrix(2*(rho(z2)-y) %*% diag(rho_prime(z2)), ncol = length(z2))
}

# Computing derivatives wrt beta1, gamma1, beta2, gamma2 by back-propagation
# for a single dataset
backpropagate<-function(forward_list, y, beta1, beta2, gamma1, gamma2){
  delta2 = nabla_ell(forward_list$z2, y)
  delta1 = delta2 %*% beta2 %*% diag(as.vector(rho_prime(forward_list$z1)))
  list(nabla_beta1 = t(delta1) %*% t(forward_list$x0),
       nabla_beta2 = t(delta2) %*% t(forward_list$x1),
       nabla_gamma1 = delta1,
       nabla_gamma2 = delta2)
}
```

We now give some functions which iterate over all datasets.

```
# Use the gradient of parameters when using all datasets
nabla_parms_alldatasets<-function(dat2, beta1, beta2, gamma1, gamma2){
  nabla_beta1 = 0 * beta1
  nabla_beta2 = 0 * beta2
  nabla_gamma1 = 0 * t(gamma1)
  nabla_gamma2 = 0 * t(gamma2)

  for(i in 1:nsam) {
    x0 = as.numeric(matrix(dat2[i,1:4], nrow=4, ncol=1))
    y = as.numeric(dat2[i,5:7])
    forward_list = evaluate_forward_nn(x0, beta1, beta2, gamma1, gamma2)
    nabla_parms = backpropagate(forward_list, y, beta1, beta2, gamma1, gamma2)
    nabla_beta1 = nabla_beta1 + nabla_parms$nabla_beta1
    nabla_beta2 = nabla_beta2 + nabla_parms$nabla_beta2
    nabla_gamma1 = nabla_gamma1 + nabla_parms$nabla_gamma1
    nabla_gamma2 = nabla_gamma2 + nabla_parms$nabla_gamma2
  }
```

```
  nabla_beta1 = nabla_beta1 / nsam
  nabla_beta2 = nabla_beta2 / nsam
  nabla_gamma1 = nabla_gamma1 / nsam
  nabla_gamma2 = nabla_gamma2 / nsam
  list(nabla_beta1 = nabla_beta1,
       nabla_beta2 = nabla_beta2,
       nabla_gamma1 = nabla_gamma1,
       nabla_gamma2 = nabla_gamma2)
}

compute_loss_alldatasets<-function(dat2, beta1, beta2, gamma1, gamma2){
  nsam = nrow(dat2)
  loss = 0
  for(i in 1:nsam){
    x0 = as.numeric(matrix(dat2[i,1:4], nrow=4, ncol=1))
    y = as.numeric(dat2[i,5:7])
    forward_list = evaluate_forward_nn(x0, beta1, beta2, gamma1, gamma2)
    loss = loss +   ell(as.vector(forward_list$z2), y)
  }
  loss/nsam
}
```

Now, let us start learning using deterministic gradient descent with a learning rate which needs to be determined.

```
# Number of nodes in the input layer (4)
j0 = ncol(dat)-1
# Number of nodes in the hidden layer, may be chosen by user
j1 = 5
# Number of nodes in the output layer (3)
j2 = length(unique(dat[,5]))

# Learning rate
learning_rate = 20
# abort update if change is less than
iterations = 5000

# Initialize the weights and biases
set.seed(1)
beta1 = matrix(runif(j0*j1, min=-.1, max=.1), ncol = j0, nrow = j1)
beta2 = matrix(runif(j1*j2, min=-.1, max=.1), ncol = j1, nrow = j2)
gamma1 = matrix(runif(j1, min=-.1, max=.1), nrow=j1, ncol=1)
gamma2 = matrix(runif(j2, min=-.1, max=.1), nrow=j2, ncol=1)

for(i in 1:iterations) {
  a = nabla_parms_alldatasets(dat2, beta1, beta2, gamma1, gamma2)
  beta1 = beta1 - learning_rate * a$nabla_beta1
  beta2 = beta2 - learning_rate * a$nabla_beta2
  gamma1 = gamma1 - learning_rate * t(a$nabla_gamma1)
  gamma2 = gamma2 - learning_rate * t(a$nabla_gamma2)
  if( i/100 == round(i/100)) {
    cat("Iteration", i, "loss : ")
    cat(compute_loss_alldatasets(dat2, beta1, beta2, gamma1, gamma2), "\n")
  }
```

