Machine Learning

Originally developed by Martin Benning
Adapted and modified by Nicola Perra

Last updated on: September 8, 2023

Lecture Notes 2023-24

Contents

Chapter 1

Mathematical preliminaries

In this first chapter we briefly revise basic mathematical preliminaries that we are going to use throughout this module. Those preliminaries span topics ranging from linear algebra over calculus to basic probability and statistics.

1.1 Linear algebra

Throughout this module, we will extensively deal with vectors and matrices. The term *vector* refers to a an object that both has a magnitude and a direction. We will usually write vectors $x \in \mathbb{R}^d$ (or $x \in \mathbb{R}^{d\times 1}$) as

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix},$$

which we generally also refer to as *column-vectors*. Here each individual x_j, for $j \in \{1, \dots, d\}$, is a real-valued scalar, i.e. $x_j \in \mathbb{R}$ for all $j \in \{1, \dots, d\}$. In contrast to a column-vector, we can also consider *row-vectors* $x^\top \in \mathbb{R}^{1\times d}$, i.e.

$$x^\top = \begin{pmatrix} x_1 & x_2 & \dots & x_d \end{pmatrix}.$$

Both are special cases of a *matrix*, which is a rectangular array of scalars. We write a matrix $X \in \mathbb{R}^{d_1\times d_2}$ of size $d_1 \times d_2$ as

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1d_2} \\ x_{21} & x_{22} & \dots & x_{2d_2} \\ \vdots & & \ddots & \vdots \\ x_{d_1 1} & x_{d_1 2} & \dots & x_{d_1 d_2} \end{pmatrix}.$$

The *transpose* $X^\top \in \mathbb{R}^{d_2\times d_1}$ of matrix $X \in \mathbb{R}^{d_1\times d_2}$ is defined by interchanging the rows and columns, i.e.

$$X^\top = \begin{pmatrix} x_{11} & x_{21} & \dots & x_{d_1 1} \\ x_{12} & x_{22} & \dots & x_{d_1 2} \\ \vdots & & \ddots & \vdots \\ x_{1d_2} & x_{2d_2} & \dots & x_{d_1 d_2} \end{pmatrix}.$$

Note that we can immediately conclude $(X^\top)^\top = X$, and that a row-vector is simply the transpose of a column-vector (and vice versa), which in hindsight explains our use of the notation $x^\top$ for the row-vector. Two very important concepts in the context of matrices are the *range* and the *kernel* (or nullspace) of a matrix. The range $\mathrm{ran}(X)$ of a matrix X is the set of all vectors that can be expressed in terms of X, i.e.

$$\mathrm{ran}(X) := \{Xz \,|\, z \in \mathbb{R}^n\} \,.$$

The kernel $\ker(X)$ of a matrix X is the set of all vectors that X maps onto the zero vector, i.e.

$$\ker(X) := \{z \in \mathbb{R}^n \,|\, Xz = 0\} \,.$$

In the following, we want to recall the two main products that are relevant for this module. The most important vector-vector product is the so-called *dot product* or *inner product* of two vectors $x, y \in \mathbb{R}^d$ of identical dimension, defined as

$$\langle x, y \rangle := \sum_{j=1}^{d} x_j y_j \,.$$

Note that we simply multiply the individual coordinates of the vectors x and y and sum over all products. Another common notation for the dot product is $x \cdot y$.

The *matrix-vector product* for a matrix $X \in \mathbb{R}^{d_1 \times d_2}$ and a column-vector $y \in \mathbb{R}^{d_2 \times 1}$ is defined as

$$(Xy)_i := \sum_{j=1}^{d_2} x_{ij} y_j \qquad \text{for all} \quad i \in \{1, \ldots, d_1\} \,.$$

We denote the resulting (column) vector simply as $Xy \in \mathbb{R}^{d_1 \times 1}$. In identical fashion, we can define a *matrix-matrix product* for matrices $X \in \mathbb{R}^{d_1 \times d_2}$ and $Y \in \mathbb{R}^{d_2 \times d_3}$ as

$$(XY)_{ij} := \sum_{l=1}^{d_2} x_{il} y_{lj} \quad \text{for all} \quad i \in \{1, \ldots, d_1\} \text{ and } j \in \{1, \ldots, d_3\} \,.$$

Again, we denote the resulting matrix simply as $XY \in \mathbb{R}^{d_1 \times d_3}$. We want to emphasise that the inner product can be viewed as a special case of the matrix-matrix product, if we consider the matrix-matrix product of a row-vector and a column-vector, i.e. for $x, y \in \mathbb{R}^d$ we have

$$\langle x, y \rangle = x^\top y \,.$$

Having defined an inner product, it is straight-forward to define the *Euclidean norm* of a vector. The Euclidean norm $\|\cdot\| : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ of a vector $x \in \mathbb{R}^d$ is defined as

$$\|x\| := \sqrt{\langle x, x \rangle} = \sqrt{\sum_{j=1}^{d} x_j^2} \,.$$

Please note that sometimes we will denote the Euclidean norm by $\|\cdot\|_2$ rather than $\|\cdot\|$ in order to distinguish them from other norms. Another module-relevant vector norm is the *one-norm* $\|\cdot\|_1 : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$, which is defined as

$$\|x\|_1 := \sum_{j=1}^{d} |x_j| \,.$$

There are numerous ways of defining a *matrix norm*, but the only two relevant matrix norms for this module are the *Frobenius norm*, i.e.

$$\|X\|_{\text{Fro}} := \sqrt{\sum_{i=1}^{d_1}\sum_{j=1}^{d_2} x_{ij}^2}\,,$$

and the standard matrix norm

$$\|X\| := \sup_{\|y\|\leq 1} \|Xy\| = \sup_{y\neq 0} \frac{\|Xy\|}{\|y\|}\,.$$

Here $\|\cdot\|$ denotes the Euclidean norm, and sup is the supremum. Before we move on to recall basic concepts of calculus, we want to briefly address the concepts of *eigenvalues* and *eigenvectors* of square matrices. An eigenvalue λ and an eigenvector w_λ are characterised by the equation

$$Xw_\lambda = \lambda w_\lambda\,. \tag{1.1}$$

This means that w_λ is invariant under matrix multiplication albeit a scaling with factor λ. If we take an inner product of (1.1) with w_λ, we immediately observe that an eigenvalue λ takes on the value

$$\lambda = \frac{\langle Xw_\lambda, w_\lambda\rangle}{\|w_\lambda\|^2}\,.$$

Assuming that all eigenvectors are normalised, i.e. $\|w_\lambda\| = 1$, we conclude $\lambda = \langle Xw_\lambda, w_\lambda\rangle$. For eigenvalues σ^2 and eigenvectors v_σ of $X^\top X$, i.e.

$$X^\top X v_\sigma = \sigma^2 v_\sigma\,,$$

we observe $\sigma = \|Xv_\sigma\|/\|v_\sigma\| \geq 0$. This implies that the matrix norm equals the square-root of the largest eigenvalue of $X^\top X$, i.e.

$$\|X\| = \sup_{v\neq 0}\frac{\|Xv\|}{\|v\|} = \sup\left\{\sigma\in\mathbb{R}_{\geq 0}\;\middle|\; X^\top Xv = \sigma^2 v\right\}\,.$$

The square-root of the eigenvalues of $X^\top X$, i.e. σ, are known as *singular values* of X. More information and properties of singular values can be found here. The eigenvectors are known as the *right singular vectors* of X. In identical fashion we can derive eigenvectors u_σ of $XX^\top$, which are known as the *left singular vectors* of X. In this module, the most important properties are that Xw, $X^\top y$, $X^\top Xw$ and $(X^\top X)^{-1}b$ (if $(X^\top X)^{-1}$ exists) can be expressed in terms of the *singular value decomposition* of X, i.e.

$$X = U\Sigma V^\top\,, \tag{1.2}$$

where $U\in\mathbb{R}^{s\times\min(s,d)}$ and $V\in\mathbb{R}^{d\times\min(s,d)}$ are the matrices that contain all singular vectors $\{u_{\sigma_i}\}_{i=1}^{\min(s,d)}$ and $\{v_{\sigma_i}\}_{i=1}^{\min(s,d)}$ as their columns, while $\Sigma\in\mathbb{R}^{\min(s,d)\times\min(s,d)}$ is the diagonal matrix whose diagonal contains all singular vectors $\sigma_1\geq\sigma_2\geq\ldots\geq\sigma_{\min(s,d)}$. With the help of Equation (1.2), we can express Xw, $X^\top y$, $X^\top Xw$ and $(X^\top X)^{-1}b$ as

$$Xw = \sum_{j=1}^{\min(s,d)} \sigma_j u_j\langle v_j, w\rangle\,,\qquad X^\top y = \sum_{j=1}^{\min(s,d)} \sigma_j v_j\langle u_j, y\rangle\,,$$

and

$$X^\top X w = \sum_{j=1}^{\min(s,d)} \sigma_j^2 v_j \langle v_j, w \rangle \,, \qquad (X^\top X)^{-1} b = \sum_{j=1}^{\min(s,d)} \sigma_j^{-2} v_j \langle v_j, b \rangle \,.$$

For more information on singular value decompositions we refer to this page.

1.2 Calculus

We will require the computation of (partial) derivatives, gradients and Hessian matrices during this module. Suppose we are given a continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$, then its *derivative* $f' : \mathbb{R} \to \mathbb{R}$ is defined as

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

The notation f' is known as Lagrange's notation and is the most common notation for derivatives. Another useful notation is Leibniz's notation $\frac{df}{dx}$ that emphasises that the derivative of f with respect to (w.r.t.) x is taken. Many useful differentiation rules exist. If you require a little refresher you can recall many of those rules here. For continuously differentiable functions $f : \mathbb{R}^d \to \mathbb{R}$ in multiple variables we can define *partial derivatives* as

$$\frac{\partial f}{\partial x_j}(x_1, \ldots, x_d) := \lim_{h \to 0} \frac{f(x_1, \ldots, x_{j-1}, x_j + h, x_{j+1}, \ldots, x_d) - f(x_1, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_d)}{h} \,.$$

If we compute the partial derivatives w.r.t. all arguments $x_1, \ldots, x_d$ and store them in a column-vector, we obtain the *gradient* $\nabla f : \mathbb{R}^d \to \mathbb{R}^d$ of a function $f : \mathbb{R}^d \to \mathbb{R}$, i.e.

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) & \frac{\partial f}{\partial x_2}(x) & \cdots & \frac{\partial f}{\partial x_d}(x) \end{pmatrix} \,.$$

Here $x = (x_1, \ldots, x_d)$ is a short-hand vector notation for all arguments $x_1, \ldots, x_d$. For function $f : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ with multiple outputs we can define the *Jacobian matrix* $J_f : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1 \times d_2}$ as

$$J_f(x) := \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_{d_2}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_{d_1}}{\partial x_1} & \cdots & \frac{\partial f_{d_1}}{\partial x_{d_2}} \end{pmatrix} \,.$$

For many differentiation rules there exist higher-dimensional counterparts. Of particular interest to us is the multi-dimensional chain-rule, which for a composition $f \circ g$ of functions $f : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ and $g : \mathbb{R}^{d_3} \to \mathbb{R}^{d_2}$ reads

$$J_{f \circ g}(x) = J_f(g(x)) J_g(x) \,.$$

In particular, for functions $f : \mathbb{R}^{d_1} \to \mathbb{R}$ and $g : \mathbb{R}^{d_2} \to \mathbb{R}^{d_1}$ we observe

$$\nabla (f \circ g)(x) = \nabla f(g(x))^\top J_g(x) \,.$$

We conclude this section with second-order derivatives for multi-variable functions. A *second partial derivative* is the application of the partial derivative to a partial derivative, assuming that such a partial derivative exists, i.e.

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) := \frac{\partial f}{\partial x_i}\frac{\partial f}{\partial x_j}(x)\,, \qquad \text{for} \qquad i,j \in \{1,\dots,d\}\,,$$

for $x = (x_1,\dots,x_d)$ and a function $f : \mathbb{R}^d \to \mathbb{R}$. If $i = j$, we simply write $\partial^2 f/\partial x_i^2$. Based on this concept, one can define the *Hessian matrix* $H_f : \mathbb{R}^d \to \mathbb{R}^{d\times d}$ of second-order partial derivatives as

$$H_f(x) := \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{pmatrix},$$

for $x = (x_1,\dots,x_d)$, assuming that all second-order partial derivatives exist. Note that gradient, Jacobian and Hessian are connected via

$$H_f(x) = J_{\nabla f}(x)\,.$$

This concludes our section on calculus. In the next and final section of the mathematical preliminaries, we revisit some basic rules and notations of probability & statistics.

1.3 Probability & statistics

The *expected value*, or *expectation*, of a random variable X with finite outcomes $\{x_i\}_{i=1}^s$ with probabilities $\{\rho_i\}_{i=1}^s$ is defined as

$$\mathbb{E}_i[x_i] := \sum_{i=1}^{s} x_i \rho_i\,. \tag{1.3}$$

Since probabilities are non-negative and sum up to one, the expected value (1.3) is a *weighted average*. In case all outcomes are equiprobable, i.e. $\rho_i = 1/s$ for all $i \in \{1,\dots,s\}$, the expected value is the normal *average*, or *arithmetic mean*.
In the absolutely continuous case, the expectation is defined as

$$\mathbb{E}_x[x] = \int_{\mathbb{R}} x\,\rho(x)\,dx\,,$$

assuming that the cumulative distribution function of its underlying random variable X admits a probability density function (PDF) ρ and that the above integral exists.
Please note that expectations can also be computed for measurable functions f, i.e.

$$\mathbb{E}_x[f(x)] := \int_{\mathbb{R}} f(x)\,\rho(x)\,dx\,. \tag{1.4}$$

For the example of an indicator function over the one-dimensional interval $[a,b]$, i.e.

$$\iota_{[a,b]}(x) = \begin{cases} 1 & x \in [a,b] \\ 0 & x \notin [a,b] \end{cases},$$

we instantly observe

$$\mathbb{E}_x\left[\iota_{[a,b]}(x)\right] = \int_{\mathbb{R}} \iota_{[a,b]}(x)\,\rho(x)\,dx = \int_a^b \rho(x)\,dx = P(a \leq X \leq b)\,.$$

The right-hand-side denotes the probability that a random variable takes a value that lies in the interval $[a, b]$.
The *variance* of a random variable X is defined as

$$\begin{aligned}\mathrm{Var}_x[x] &:= \mathbb{E}_x\left[(x - \mathbb{E}_x[x])^2\right]\,,\\ &= \mathbb{E}_x[x^2] - \mathbb{E}_x[x]^2\,.\end{aligned}$$

The square-root of the variance, i.e. $\sigma_x := \sqrt{\mathrm{Var}_x[x]}$, is known as the *standard deviation*.

Note that expectations and variances can easily be extended to multi-variable functions. If we have functions f in two (absolutely continuous) random variables X and Y from a joint cumulative distribution with underlying PDF ρ for example, we simply write

$$\mathbb{E}_{x,y}[f(x,y)] = \int_{\mathbb{R}}\int_{\mathbb{R}} f(x,y)\,\rho(x,y)\,dx\,dy\,.$$

Note that two random variables X and Y are said to be *independent* if their probabilities or their PDFs factor, i.e.

$$\rho(x,y) = \rho_X(x)\rho_Y(y)\,.$$

For an arbitrary number n of random variables $\{X_i\}_{i=1}^n$, we have

$$\rho(x_1,\ldots,x_n) = \prod_{i=1}^n \rho_{X_i}(x_i)\,.$$

A collection of random variables is *independent and identically distributed (iid)* if each random variable has the same probability distribution (and thus the same PDF) and if all random variables are independent. Hence, we can write the joint PDF as

$$\rho(x_1,\ldots,x_n) = \prod_{i=1}^n \tilde{\rho}(x_i)\,,$$

where $\tilde{\rho}$ denotes the PDF of the underlying probability distribution. We conclude this chapter with the definitions of the *likelihood function* and the *posterior probability*. Let X be a (continuous) random variable with probability density function $\rho_\theta(x)$ that depends on parameters θ. Then the function

$$\rho(x\,|\,\theta) := \rho_\theta(x)$$

is the *likelihood function* of θ, given the outcome x of X, or the probability of outcome x for the parameter value θ.

Example 1.1 (Probability vs Likelihood)**.** Rat or mouse example here...?

The *posterior probability* on the other hand is the probability of the parameters θ given the outcome x of X, i.e. $\rho(\theta\,|\,x)$. Given the *prior* probability distribution function $\rho(\theta)$ for the parameters θ, both are connected via *Bayes' rule* or *Bayes' theorem* for PDFs:

$$\rho(\theta\,|\,x) = \frac{\rho(x\,|\,\theta)\rho(\theta)}{\rho(x)}\,, \tag{1.5}$$

named after Reverend Thomas Bayes. With this we conclude this chapter on mathematical preliminaries and begin our introduction of supervised machine learning.

Chapter 2

Supervised learning

In this module we extensively study numerous aspects of supervised machine learning. The two predominant problems in supervised machine learning are regression and classification problems. Beginning with a statistical motivation for supervised linear regression problems, we spent the first half of this module on regression problems, before moving on to classification problems.

2.1 Statistical motivation

In supervised machine learning, the goal is to find a function mapping $f : \mathbb{R}^n \to \mathbb{R}^m$ that approximately maps a collection of s known input arguments $\{x_i\}_{i=1}^s$, with $x_i \in \mathbb{R}^n$ for all $i \in \{1, \dots, s\}$, onto a collection of s output elements $\{y_i\}_{i=1}^s$, with $y_i \in \mathbb{R}^m$ for all $i \in \{1, \dots, s\}$, i.e.

$$f(x_i) \approx y_i \,, \qquad \forall i \in \{1, \dots, s\} \,. \tag{2.1}$$

The name 'supervised' stems from the fact that a collection of outputs $\{y_i\}_{i=1}^s$ needs to be known in advance. In the following, we want to specify what we mean by 'approximately', and why it is not necessarily desirable to have a strict equality in (2.1).