```
}
```

```
## Iteration 100 loss : 0.4552601
## Iteration 200 loss : 0.4216868
## Iteration 300 loss : 0.3938416
## Iteration 400 loss : 0.3639856
## Iteration 500 loss : 0.3335787
## Iteration 600 loss : 0.3041956
## Iteration 700 loss : 0.2767959
## Iteration 800 loss : 0.2518362
## Iteration 900 loss : 0.2294451
## Iteration 1000 loss : 0.2095605
## Iteration 1100 loss : 0.1920186
## Iteration 1200 loss : 0.1766066
## Iteration 1300 loss : 0.1630945
## Iteration 1400 loss : 0.1512542
## Iteration 1500 loss : 0.1408706
## Iteration 1600 loss : 0.1317484
## Iteration 1700 loss : 0.1237141
## Iteration 1800 loss : 0.1166164
## Iteration 1900 loss : 0.1103248
## Iteration 2000 loss : 0.1047281
## Iteration 2100 loss : 0.09973116
## Iteration 2200 loss : 0.09525354
## Iteration 2300 loss : 0.0912268
## Iteration 2400 loss : 0.08759282
## Iteration 2500 loss : 0.08430214
## Iteration 2600 loss : 0.08131255
## Iteration 2700 loss : 0.07858794
## Iteration 2800 loss : 0.07609735
## Iteration 2900 loss : 0.07381411
## Iteration 3000 loss : 0.07171518
## Iteration 3100 loss : 0.06978062
## Iteration 3200 loss : 0.06799309
## Iteration 3300 loss : 0.06633746
## Iteration 3400 loss : 0.0648005
## Iteration 3500 loss : 0.06337061
## Iteration 3600 loss : 0.06203757
## Iteration 3700 loss : 0.06079236
## Iteration 3800 loss : 0.05962699
## Iteration 3900 loss : 0.05853437
## Iteration 4000 loss : 0.0575082
## Iteration 4100 loss : 0.05654286
## Iteration 4200 loss : 0.05563329
## Iteration 4300 loss : 0.05477499
## Iteration 4400 loss : 0.0539639
## Iteration 4500 loss : 0.05319636
## Iteration 4600 loss : 0.05246907
## Iteration 4700 loss : 0.05177904
## Iteration 4800 loss : 0.05112357
## Iteration 4900 loss : 0.05050019
## Iteration 5000 loss : 0.04990666
```

Now, let us see how well the neural network classified the Iris data.

```r
# For predicting a class, we use the largest component of rho(z2)
prediction = NULL
for(i in 1:nsam) {
  x0 = as.numeric(matrix(dat2[i,1:4], nrow=4, ncol=1))
  prediction = rbind(prediction,
              as.vector(rho(evaluate_forward_nn(x0, beta1, beta2, gamma1, gamma2)$z2)))
}

class = NULL
truth = NULL
for(i in 1:nsam) {
  class = c(class, which.max(prediction[i,]))
  truth = c(truth, which.max(dat2[i, 5:7]))
}

cat("A total of", sum(class!=truth), "out of", nsam, "plants have been misclassified.")
```

```
## A total of 4 out of 150 plants have been misclassified.
```

A simple example for reinforcement learning

Peter Pfaffelhuber

2023-01-18

The model

We are considering the example of the moving robot (from Figure~5.1) from the manuscript. We have 15 states and 4 actions as follows, which are also seen in Figure~5.2:

```
# There are 15 states
S = 1:15
# There are 4 possible actions, the fifth is to stay where you are
A = 1:5
# This is the reward of moving into the  target state 1
r = matrix(0, nrow=15, ncol=5)
r[2,4] = r[5,1] = r[12, 3] = r[15, 2] = 1
# For each state s and each action a, this gives the resulting state
s=1
next_state<-function(a) {
  res = NULL
  res = rbind(res, c(NA, NA, NA, NA, 1)) # row 1
  res = rbind(res, c(NA, 3, 6, 1, 2))    # row 2
  res = rbind(res, c(NA, 4, 7, 2, 3))    # row 3
  res = rbind(res, c(NA, NA, 8, 3, 4))   # row 4
  res = rbind(res, c(1, 6, 9, NA, 5))    # row 5
  res = rbind(res, c(2, 7, 10, 5, 6))    # row 6
  res = rbind(res, c(3, 8, 11, 6, 7))    # row 7
  res = rbind(res, c(4, NA, 12, 7, 8))   # row 8
  res = rbind(res, c(5, 10, 13, NA, 9))  # row 9
  res = rbind(res, c(6, 11, 14, 9, 10))  # row 10
  res = rbind(res, c(7, 12, 15, 10, 11)) # row 11
  res = rbind(res, c(8, NA, 1, 11, 12))  # row 12
  res = rbind(res, c(9, 14, NA, NA, 13)) # row 13
  res = rbind(res, c(10, 15, NA, 13, 14))# row 14
  res = rbind(res, c(11, 1, NA, 14, 15)) # row 15
  res[,a]
}

Psasp<-function(a) {
  sp_vector = next_state(a)
  sp_matrix = NULL
  for(s in 1:15) {
    sp_matrix = rbind(sp_matrix, (1:15) == sp_vector[s])
  }
  sp_matrix[is.na(sp_matrix)]<-0
  sp_matrix + 0
}
```

Policy evaluation

For policy evaluation, we have to introduce a policy $\pi = (\pi_s(a))_{s=1,\ldots,15, a=1,\ldots 4}$, where $\pi_s(a)$ is the probability to take action a when in state s.

```
# This policy goes into a random direction
pi = matrix(c(0,  0,  0,  0,  1, # state 1
              0,1/3,1/3,1/3,  0, # state 2
              0,1/3,1/3,1/3,  0, # state 3
              0,  0,1/2,1/2,  0, # state 4
            1/3,1/3,1/3,  0,  0, # state 5
            1/4,1/4,1/4,1/4,  0, # state 6
            1/4,1/4,1/4,1/4,  0, # state 7
            1/3,  0,1/3,1/3,  0, # state 8
            1/3,1/3,1/3,  0,  0, # state 9
            1/4,1/4,1/4,1/4,  0, # state 10
            1/4,1/4,1/4,1/4,  0, # state 11
            1/3,  0,1/3,1/3,  0, # state 12
            1/2,1/2,  0,  0,  0, # state 13
            1/3,1/3,  0,1/3,  0, # state 14
            1/3,1/3,  0,1/3,  0 # state 15
), nrow=15, byrow=TRUE)
```

Now we can compute the value of the above policy using matrix inversion.

```
gamma = 0.9
# We need to solve
# v = sum_a pi[,] * r[,a] + (gamma * sum_a Psasp(a)) %*% v)
# We transform this to A%*%v = b with
A = diag(rep(1,15))
b = 0
for(a in 1:4) {
  b = b + pi[,a] * r[,a]
  A = A - gamma * pi[,a] * Psasp(a)
}
vast = solve(A)%*%b

print(vast)
```

```
##            [,1]
##  [1,] 0.0000000
##  [2,] 0.5763493
##  [3,] 0.3866602
##  [4,] 0.3479942
##  [5,] 0.5763493
##  [6,] 0.4233929
##  [7,] 0.3645239
##  [8,] 0.3866602
##  [9,] 0.3866602
## [10,] 0.3645239
## [11,] 0.4233929
## [12,] 0.5763493
## [13,] 0.3479942
## [14,] 0.3866602
## [15,] 0.5763493
```

Now we compute the same iteratively.