In Figure 2.1 we see a collection of data points $\{(x_i, y_i)\}_{i=1}^s$ that represent the weight- and height-information for s individuals, so the goal is to find a mapping that approximately returns the height-information when given the weight-information. We observe that the data points seem to follow a linear trend, but with substantial and presumably random deviations from this line. Instead of seeking a function that maps every x_i onto each corresponding y_i, one could rather try to find a line that has minimal distance to each point (x_i, y_i). What distance, you may ask? If we collect the error in prediction for each sample we obtain a plot like the one in Figure 2.1. The error seems to follow a normal distribution, which is why it makes sense to model the deviation as normal-distributed independent random variables with mean zero and variance σ^2, i.e. for

$$\varepsilon_i := y_i - f(x_i) \,, \qquad \forall i \in \{1, \dots, s\} \,,$$

we know that every ε_i is distributed according to

$$\rho(\varepsilon_i|0, \sigma) = \mathcal{N}(\varepsilon_i|0, \sigma) := \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\varepsilon_i^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(f(x_i)-y_i)^2}{2\sigma^2}} \,.$$

Assuming that all ε_i are independent random variables, the joint probability density reads as

$$\rho(\varepsilon|0, \sigma) = \prod_{i=1}^{s} \rho(\varepsilon_i|0, \sigma) = (2\pi\sigma^2)^{-\frac{s}{2}} \prod_{i=1}^{s} e^{-\frac{(f(x_i)-y_i)^2}{2\sigma^2}} \,.$$

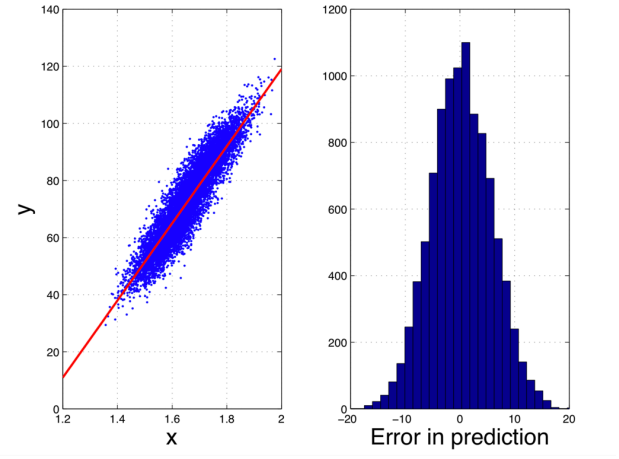


Figure 2.1: A simple linear regression example for data points with weights- (x-axis) and height-information (y-axis). The figure on the left shows a line fitted to the given data points that has minimal mean-squared error w.r.t. all data points in the sense that it minimises (2.3). The plot on the right-hand-side shows the the mean-squared error of the individual samples w.r.t. the fitted line.

If we assume that our model f is parametrised with parameters w, which in the following we denote as f_w, it seems wise to choose those parameters such that the likelihood for the joint probability distribution is maximised, i.e. we look for parameters $\hat{w}$ that satisfy

$$\begin{aligned}\hat{w} &= \arg\max_w \left\{ \prod_{i=1}^{s} \rho(\varepsilon_i|0,\sigma) \right\}, \\ &= \arg\max_w \left\{ (2\pi\sigma^2)^{-\frac{s}{2}} \prod_{i=1}^{s} e^{-\frac{(f_w(x_i)-y_i)^2}{2\sigma^2}} \right\}.\end{aligned}$$

Due to the monotonicity of the natural logarithm, we can alternatively estimate $\hat{w}$ by estimating the minimiser of the negative log-likelihood, i.e.

$$\begin{aligned}\hat{w} &= \arg\min_w \left\{ -\log\left(\prod_{i=1}^{s} \rho(\varepsilon_i|0,\sigma) \right) \right\}, \\ &= \arg\min_w \left\{ -\sum_{i=1}^{s} \log\left(\rho(\varepsilon_i|0,\sigma)\right) \right\}, \\ &= \arg\min_w \left\{ \frac{s}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{s}(f_w(x_i)-y_i)^2 \right\}, \\ &= \arg\min_w \left\{ \frac{1}{2}\sum_{i=1}^{s}(f_w(x_i)-y_i)^2 \right\}, \\ &= \arg\min_w \left\{ \frac{1}{2s}\sum_{i=1}^{s}(f_w(x_i)-y_i)^2 \right\}. \end{aligned} \tag{2.2}$$

Hence, optimal parameters $\hat{w}$ have to be chosen in order to minimise the squared Euclidean norm

w.r.t. each sample $\{(x_i, y_i)\}_{i=1}^s$. The function

$$\mathrm{MSE}(w) := \frac{1}{2s}\sum_{i=1}^{s}(f_w(x_i) - y_i)^2 \tag{2.3}$$

is known as the *mean-squared error* (MSE) and minimising it is known as the *method of least-squares*. In the following we want to address the question of how to parametrise f_w and start with a linear model.

2.2 Linear & polynomial regression

In linear regression, we parametrise our model f_w in terms of a linear transformation, i.e. for a weight $w \in \mathbb{R}^{d+1}$ and $f_w : \mathbb{R}^d \to \mathbb{R}$ we choose

$$f_w(x) := \langle x, w\rangle = \sum_{j=0}^{d} x_j w_j\,, \tag{2.4}$$

where we define $x_0 = 1$ in order to allow scalar translations w_0. Suppose we are given s pairs of input/output samples $\{(x_i, y_i)\}_{i=1}^s$, we can estimate a weight w following (2.2) by minimising the least-squares error with respect to all samples, i.e.

$$w_t = \operatorname*{arg\,min}_{w\in\mathbb{R}^{d+1}} \left\{ \frac{1}{2s}\sum_{i=1}^{s} |\langle x_i, w\rangle - y_i|^2 \right\}. \tag{2.5}$$

An alternative way of writing (2.5) is in terms of matrix multiplication and Euclidean norm as for a matrix

$$X := \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_{s1} & x_{s2} & \cdots & x_{sd} \end{pmatrix},$$

and a vector $y := \begin{pmatrix} y_1 & y_2 & \dots & y_s \end{pmatrix}$. Later in this module we will verify that the unique solution to (2.5) are weights w_t that satisfy

$$\nabla \mathrm{MSE}(w_t) = 0\,. \tag{2.6}$$

It will be left as a coursework exercise to show that the solution of (2.6) is the linear system of equations of the form

$$\begin{aligned} &\sum_{i=1}^{s} \begin{pmatrix} x_{i0}^2 & x_{i0}x_{i1} & \cdots & x_{i0}x_{id} \\ x_{i1}x_{i0} & x_{i1}^2 & \cdots & x_{i1}x_{id} \\ \vdots & \vdots & \ddots & \vdots \\ x_{id}x_{i0} & x_{id}x_{i1} & \cdots & x_{id}^2 \end{pmatrix} \hat{w} = \sum_{i=1}^{s} y_i x_i\,, \\ \Leftrightarrow \quad &\begin{pmatrix} \|x_0\|^2 & \langle x_0, x_1\rangle & \cdots & \langle x_0, x_d\rangle \\ \langle x_1, x_0\rangle & \|x_1\|^2 & \cdots & \langle x_1, x_d\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_d, x_0\rangle & \langle x_d, x_1\rangle & \cdots & \|x_d\|^2 \end{pmatrix} \hat{w} = \sum_{i=1}^{s} y_i x_i\,. \end{aligned} \tag{2.7}$$

Note that here the inner product is with respect to the sample-dimension, i.e. $\langle x_k, x_l \rangle = \sum_{i=1}^{s} x_{ik} x_{il}$. We want to mention that we can easily extend (2.4) to functions with m-dimensional output via

$$f_W(x) := \begin{pmatrix} \langle x, w_1 \rangle \\ \langle x, w_2 \rangle \\ \vdots \\ \langle x, w_m \rangle \end{pmatrix},$$

or equivalently $f_W(x) = W^\top x$ in matrix-multiplication form, for the matrix $W \in \mathbb{R}^{(d+1)\times m}$ defined as

$$W := \begin{pmatrix} | & | & & | \\ w_1 & w_2 & \dots & w_m \\ | & | & & | \end{pmatrix}$$

In many applications, the underlying true function f is nonlinear in its argument and linear models are insufficient in terms of their systematic bias. A simple class of nonlinear model functions are polynomials.

2.2.1 Polynomial regression

In polynomial regression the goal is to fit polynomials of degree d to the data samples. This can easily be achieved by equipping (2.4) with a vector of polynomials of degree d, i.e.

$$f_w(x) := \langle \phi(x), w \rangle = \sum_{j=0}^{d} \phi(x)_j w_j \,, \tag{2.8}$$

with $\phi : \mathbb{R} \to \mathbb{R}^{d+1}$ defined as

$$\phi(x) := \begin{pmatrix} 1 & x & x^2 & \dots & x^d \end{pmatrix}^\top .$$

Note that (2.8) is nonlinear in the argument x, but linear in the weight vector w. Suppose we have s pairs $\{(x_j, y_j)\}_{j=1}^{s}$ as usual, then for each x_i we have $\phi(x_i) \in \mathbb{R}^{d+1}$. We can, therefore, define the short-hand notations

$$\Phi(X) := \begin{pmatrix} \phi(x_1)^\top \\ \phi(x_2)^\top \\ \vdots \\ \phi(x_s)^\top \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^d \\ 1 & x_2 & x_2^2 & \dots & x_2^d \\ \vdots & & & & \\ 1 & x_s & x_s^2 & \dots & x_s^d \end{pmatrix} \in \mathbb{R}^{s\times(d+1)}, \qquad \text{and} \qquad y := \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_s \end{pmatrix} \in \mathbb{R}^s .$$

As in the linear regression case we can find optimal weights by solving the least-squares problem

$$w_t = \underset{w\in\mathbb{R}^{(d+1)}}{\arg\min} \left\{ \frac{1}{2s} \|\Phi(X)w - y\|^2 \right\} \tag{2.9}$$

where $\|\cdot\|$ denotes the Euclidean norm, or

$$W_t = \underset{W\in\mathbb{R}^{(d+1)\times m}}{\arg\min} \left\{ \frac{1}{2s} \|\Phi(X)W - Y\|_{\text{Fro}}^2 \right\}$$

in case we want to parametrise a function with m-dimensional output. Here $\|\cdot\|_{\text{Fro}}$ denotes the Frobenius matrix norm, and $Y \in \mathbb{R}^{s\times m}$ is a matrix.