```
# Compute the function f from (5.4)
f<-function(v) {
  res = 0 * v
  for(a in 1:5) {
    res = res + pi[,a]*(r[,a] + gamma * Psasp(a) %*% v)
  }
  res
}

# f must be of the form f(v0,...)
iterate<-function(f, v0, eps, ...){
  cont = TRUE
  vold = v0
  v = v0 + 2*eps
  # Now comes the iteration
  while(TRUE) {
    if(sum(abs(v-vold))<eps) {
      cont = FALSE
    }
    if(!cont) break
    vold = v
    v = f(vold,...)
  }
  v
}

v0 = 1 * (1:15)
eps=1e-05
v = iterate(f, v0, eps)
print(v)
```

```
##                [,1]
##  [1,] 5.468866e-06
##  [2,] 5.763548e-01
##  [3,] 3.866657e-01
##  [4,] 3.479997e-01
##  [5,] 5.763548e-01
##  [6,] 4.233984e-01
##  [7,] 3.645294e-01
##  [8,] 3.866657e-01
##  [9,] 3.866657e-01
## [10,] 3.645294e-01
## [11,] 4.233984e-01
## [12,] 5.763548e-01
## [13,] 3.479997e-01
## [14,] 3.866657e-01
## [15,] 5.763548e-01
```

Policy optimization

We now use the Bellman optimality equation for obtaining a greedy optimal policy.‘

```
# Compute the function f from (5.6)
h<-function(v) {
  res = 0 *r
  for(a in 1:5) {
    res[,a] = (r[,a] + gamma * Psasp(a) %*% v)
  }
  apply(res, 1, max)
}

v0 = 1 * (1:15)
eps=1e-5
v = iterate(h, v0, eps)
cat("Optimal state values: ", round(v, digits=3))
```

```
## Optimal state values:  0 1 0.9 0.81 1 0.9 0.81 0.9 0.9 0.81 0.9 1 0.81 0.9 1
```

```
# Compute g and output an optimal policy
g = 0 *r
for(a in 1:5) {
  g[,a] = (r[,a] + gamma * Psasp(a) %*% v)
}
pi_optimal = 0*g
for(s in 1:15){
  pi_optimal[s,which.max(g[s,])] = 1
}
cat("Optimal policy: ")
```

```
## Optimal policy:
```

```
pi_optimal
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    0    0    0    0    1
##  [2,]    0    0    0    1    0
##  [3,]    0    0    0    1    0
##  [4,]    0    0    1    0    0
##  [5,]    1    0    0    0    0
##  [6,]    1    0    0    0    0
##  [7,]    1    0    0    0    0
##  [8,]    0    0    1    0    0
##  [9,]    1    0    0    0    0
## [10,]    1    0    0    0    0
## [11,]    0    1    0    0    0
## [12,]    0    0    1    0    0
## [13,]    1    0    0    0    0
## [14,]    0    1    0    0    0
## [15,]    0    1    0    0    0
```

Simulations for Monte Carlo techniques

Let us simulate several paths of the policy (which takes every possible direction with equal probability) above. Afterwards, we estimate v_π using the basic and the sample-efficient Monte Carlo algorithm.

```
# number of paths to simulate
n = 1000
simulations = list()
```

```
for(i in 1:n){
  s = sample(2:15, 1)
  a = sample(1:5, 1, prob = pi[s,])
  path = c(s,a)
  while(s!=1) {
      s = which.max(Psasp(a)[s,])
      a = sample(1:5, 1, prob = pi[s,])
      path = cbind(path, c(s,a))
    }
  simulations[[i]] = path
}

# print the first two paths
simulations[[1]]
```

```
##      path
## [1,]   15 1
## [2,]    2 5
```

```
simulations[[2]]
```

```
##      path
## [1,]   13 14 15 14 10 11 12 8 4 3 7 3 4 8 7 11 12 1
## [2,]    2  2  4  1  2  2  1 1 4 3 1 2 3 4 3  2  3 5
```

Basic Monte Carlo policy evaluation

See Remark 5.17.