2.2.2 Regression with general basis functions

The previous example of polynomial regression is just a special case of regression with more general (nonlinear) basis functions. The parametrisation of f_w remains the same as in (2.8), but the basis functions $\phi : \mathbb{R}^m \to \mathbb{R}^{d+1}$ can represent more general classes of functions and not just polynomials. A typical example are *radial basis functions*, i.e.

$$\phi(x) := \begin{pmatrix} \varphi(\|x - \mu_0\|) & \varphi(\|x - \mu_1\|) & \dots & \varphi(\|x - \mu_d\|) \end{pmatrix}^\top ,$$

for a function $\varphi : \mathbb{R} \to \mathbb{R}$ and points $\{\mu_i\}_{i=0}^d$ with $\mu_i \in \mathbb{R}^m$ for each $i \in \{0, \dots, d\}$. Classical choices for φ are

$$\varphi(x) := \begin{cases} 1 - \frac{1}{\alpha}|x| & |x| \le \alpha \\ 0 & |x| > \alpha \end{cases} , \qquad \text{and} \qquad \varphi(x) := \exp\left(-\frac{x^2}{2\sigma^2}\right) .$$

In later sections we want to discuss how to analyse and minimise rather general empirical risk functions with loss functions ℓ that are not necessarily quadratic functions. Before we do so, we want to recall some basic concepts in convex analysis first.

2.3 Convex analysis

In this section we want to more closely investigate cost- or loss-functions such as the MSE (2.3) introduced in the previous section. In particular, we want to define convexity of a function, verify when a function is convex and show why convexity is useful.

Definition 2.1 (Convex set)**.** *A subset $\mathcal{C} \subset \mathbb{R}^n$ is said to be* convex *if*

$$\lambda w + (1 - \lambda)v \in \mathcal{C} ,$$

for all elements $w, v \in \mathcal{C}$ and $\lambda \in [0, 1]$.

Based on this definition we can go ahead and define convex functions.

Definition 2.2 (Convex function)**.** *A function $E : \mathcal{C} \to \mathbb{R}$ is said to be* convex *if for all arguments $w, v \in \mathbb{R}^n$ and scalars $\lambda \in [0, 1]$ we observe*

$$E(\lambda w + (1 - \lambda)v) \le \lambda E(w) + (1 - \lambda)E(v) .$$

In return, we can define concave functions as follows.

Definition 2.3 (Concave function)**.** *A function $E : \mathcal{C} \to \mathbb{R}$ is said to be* concave *if for all arguments $w, v \in \mathbb{R}^n$ and scalars $\lambda \in [0, 1]$ we observe*

$$E(\lambda w + (1 - \lambda)v) \ge \lambda E(w) + (1 - \lambda)E(v) .$$

Note that for a convex function E we automatically observe that $-E$ is concave, and vice versa. Before we continue, we want to introduce an extremely useful concept for the analysis of convex and continuously differentiable functions: Bregman distances, respectively Bregman divergences.

Definition 2.4 (Bregman distance)**.** *Let $E : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function, i.e. $\nabla E(w)$ exists for all $w \in \mathbb{R}^n$ and is continuous. Then its corresponding* Bregman distance *$D_E : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is defined as*

$$D_E(u, v) := E(u) - E(v) - \langle \nabla E(v), u - v \rangle ,$$

for all arguments $u, v \in \mathbb{R}^n$.

Note that Bregman distances are not necessarily distances in the sense of a metric. They merely describe the distance of a function E at point u to its linearisation around v. Before we continue to show that this distance is non-negative if and only if E is convex, we first want to introduce another useful distance based on the sum of two Bregman distances.

Definition 2.5 (Jensen-Shannon distance)**.** *Suppose $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function defined over a convex set $\mathcal{C}$. Then its* Jensen-Shannon distance *is defined as*

$$J_E^\lambda(u, v) := \lambda D_E(u, w) + (1 - \lambda) D_E(v, w)\,,$$

for $w := \lambda u + (1 - \lambda)v \in \mathcal{C}$ and $\lambda \in [0, 1]$.

Straight-forward computations reveal that this Jensen-Shannon distance is nothing but the difference between the convex combination of $E(u)$ and $E(v)$ and E applied to the convex combination of u and v, i.e.

$$J_E^\lambda(u, v) = \lambda E(u) + (1 - \lambda)E(v) - E(\lambda u + (1 - \lambda)v)\,. \tag{2.10}$$

Remark 2.1. From the Definition 2.2 we instantly observe that $J_E^\lambda(u, v) \geq 0$ for all $u, v \in \mathcal{C}$ if and only if E is convex.

Remark 2.2. For the special choice $\lambda = \frac{1}{2}$ the Jenson-Shannon distance reads

$$J_E^{\frac{1}{2}}(u, v) = \frac{1}{2}\left(D_E(u, w) + D_E(v, w)\right) = \frac{1}{2}(E(u) + E(v)) - E\left(\frac{u + v}{2}\right)\,,$$

and is known as the *Burbea-Rao distance*. In particular, the Burbea-Rao distance is symmetric, i.e $J_E^{\frac{1}{2}}(u, v) = J_E^{\frac{1}{2}}(v, u)$.

From (2.10) we immediately observe that the Jensen-Shannon distance is non-negative if and only if E is convex, which immediately follows from the definition of convexity, Definition 2.2. The following corollary shows that also the Bregman distance is non-negative if and only if E is convex.

Corollary 2.1. *Let $E : \mathcal{C} \to \mathbb{R}$ be a differentiable function defined over a convex set $\mathcal{C} \subset \mathbb{R}^n$. Then E is convex if and only if its corresponding Bregman distance D_E is non-negative for all arguments, i.e.*

$$D_E(u, v) \geq 0\,,$$

for all $u, v \in \mathcal{C}$.

Proof (non-examinable). $\Rightarrow$: We first consider the case $n = 1$. Given $u, v \in \mathcal{C}$, we conclude $u + \lambda(v - u) \in \mathcal{C}$ due to the convexity of $\mathcal{C}$. From the convexity of E we observe

$$\begin{aligned} & E(u + \lambda(v - u)) \leq (1 - \lambda)E(u) + \lambda E(v)\,, \\ \Rightarrow \qquad & \lambda E(v) - \lambda E(u) - (E(u + \lambda(v - u)) - E(u)) \geq 0\,, \\ \Rightarrow \qquad & E(v) - E(u) - \frac{E(u + \lambda(v - u)) - E(u)}{\lambda} \geq 0\,. \end{aligned}$$

Taking the limit $\lambda \to 0$ of the left-hand-side then yields

$$\lim_{\lambda \to 0} E(v) - E(u) - \frac{E(u + \lambda(v - u)) - E(u)}{\lambda} = E(v) - E(u) - E'(u)(v - u)\,;$$

hence, we can conclude

$$E(v) - E(u) - E'(u)(v-u) \geq 0\,. \tag{2.11}$$

For general n we define the function $g(\lambda) := E((1-\lambda)u + \lambda v)$. By applying the chain rule we obtain

$$g'(\lambda) = \langle \nabla E((1-\lambda)u + \lambda v), v-u \rangle\,.$$

Since E is convex, it is straight-forward to see that g is also convex and that (2.11) therefore implies

$$g(s) - g(t) - g'(t)(s-t) \geq 0\,,$$

for any $s, t \in [0,1]$. For the particular choices $s = 1$ and $t = 0$ this inequality reads as $g(1) - g(0) - g'(0) \geq 0$, which is equivalent to

$$E(v) - E(u) - \langle \nabla E(u), v-u \rangle \geq 0\,.$$

Consequently, the Bregman distance D_E is non-negative for all input arguments in $\mathcal{C}$.
$\Leftarrow$: suppose we have $u, v \in \mathcal{C}$, then $w \in \mathcal{C}$ with $w := u + \lambda(v-u)$ due to the convexity of $\mathcal{C}$. Since the Bregman distance is non-negative for all arguments in $\mathcal{C}$, we can conclude

$$D_E(u,w) \geq 0 \qquad \text{and} \qquad D_E(v,w) \geq 0\,,$$

and therefore also

$$J_E^\lambda(u,v) = \lambda D_E(u,w) + (1-\lambda) D_E(v,w) \geq 0$$

for $\lambda \in [0,1]$. This implies

$$0 \leq J_E^\lambda(u,v) = \lambda E(u) + (1-\lambda)E(v) - E(\lambda u + (1-\lambda)v)\,.$$

Hence, the function E is indeed convex. □

There is a close connection between convexity and global minimisers of a function. A global minimiser is defined as follows.

Definition 2.6 (Global minimiser)**.** *Suppose $E : \mathbb{R}^n \to \mathbb{R}$ is a function for which there exists a constant $\hat{E}$ with*

$$-\infty < \hat{E} \leq E(w)\,, \qquad \forall w \in \mathbb{R}^n\,,$$

i.e. the infimum of E exists and is attained. Then $\hat{E}$ is called the global minimum*. Moreover, let $\hat{w}$ denote the argument for which $E(\hat{w}) = \hat{E}$ and, hence,*

$$E(\hat{w}) \leq E(w)$$

holds true, for all $w \in \mathbb{R}^n$. Then $\hat{w}$ is known as the global minimiser *of E.*

For convex functions $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ that are also differentiable, i.e. $\nabla E(w)$ exists for all $w \in \mathbb{R}^n$, we observe that their gradient can help us to determine a global minimiser of E.

Lemma 2.1. *Suppose $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ is a convex function that is bounded from below and also differentiable. Then an argument $\hat{w}$ satisfies $\nabla E(\hat{w}) = 0$ if and only if $\hat{w}$ is a global minimiser.*

Proof. Since E is convex, we know $D_E(w, \hat{w}) \geq 0$ for all $w \in \mathcal{C}$ due to Corollary 2.1. This is equivalent to

$$E(w) - E(\hat{w}) - \langle \underbrace{\nabla E(\hat{w})}_{=0}, w - \hat{w} \rangle \geq 0 \,,$$

which proves that $\hat{w}$ is a global minimiser. □

Last but not least, we verify that a twice differentiable function in one variable is convex if and only if the second derivative is non-negative for all arguments.

Corollary 2.2 (Second order convexity)**.** *A twice differentiable one-dimensional function $E : \mathcal{C} \to \mathbb{R}$ over a convex set $\mathcal{C} \subset \mathbb{R}$ is convex if and only if $E''(x) \geq 0$ for all $x \in \mathcal{C}$.*

Proof (non-examinable). We verify that this condition is equivalent to Corollary 2.1:
$\Rightarrow$: We assume that E satisfies $E''(w) \geq 0$ for all $w \in \mathcal{C}$. Then we can conclude for arbitrary $w, v \in \mathcal{C}$ with $v < w$ that

$$0 \leq \int_v^w E''(u)(w-u)\,du = \big[E'(u)(w-u)\big]_{u=v}^{u=w} + \int_v^w E'(u)\,du = E(w) - E(v) - E'(v)(w-v)$$

is satisfied, which proves that $E''(w) \geq 0$ implies $D_E(w, v) \geq 0$.
$\Leftarrow$: We assume that $D_E(w, v) \geq 0$ is true for all $w, v \in \mathcal{C}$. Hence, we conclude

$$E'(v)(w-v) \leq E(w) - E(v) \leq E'(w)(w-v) \,,$$

which implies $E'(v)(w-v) \leq E'(w)(w-v)$. Subtracting $E'(v)(w-v)$ on both sides of the inequality and dividing by $(w-v)^2$ then yields $0 \leq \frac{E'(w)-E'(v)}{w-v}$ for all $w, v \in \mathcal{C}$. Taking the limit $w \to v$, i.e.

$$0 \leq \lim_{w \to v} \frac{E'(w) - E'(v)}{w - v} = E''(w) \,,$$

concludes the proof. □

Based on the previous considerations we know that (2.7) can only be a solution of (2.5) if $\nabla E(\hat{w}) = 0$ for $E(w) := \frac{1}{2s} \sum_{i=1}^{s} |\langle x_i, w \rangle - y_i|^2$ and

$$\hat{w} = \begin{pmatrix} \|x_1\|^2 & \langle x_1, x_2 \rangle & \cdots & \langle x_1, x_n \rangle \\ \langle x_2, x_1 \rangle & \|x_2\|^2 & \cdots & \langle x_2, x_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle x_n, x_1 \rangle & \langle x_n, x_2 \rangle & \cdots & \|x_n\|^2 \end{pmatrix}^{-1} \sum_{i=1}^{s} y_i x_i \,.$$

In order to verify this, we need to compute the gradient ∇E and show that it coincides with the previous equation, which is left as a coursework exercise.

Note that we can also write the scalar product as the multiplication of a row with a column vector, i.e. $\langle x, w\rangle = x^\top w$. With this notation we can write the previous problem in a more compact form by defining

$$X := \begin{pmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_s^\top \end{pmatrix} \in \mathbb{R}^{s\times n} \qquad \text{and} \qquad y := \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_s \end{pmatrix} \in \mathbb{R}^s .$$

Using this notation the linear regression problem (2.5) then reads

$$\hat{w} = \operatorname*{arg\,min}_{w\in\mathbb{R}^n} \left\{ \frac{1}{2s} \|Xw - y\|^2 \right\} .$$

The corresponding optimality condition $\nabla E(\hat{w}) = 0$ reads as

$$X^\top X\hat{w} = X^\top y\,, \tag{2.12}$$

which is also known as the normal equation associated with $X\hat{w} = y$.

2.3.1 A comment on existence and uniqueness

Before we investigate issues of regression problems due to ill-conditioning of the data matrices we want to briefly talk about existence and uniqueness of regression problems of the form (2.12). Do solutions to (2.12) always exist, and if they exist, are they unique? To answer the first question, we require the following lemma.

Lemma 2.2. *For every matrix $X \in \mathbb{R}^{s\times(d+1)}$ we observe*

$$\ker(X^\top X) = \ker(X) \qquad \textit{and} \qquad \operatorname{ran}(X^\top X) = \operatorname{ran}(X^\top)\,.$$

Proof (non-examinable). For $w \in \ker(X)$ we know $Xw = 0$. Multiplying $X^\top$ from the left, i.e. $X^\top Xw = 0$, therefore implies $w \in \ker(X^\top X)$ and, thus, $\ker(X) \subset \ker(X^\top X)$. For $w \in \ker(X^\top X)$ we know $X^\top Xw = 0$. Taking the inner product with w then yields $0 = \langle X^\top Xw, w\rangle = \langle Xw, Xw\rangle = \|Xw\|^2$, which already implies $Xw = 0$. Hence, we concluded $w \in \ker(X)$, and thus, $\ker(X^\top X) \subset \ker(X)$. As an immediate consequence, we conclude $\ker(X) = \ker(X^\top X)$.

A fundamental statement in linear algebra says that $\ker(X)^\perp = \operatorname{ran}(X^\top)$, where $\perp$ denotes the orthogonal complement. Based on the first part of the proof, we verify

$$\operatorname{ran}(X^\top) = \ker(X)^\perp = \ker(X^\top X)^\perp = \ker((X^\top X)^\top)^\perp = \operatorname{ran}(X^\top X)\,,$$

which concludes the proof. □

An immediate consequence of Lemma 2.2 is that a solution of (2.12) always exists.

Theorem 2.1. *There always exists a solution of the Normal Equation* (2.12)*, for all $X \in \mathbb{R}^{s\times(d+1)}$ and $y \in \mathbb{R}^s$.*

Proof (non-examinable). By definition we have $X^\top y \in \operatorname{ran}(X^\top)$. From Lemma 2.2 we know that $\operatorname{ran}(X^\top) = \operatorname{ran}(X^\top X)$, which implies $X^\top y \in \operatorname{ran}(X^\top X)$. This means that there must exist a $w \in \mathbb{R}^{d+1}$ with $X^\top Xw = X^\top y$. □

Now that we know that a solution to (2.12) always exists, we need to ask ourselves when that solution is unique. We immediately see that this is the case if and only if $\ker(X) = \{0\}$. If the kernel of X contains elements v other than 0, we observe

$$X^\top y = X^\top X\hat{w} = X^\top(X\hat{w} + \underbrace{Xv}_{=0}) = X^\top X(\hat{w} + v)\,,$$

due to the linearity of the matrix-matrix- and matrix-vector product. Hence, $\hat{w} + v$ is also a solution of (2.12). Note that this is not a contradiction to $\hat{w}$ being a global minimiser of (2.3), since

$$\mathrm{MSE}(\hat{w} + v) = \mathrm{MSE}(\hat{w})\,.$$

Amongst other challenges, we are going to address the non-uniqueness issue in the following section.