```
# Recall that gamma=0.9 as above
hatg = matrix(0, nrow=15, ncol=5)
count_states = matrix(0, nrow=15, ncol=5)
for(i in 1:n){
  path = simulations[[i]]
  len = length(path[1,])-1
  count_states[path[1,1], path[2,1]] = count_states[path[1,1], path[2,1]] + 1
  rewards = apply(path, 2, function(x) r[x[1], x[2]])
  hatg[path[1,1], path[2,1]] = hatg[path[1,1], path[2,1]] + sum(rewards * gamma^(0:len))
}
# This is the estimate for g_pi...
hatg = hatg / count_states

# ...which we translate into the estimate for v_pi
hatv = 0 * 1:15
for(s in 1:15) hatv[s] = sum(pi[s,] * hatg[s,], na.rm = TRUE)
```

Now we can compare the estimate with the true v^* from above.

```
hatv
```

```
##  [1] 0.0000000 0.5710804 0.3982578 0.2926631 0.5656163 0.4146734 0.4004545
##  [8] 0.4026079 0.4063425 0.3070642 0.3920658 0.5643103 0.3373487 0.4033156
## [15] 0.5979941
```

```
vast[,1]
```

```
##  [1] 0.0000000 0.5763493 0.3866602 0.3479942 0.5763493 0.4233929 0.3645239
```

```
##  [8] 0.3866602 0.3866602 0.3645239 0.4233929 0.5763493 0.3479942 0.3866602
## [15] 0.5763493
```

```
# Difference
sum(abs(hatv - vast[,1]))
```

```
## [1] 0.3129822
```

Sample-efficient Monte Carlo policy evaluation

See Remark 5.18.

```
tildeg = matrix(0, nrow=15, ncol=5)
count_states = matrix(0, nrow=15, ncol=5)
for(i in 1:n){
  path = simulations[[i]]
  rewards = apply(path, 2, function(x) r[x[1], x[2]])
  len = length(path[1,])-1
  for(j in 1:len) {
    count_states[path[1,j], path[2,j]] = count_states[path[1,j], path[2,j]] + 1
    tildeg[path[1,j], path[2,j]] = tildeg[path[1,j], path[2,j]] +
      sum(rewards[j:(len+1)] * gamma^(0:(len-j+1)))
  }
}
# This is the estimate for g_pi...
tildeg = tildeg / count_states

# ...which we translate into the estimate for v_pi
tildev = 0 * 1:15
for(s in 1:15) tildev[s] = sum(pi[s,] * tildeg[s,], na.rm = TRUE)
```

Now we can compare the estimate with the true v^* from above.

```
tildev
```

```
##  [1] 0.0000000 0.5799202 0.3898097 0.3483998 0.5891318 0.4258218 0.3625520
##  [8] 0.3893556 0.3972949 0.3706818 0.4261934 0.5753283 0.3405578 0.3843706
## [15] 0.5721650
```

```
vast[,1]
```

```
##  [1] 0.0000000 0.5763493 0.3866602 0.3479942 0.5763493 0.4233929 0.3645239
##  [8] 0.3866602 0.3866602 0.3645239 0.4233929 0.5763493 0.3479942 0.3866602
## [15] 0.5763493
```

```
# Difference
sum(abs(tildev - vast[,1]))
```

```
## [1] 0.06152895
```

Temporal difference learning

Now, we use the simulations from above in order to apply the temporal difference learning algorithms from Theorem 5.22.

```
eta<-function(t) t^(-0.6)
V = 0*vast[,1]
t=0
for(i in 1:n) {
```

```
  path = simulations[[i]]
  #cat(V, "\n")
  for(j in 1:(length(path[1,])-1)) {
    t = t+1
    V[path[1,j]] = V[path[1,j]] - eta(t) *
      (V[path[1,j]] - r[path[1,j], path[2,j]] - gamma * V[path[1,j+1]])
  }
}
V
```

```
##  [1] 0.0000000 0.4814890 0.2617196 0.2104983 0.4826330 0.3122930 0.2492131
##  [8] 0.2690184 0.2707934 0.2579992 0.3325015 0.4894403 0.2279379 0.2825888
## [15] 0.5157670
```

```
sum(abs(tildev - vast[,1]))
```

```
## [1] 0.06152895
```

Q-learning

Now, we use the simulations from above in order to apply the Q-learning algorithm from Theorem 5.24.

```
eta<-function(t) t^(-0.6)
G = 0*g
t=0
for(i in 1:n) {
  path = simulations[[i]]
  #cat(V, "\n")
  for(j in 1:(length(path[1,])-1)) {
    t = t+1
    G[path[1,j], path[2,j]] = G[path[1,j], path[2,j]] - eta(t) *
      (G[path[1,j], path[2,j]] - r[path[1,j], path[2,j]] - gamma * max(G[path[1,j+1], ]))
  }
}
pi_optimal_Q = 0*g
for(s in 1:15){
  pi_optimal_Q[s,which.max(G[s,])] = 1
}
cat("Optimal policy from Q-learning: ")
```

```
## Optimal policy from Q-learning:
```

```
pi_optimal_Q
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]    1    0    0    0    0
##  [2,]    0    0    0    1    0
##  [3,]    0    0    0    1    0
##  [4,]    0    0    1    0    0
##  [5,]    1    0    0    0    0
##  [6,]    1    0    0    0    0
##  [7,]    0    0    1    0    0
##  [8,]    0    0    1    0    0
##  [9,]    1    0    0    0    0
## [10,]    0    0    1    0    0
## [11,]    0    0    1    0    0
```

```
## [12,]    0    0    1    0    0
## [13,]    0    1    0    0    0
## [14,]    0    1    0    0    0
## [15,]    0    1    0    0    0
```

```
pi_optimal - pi_optimal_Q
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]   -1    0    0    0    1
##  [2,]    0    0    0    0    0
##  [3,]    0    0    0    0    0
##  [4,]    0    0    0    0    0
##  [5,]    0    0    0    0    0
##  [6,]    0    0    0    0    0
##  [7,]    1    0   -1    0    0
##  [8,]    0    0    0    0    0
##  [9,]    0    0    0    0    0
## [10,]    1    0   -1    0    0
## [11,]    0    1   -1    0    0
## [12,]    0    0    0    0    0
## [13,]    1   -1    0    0    0
## [14,]    0    0    0    0    0
## [15,]    0    0    0    0    0
```

Using Dvoretzky's Theorem to approximate an expectation

Peter Pfaffelhuber

2023-01-25

Dvoretzky's Theorem

Let us recall the application of Dvoretzky's Theorem which we use here: Let $X_1, X_2, ...$ be real-valued iid random variables with expectation $E[X_1]$. Then, setting

$$W_1 := X_1, \qquad W_{t+1} = W_t - \eta_t(W_t - X_{t+1})$$

for $\eta_1, \eta_2, ...$ satisfying $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$, we find that $W_t \xrightarrow{t\to\infty} E[X_1]$. We want to show this property by using $X_1 \sim U([1:6])$ and some different η_t's.

```
n = 10000
x = sample(1:6, n, replace = TRUE)

# This is an implementation of the recursion.
approxexp<-function(x, eta) {
  n = length(x)
  w = x[1]
  for(i in 2:n) {
    w = w - eta(i)*(w - x[i])
  }
  w
}
```

Now, we can apply this recursion for different η_t's.

```
eta1<-function(t) {
  1/(t+1)
}
cat("This is the mean:", approxexp(x, eta1), "\n")
```

```
## This is the mean: 3.482252
```

```
eta2<-function(t) {
  log(t)/(t+1)
}
cat("This uses eta2:", approxexp(x, eta2), "\n")
```

```
## This uses eta2: 3.514353
```

```
eta3<-function(t) {
  log(t)^2/(t+1)
}
cat("This uses eta3:", approxexp(x, eta3), "\n")
```

```
## This uses eta3: 3.434936
```

```r
eta4<-function(t) {
  1/(t+1)^2
}
cat("This uses eta4 and does not converge:", approxexp(x, eta4), "\n")
```

```
## This uses eta4 and does not converge: 5.496295
```