2.4 Ill-conditioned regression problems & regularisation

Solving a finite-dimensional regression problem can be challenging if the underlying data matrix is ill-conditioned. Before we define what this means, we want to lead with a simple example. Suppose we want to fit a line to two data point pars (x_1, y_1) and (x_2, y_2) with $x_1 = 1 - c$, $y_1 = 1$, $x_2 = 1 + c$ and $y_2 = 1$. As in the previous section, we can solve the regression problem via (2.12) for the matrix

$$X = \begin{pmatrix} 1 & 1-c \\ 1 & 1+c \end{pmatrix}$$

and the data vector $y = (y_1, y_2)^\top = \begin{pmatrix} 1 & 1 \end{pmatrix}^\top$. The linear system that we obtain reads

$$\begin{pmatrix} 1 & 1 \\ 1 & 1+c^2 \end{pmatrix} \hat{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}\,,$$

or

$$\hat{w} = \begin{pmatrix} 1+c^{-2} & -c^{-2} \\ -c^{-2} & c^{-2} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix}\,,$$

if we solve for $\hat{w}$ directly. It is straight-forward to see (or to calculate) that $\hat{w} = \begin{pmatrix} 1 & 0 \end{pmatrix}^\top$ is the unique solution of this linear system, which makes perfect sense as the data points lie on a line with constant shift $\hat{w}_0 = 1$ and slope $\hat{w}_1 = 1$. We now want to investigate how things change if we make an error when taking our measurements. Suppose instead of y we measure $y_\delta = \begin{pmatrix} 1-\varepsilon & 1+\varepsilon \end{pmatrix}^\top$, for some constant $\varepsilon > 0$. The linear system remains the same, but the right-hand-side changes, and we have to solve

$$\begin{pmatrix} 1 & 1 \\ 1 & 1+c^2 \end{pmatrix} \hat{w}_\delta = \begin{pmatrix} 1 \\ 1+c\varepsilon \end{pmatrix}\,,$$

respectively

$$\hat{w}_\delta = \begin{pmatrix} 1+c^{-2} & -c^{-2} \\ -c^{-2} & c^{-2} \end{pmatrix} \begin{pmatrix} 1 \\ 1+c\varepsilon \end{pmatrix}\,;$$

hence, the solution to this problem reads $\hat{w}_\delta = \begin{pmatrix} 1 - c^{-1}\varepsilon & c^{-1}\varepsilon \end{pmatrix}^\top$. Now we observe something interesting: if ε is reasonably small and the constant c is significantly larger, not much changes in terms of the solution. Consider e.g. the values $c = 1/2$ and $\varepsilon = 1/100$. We immediately compute $\hat{w}_\delta = \begin{pmatrix} 0.98 & 0.02 \end{pmatrix}^\top$, which is not very different from $\hat{w} = \begin{pmatrix} 1 & 0 \end{pmatrix}^\top$. However, we get a completely different picture if c is substantially smaller than ε. If we for instance choose $c = 1/1000$ while ε remains $\varepsilon = 1/100$, the solution $\hat{w}_\delta$ changes to $\hat{w}_\delta = \begin{pmatrix} -9 & 10 \end{pmatrix}^\top$. This is very different from $\hat{w}$. The error in the data $\delta := \|y - y_\delta\| = \sqrt{2}\varepsilon$ is very small ($\sqrt{2}/100 \approx 0.0141$), whereas the error in the reconstruction, i.e. $\|\hat{w} - \hat{w}_\delta\| = \sqrt{164} \approx 12.8063 \gg 0.0141$, is heavily amplified. In this particular example it is probably not surprising to see such a phenomenon: the closer the two inputs x_1 and x_2 move together, the more significant the impact of small variations in y on the slope and shift of the regression line. But can this phenomenon also appear in more complex scenarios?

To address this question we want to look at the solution of (2.12) for more general data matrices X (or matrices $\Phi(X)$) and outputs y and y_δ with the help of the singular value decomposition. For simplicity, we focus on the X notation, but all steps can be carried out for any matrix. Suppose that $X = U\Sigma V^\top$ is the singular value decomposition of X. Let us remind how in the classic regression problem $X \in \mathbb{R}^{s\times d+1}$ where s are the samples and d is the number independent variables used. In these settings $U \in \mathbb{R}^{s\times s}$, $\Sigma \in \mathbb{R}^{s\times d+1}$, and $V \in \mathbb{R}^{d+1\times d+1}$. Hence we can easily see how $X^\top X = V\Gamma^2 V^\top$, where Γ^2 is defined as a squared $\mathbb{R}^{d+1\times d+1}$ matrix with squared singular values on the diagonal. Indeed:

$$X^\top X = (U\Sigma V^\top)^\top U\Sigma V^\top = V\Sigma^\top U^\top U\Sigma V^\top = V\Sigma^\top\Sigma V^\top = V\Gamma^2 V^\top \tag{2.13}$$

We assume that $X^\top X$ is invertible and that $d < s$ we can write:

$$(X^\top X)^{-1} = (V\Gamma^2 V^\top)^{-1} = V\Gamma^{-2}V^\top \tag{2.14}$$

where we used the orthogonality of V, i.e., $V^\top V = V^{-1}V = 1$. Hence, we rewrite (2.3) in terms of the singular value decomposition of X, we obtain

$$\hat{w} = V\Gamma^{-2}V^\top X^\top y \qquad \text{and} \qquad \hat{w}_\delta = V\Gamma^{-2}V^\top X^\top y_\delta\,,$$

we can now use that $X^\top y$ (where in these settings $y \in \mathbb{R}^{s\times 1}$ and $X^\top \in \mathbb{R}^{d+1\times s}$) can be written as $X^\top y = V\Sigma^\top U^\top y$. Hence, we have

$$\hat{w} = V\Gamma^{-2}V^\top V\Sigma^\top U^\top y \qquad \text{and} \qquad \hat{w}_\delta = V\Gamma^{-2}V^\top V\Sigma^\top U^\top y_\delta\,,$$

that leads to

$$\hat{w} = V(\Sigma^\top)^{-1}U^\top y \qquad \text{and} \qquad \hat{w}_\delta = V(\Sigma^\top)^{-1}U^\top y_\delta\,,$$

where $(\Sigma^\top)^{-1}$ is a $\mathbb{R}^{d+1\times s}$ matrix whose elements are all zero except for a $\mathbb{R}^{d+1\times d+1}$ block where in the diagonal we have the inverse of the singular values. Hence,

$$\hat{w} - \hat{w}_\delta = V(\Sigma^\top)^{-1}U^\top(y - y_\delta)$$

for their difference. Computing the Euclidean norm of this difference yields

$$\begin{aligned}
\|\hat{w} - \hat{w}_\delta\|^2 &= \|V(\Sigma^\top)^{-1}U^\top(y - y_\delta)\|^2 \\
&\leq \|V\|^2\|(\Sigma^\top)^{-1}\|^2\|U^\top\|^2\|(y - y_\delta)\|^2 \\
&\leq \|(\Sigma^\top)^{-1}\|^2\|(y - y_\delta)\|^2 \\
&= \frac{\delta^2}{\sigma_{d+1}^2}\,.
\end{aligned}$$

The previous estimate implies that, in the worst case scenario, the error in the data gets amplified by a factor of $1/\sigma_{d+1}$, where σ_{d+1} is the smallest singular value. If σ_{d+1} get arbitrarily close to zero, this factor can become arbitrarily large. If we assume that y and y_δ are of the form $y = Xw^\dagger$ and $y_\delta = Xv_\delta$ with $\|w^\dagger - v_\delta\| \leq \epsilon$, we can further estimate

$$\begin{aligned}\|\hat{w} - \hat{w}_\delta\|^2 &\leq \|(\Sigma^\top)^{-1}\|^2 \|y - y_\delta\|^2 \\ &= \|(\Sigma^\top)^{-1}\|^2 \|X(w^\dagger - v_\delta)\|^2 \\ &\leq \|(\Sigma^\top)^{-1}\|^2 \|X\|^2 \|w^\dagger - v_\delta\| \\ &= \kappa(X)^2 \|w^\dagger - v_\delta\|^2 \leq \kappa(X)^2 \epsilon^2 \,,\end{aligned}$$

where $\kappa(X) := \sigma_1/\sigma_{d+1}$ is the so-called *condition number* of X that determines the amplification of the error ε in the worst case scenario. The consequence of this exercise is that we now know that the ratio of the largest and smallest singular value of X is important for the amplification of errors in our data. If we have any influence on the data collection process, we should aim to collect data points $\{x_i\}_{i=1}^s$ such that the matrix X has a small condition number, i.e. X is *well-conditioned*. Matrices X with large condition numbers are called *ill-conditioned*. In the following we discuss how to compensate ill-conditioning with what is known to be Tikhonov regularisation or ridge regression.

2.4.1 Ridge regression

In the previous section we have seen that worst-case error amplification is a consequence of data matrices with large condition numbers. A relatively straight-forward idea to combat instability is by approximating the original regression problem by a problem with lower condition number. Writing the left-hand-side of the normal equation (2.12), i.e. $X^\top X\hat{w} = X^\top y$, in terms of its singular value decomposition reads

$$\sum_{j=1}^{d+1} \sigma_j^2 v_j \langle v_j, \hat{w}\rangle \,. \tag{2.15}$$

We now replace (2.15) with a version where we shift the singular values by a constant, positive factor α, i.e.

$$\sum_{j=1}^{d+1} (\sigma_j^2 + \alpha) v_j \langle v_j, w_\alpha\rangle \,. \tag{2.16}$$

Reverting back from (2.16) to the matrix-vector-multiplication representation, we have effectively modified the normal equation to

$$\Big(X^\top X + \alpha I\Big) w_\alpha = X^\top y \,, \tag{2.17}$$

where $I \in \{0,1\}^{(d+1)\times(d+1)}$ denotes the identity matrix. The nice thing about $X^\top X + \alpha I$ is that its condition number is $\kappa(X^\top X + \alpha I) = \sqrt{(\sigma_1^2 + \alpha)/(\sigma_{d+1}^2 + \alpha)}$, instead of $\kappa(X^\top X) = \sigma_1/\sigma_{d+1}$. If we have a matrix $X^\top X$ with $\sigma_1 = \sqrt{2}$ and $\sigma_{d+1} = 1/\sqrt{2000000}$ as in the previous section for example, we have a large condition number of $\kappa = 2000$. If we consider (2.17) instead of (2.12) for $\alpha = 1$, we observe $\kappa(X^\top X + \alpha I) \approx \sqrt{3} \ll 2000 = \kappa(X^\top X)$. As a consequence, any worst-case error amplification for the ridge regression is much smaller, at the cost of potentially altering the

original problem dramatically. One thing that is also not clear is how to choose the parameter α; we will address this question in the next section. Before we do so, we want to present an alternative characterisation of (2.17).

Theorem 2.2 (Ridge regression)**.** *The solution w_α of* (2.17) *is the unique solution of the optimisation problem*

$$w_\alpha = \operatorname*{arg\,min}_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2} \|Xw - y\|^2 + \frac{\alpha}{2} \|w\|^2 \right\} . \tag{2.18}$$

Proof. This proof is left as an exercise. □

The beauty of formulation (2.18) is that we immediately see that we still try to minimise the mean-squared error, but at the same time also ensure that the squared norm of the weights does not become too large. Both goals have to be balanced with a reasonable choice of α. We discuss this choice in the next section. Note that Problem (2.18) is known as *ridge regression* in the context of machine learning and *Tikhonov regularisation* in the context of inverse & ill-posed problems. The parameter α is known as the *regularisation parameter*. In the following section we want to investigate how to choose hyperparameters such as the regularisation parameter in some optimal way.

2.5 Model selection

Generally speaking, the major goal in most machine learning problems is to approximate (or interpolate) an unknown function $\hat{f} : \mathbb{R}^n \to \mathbb{R}^m$ for a given set of data points sampled from an unknown distribution $\mathcal{D}$. The assumption that we are in a supervised learning setting implies that we sample corresponding pairs (x, y) of input **and** output data points. The goal of supervised machine learning is then to find $\hat{f}$ such that it minimises the *population risk*, *expected risk* or *expected error*, which is defined as

$$E(f) := \mathbb{E}_{x,y} \left[\ell(f(x), y) \right] , \tag{2.19a}$$

$$= \int_{(x,y)\in\mathcal{D}} \ell(f(x), y) \rho(x, y) \, dx \, dy . \tag{2.19b}$$

Here $\ell : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a so-called loss function that measures the difference between $f(x)$ and y, while $\rho : \mathcal{D} \to \mathbb{R}$ is the unknown joint probability density function. Note that the latter implies

$$\rho(x, y) \geq 0 \quad \text{a.e.} \qquad \text{and} \qquad \int_{(x,y)\in\mathcal{D}} \rho(x, y) \, dx \, dy = 1 .$$

The key problem with computing the population risk E, let alone minimise it, is that we do not have access to it as we do not know ρ. In practice, all we can do is to draw $|S|$ i.i.d. samples from $\mathcal{D}$ and consider minimising the *empirical risk*

$$L_S(f) := \frac{1}{|S|} \sum_{(x_i,y_i)\in S} \ell(f(x_i), y_i) \tag{2.20}$$

instead. Here $|S|$ denotes the cardinality of the set S, respectively the number of elements of S. The difference of (2.19) and (2.20) is known as the *generalisation error*

$$\begin{aligned} G_S(f) &= E(f) - L_S(f)\,, \\ &= \mathbb{E}_{x,y}\left[\ell(f(x), y)\right] - \frac{1}{|S|}\sum_{(x_i,y_i)\in S} \ell(f(x_i), y_i)\,. \end{aligned}$$

Note that we have defined $\hat{f}$ as the function that minimises (2.19), i.e.

$$\hat{f} = \arg\min_{f\in\mathcal{F}} \mathbb{E}_{(x,y)\in\mathcal{D}}\left[\ell(f(x), y)\right]\,,$$

where $\mathcal{F}$ denotes some suitable function space. Now we assume that we want to approximate $\hat{f}$ with a parametric function f_{w_t}, whose parameters w_t are computed by minimising the empirical risk on a set of data points S_t that are sampled from the distribution $\mathcal{D}$; we will refer to this set of points as the *training set*. The empirical risk for this function over the set S_t is then known as the *training error*, which reads

$$L_{S_t}(f_{w_t}) = \frac{1}{|S_t|}\sum_{(x_i,y_i)\in S_t} \ell(f_{w_t}(x_i), y_i)\,,$$

and we have f_{w_t} with w_t defined as

$$w_t := \arg\min_{w\in\mathbb{R}^{d+1}} L_{S_t}(f_w)\,.$$

If we sample a different set of data points from the same distribution $\mathcal{D}$ and denote this set as the *validation set* or *test set* S_v, then we can define the *validation error*

$$L_{S_v}(f_{w_t}) = \frac{1}{|S_v|}\sum_{(x_i,y_i)\in S_v} \ell(f_{w_t}(x_i), y_i)\,.$$

The validation error is of particular interest, as it approximates the population risk in expectation. An algorithm or method that aims at approximating $\hat{f}$ via a parametric function f_{w_t} is said to generalise, if the generalisation error G_{S_v} converges to zero if the number of samples converges to infinity, i.e $\lim_{|S_v|\to\infty} G_{S_v}(f_{w_t}) = 0$. Since E cannot be computed, the generalisation error cannot be computed either. Instead, the goal of research in statistical learning is to bound the generalisation error in probability.

Example 2.1 (Ridge regression)**.** Suppose we have collected a set of samples and use them as our training dataset $S_t := \{(x_i, y_i) \in \mathcal{D} \,|\, i \in \{1, \dots, s\}\}$. In linear ridge regression, we recall that the idea is to approximate $\hat{f}$ via $f_{w_t}(x) := \langle\phi(x), w_t\rangle$, where the weights w_t are computed via

$$w_t = \arg\min_{w\in\mathbb{R}^{d+1}} \left\{ \frac{1}{2|S_t|}\sum_{i=1}^{s} |\langle\phi(x_i), w\rangle - y_i|^2 + \frac{\alpha}{2}\|w\|^2 \right\}\,.$$

Here, $\phi : \mathbb{R}^m \to \mathbb{R}^{d+1}$ is the data-augmentation map that we had introduced earlier and $\alpha > 0$ is the regularisation parameter. We can then compute the validation error on a set of different samples $S_v := \{(x_i, y_i) \in \mathcal{D} \setminus S_t \,|\, i \in \{1, \dots, s\}\}$ via

$$L_{S_v}(f_{w_t}) = L_{S_v}(\langle\phi(x_i), w_t\rangle) = \frac{1}{2|S_v|}\sum_{(x_i,y_i)\in S_v} \left(\langle\phi(x_i), w_t\rangle - y_i\right)^2\,.$$

Choosing an optimal regularisation parameter $\hat{\alpha}$ in terms of the validation error can then be formulated as the bilevel optimisation problem

$$\hat{\alpha} = \operatorname*{arg\,min}_{\alpha \geq 0} L_{S_v}(\langle \phi(x_i), w_t \rangle) \quad \text{subject to} \quad w_t = \operatorname*{arg\,min}_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2s} \sum_{i=1}^{s} |\langle \phi(x_i), w \rangle - y_i|^2 + \frac{\alpha}{2} \|w\|^2 \right\}. \tag{2.21}$$

In practice, this hyperparameter $\hat{\alpha}$ is often determined by applying a grid-search strategy to approximately solve (2.21).

In the following section we will further analyse the validation error for a new sample in expectation over all possible training sets in the context of ℓ being the mean-squared error function.

2.6 Bias-variance decomposition

Following up on the previous section, we now assume the abstract data generation model

$$y_\varepsilon = \hat{f}(x) + \varepsilon \,,$$

where $\hat{f}$ is some unknown, deterministic function mapping inputs x onto outputs y. We assume that the outputs y_ε that we measure are $y = \hat{f}(x)$ plus a random variable ε drawn from some distribution $\mathcal{D}_\varepsilon$ with zero expectation, and variance σ^2, i.e.

$$\mathbb{E}_\varepsilon [\varepsilon] = 0 \qquad \text{and} \qquad \mathrm{Var}_\varepsilon[\varepsilon] = \sigma^2 \,.$$

One can easily verify that this implies

$$\mathbb{E}_\varepsilon[y_\varepsilon] = \mathbb{E}_\varepsilon[\hat{f}(x)] = \hat{f}(x) \qquad \text{and} \qquad \mathrm{Var}_\varepsilon(y_\varepsilon) = \sigma^2$$

in particular. Further, we assume that each pair (x, y_ε) is a sample from an unknown distribution $\mathcal{D}$ and that we have collected a finite number of samples from this distribution in each set S_t. Given a parametrised prediction function f_{w_t} based on one set S_t, we want to investigate the expected squared error of the difference of $y_\varepsilon = \hat{f}(x) + \varepsilon$ and f_{w_t} at a specific pair of points $(\tilde{x}, \tilde{y})$. Hence, we investigate

$$\mathbb{E}_{t,\varepsilon} \left[\left(\hat{f}(\tilde{x}) + \varepsilon - f_{w_t}(\tilde{x}) \right)^2 \right] ,$$

where $\mathbb{E}_t$ denotes the expectation over the sets S_t, while $\tilde{x}$ is a new sample outside of the training set. We have also dropped the factor $1/2$ for notational convenience. In the following, we want to show that we can split this expectation into three important components: the so-called *bias*, the

variance and the noise variance. In particular, we observe

$$
\begin{aligned}
&\mathbb{E}_{t,\varepsilon}\left[\left(\hat{f}(\tilde{x})+\varepsilon-f_{w_t}(\tilde{x})\right)^2\right],\\
&=\mathbb{E}_{t,\varepsilon}\left[\varepsilon^2+2\,\varepsilon\left(\hat{f}(\tilde{x})-f_{w_t}(\tilde{x})\right)+\left(\hat{f}(\tilde{x})-f_{w_t}(\tilde{x})\right)^2\right],\\
&=\mathrm{Var}_\varepsilon[\varepsilon]+\mathbb{E}_t\left[2\,\mathbb{E}_\varepsilon[\varepsilon]\left(\hat{f}(\tilde{x})-f_{w_t}(\tilde{x})\right)\right]+\mathbb{E}_t\left[\left(\hat{f}(\tilde{x})-f_{w_t}(\tilde{x})\right)^2\right],\\
&=\sigma^2+\mathbb{E}_t\left[\left(\hat{f}(\tilde{x})-f_{w_t}(\tilde{x})\right)^2\right],\\
&=\sigma^2+\mathbb{E}_t\left[\left(\hat{f}(\tilde{x})-\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]+\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]-f_{w_t}(\tilde{x})\right)^2\right],\\
&=\sigma^2+\mathbb{E}_t\left[\left(\hat{f}(\tilde{x})-\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right)^2\right]+\mathbb{E}_t\left[\left(\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]-f_{w_t}(\tilde{x})\right)^2\right]\\
&+2\mathbb{E}_t\left[\left(\hat{f}(\tilde{x})-\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right)\left(\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]-f_{w_t}(\tilde{x})\right)\right],\\
&=\sigma^2+\mathbb{E}_t\left[\left(\hat{f}(\tilde{x})-\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right)^2\right]+\mathbb{E}_t\left[\left(\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]-f_{w_t}(\tilde{x})\right)^2\right]\\
&+2\mathbb{E}_t\left[\hat{f}(\tilde{x})\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right]-2\mathbb{E}_t\left[\hat{f}(\tilde{x})f_{w_t}(\tilde{x})\right]-2\mathbb{E}_t\left[\mathbb{E}_t^2\left[f_{w_t}(\tilde{x})\right]\right]+2\mathbb{E}_t\left[f_{w_t}(\tilde{x})\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right],\\
&=\sigma^2+\mathbb{E}_t\left[\left(\hat{f}(\tilde{x})-\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right)^2\right]+\mathbb{E}_t\left[\left(\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]-f_{w_t}(\tilde{x})\right)^2\right]\\
&+2\hat{f}(\tilde{x})\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]-2\hat{f}(\tilde{x})\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]-2\mathbb{E}_t^2\left[f_{w_t}(\tilde{x})\right]+2\mathbb{E}_t^2\left[f_{w_t}(\tilde{x})\right],\\
&=\sigma^2+\left(\hat{f}(\tilde{x})-\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right)^2+\mathbb{E}_t\left[\left(\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]-f_{w_t}(\tilde{x})\right)^2\right],\\
&=\sigma^2+\mathrm{Bias}_t\left[f_{w_t}(\tilde{x})\right]^2+\mathrm{Var}_t\left[f_{w_t}(\tilde{x})\right],
\end{aligned}
$$

for

$$
\begin{aligned}
\sigma^2&:=\mathrm{Var}_\varepsilon[\varepsilon],\\
\mathrm{Bias}_t\left[f_{w_t}(\tilde{x})\right]&:=\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]-\hat{f}(\tilde{x}),\\
\mathrm{Var}_t\left[f_{w_t}(\tilde{x})\right]&=\mathbb{E}_t\left[\left(f_{w_t}(\tilde{x})-\mathbb{E}_t\left[f_{w_t}(\tilde{x})\right]\right)^2\right].
\end{aligned}
$$

Intuitively, the noise variance is an irreducible error in the measurements, the bias refers to a systematic model error, while the variance of a learning method indicates how much it will move around its mean. If we for example try to approximate a nonlinear function $\hat{f}$ with a linear function f_{w_t}, we will inevitably observe a systematic bias of our model fit. In the previous lecture we referred to this phenomenon as *underfitting*. If we try to fit a f_{w_t} to even the smallest fluctuation of $\hat{f}$, we observe the phenomenon of *overfitting*.

2.7 The LASSO

In Section 2.4.1 we have introduced the ridge regression model as a way to deal with ill-conditioned regression problems. In the previous two sections we have further explained how ridge regression can be used to prevent overfitting of a model function. In Theorem 2.2 we have characterised the

ridge regression problem as an optimisation problem of the form

$$w_\alpha = \underset{w\in\mathbb{R}^{d+1}}{\arg\min}\left\{\frac{1}{2}\|\Phi(X)w - y\|^2 + \frac{\alpha}{2}\|w\|^2\right\}.$$

In this section we want to replace the squared Euclidean norm regularisation term in (2.18) with a one-norm, i.e.

$$w_\alpha = \underset{w\in\mathbb{R}^{d+1}}{\arg\min}\left\{\frac{1}{2}\|\Phi(X)w - y\|^2 + \alpha\|w\|_1\right\}, \tag{2.22}$$

with $\|w\|_1 := \sum_{j=0}^{d}|w_j|$. The motivation behind using the one-norm instead of the squared Euclidean lies in recovering simpler weights. Simple in this context means sparse, i.e. that we only want some coefficients of w_α to be non-zero while most coefficients are in fact zero. If we think of polynomial regression as an example, we could think of fitting a polynomial with a very high degree. Without knowing which coefficients of a polynomial should be relevant, we could potentially use the minimisation problem in Equation (2.22) to recover a weight vector with only a few non-zero entries. Intuitively, we can prevent overfitting this way, and we do not have to limit the degree in advance in order to do so. A regression problem of the form of (2.22) is known as the *Least Absolute Shrinkage and Selection Operator (LASSO)*. Before we discuss how to solve (2.22) computationally, we want to introduce a very basic and popular method for smooth optimisation, known as gradient descent.

2.7.1 Gradient descent

Gradient descent is an iterative procedure that aims at minimising general, differentiable functions E. Gradient descent is of the form

$$w^{k+1} = w^k - \tau\nabla E(w^k), \tag{2.23}$$

for some energy E, an initial value $w^0 \in \mathbb{R}^n$ and a step-size parameter $\tau > 0$. In case of $E(w) = \frac{1}{2s}\|Xw - y\|^2$ for example, gradient descent reads

$$\begin{aligned} w^{k+1} &= w^k - \frac{\tau}{s}X^\top\left(Xw^k - y\right), \\ &= \left(I - \frac{\tau}{s}X^\top X\right)w^k + \frac{\tau}{s}X^\top y. \end{aligned} \tag{2.24}$$

The advantage of iteratively solving (2.24) is that we only need to compute matrix multiplications and basic arithmetic operations that are reasonably cheap. Also, with an algorithm like (2.23) we can address minimisation problems more general than minimising the MSE, which is going to be useful for the LASSO problem (2.22). On the downside, we yet have to determine when and under which conditions (2.23) really converges to a solution of the minimisation problem $\hat{w} = \arg\min_{w\in\mathbb{R}^n} E(w)$ and how quickly it converges to that solution. For this, we rewrite Equation (2.23) to

$$\begin{aligned} w^{k+1} &= \underset{w\in\mathbb{R}^n}{\arg\min}\left\{E(w^k) + \langle\nabla E(w^k), w\rangle + \frac{1}{2\tau}\|w - w^k\|^2\right\}, \\ &= \underset{w\in\mathbb{R}^n}{\arg\min}\left\{E(w) + \frac{1}{2\tau}\|w - w^k\|^2 - E(w) + E(w^k) + \langle\nabla E(w^k), w - w^k\rangle\right\} \\ &= \underset{w\in\mathbb{R}^n}{\arg\min}\left\{E(w) + \frac{1}{2\tau}\|w - w^k\|^2 - D_E(w, w^k)\right\}. \end{aligned}$$

Algorithm 1 Gradient descent

Specify: Differentiable, convex function $E : \mathbb{R}^n \to \mathbb{R}$, step-size $\tau > 0$, index K
Initialise: $w^0 \in \mathbb{R}^n$
Iterate:
1: **for** $k = 0, \dots, K-1$ **do**
2: $\quad w^{k+1} = w^k - \tau \nabla E(w^k)$
3: **end for**
return w^K.

The objective function $L^k(w) := \langle \nabla E(w^k), w \rangle + \frac{1}{2\tau}\|w - w^k\|^2$ is convex and differentiable with gradient $\nabla L(w) = \nabla E(w^k) + \frac{1}{\tau}(w - w^k)$. Hence, the global minimiser can be determined via $\nabla L(w^{k+1}) = 0$, which yields (2.23). Gradient descent is summarised in Algorithm 1. The questions that we need to ask ourselves now are the following: does Algorithm 1 converge to a minimiser of the objective function E and if yes, under what conditions does it converge? The following definition will come in handy for answering this question.

Definition 2.7 (L-smooth functions)**.** *A continuously differentiable function $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ is called* L-smooth *if*

$$\|\nabla E(w) - \nabla E(v)\| \leq L\|w - v\|$$

is guaranteed for all $w, v \in \mathcal{C}$ and a positive constant $L > 0$.

Theorem 2.3 (Convergence of Algorithm 1)**.** *Let $E : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R}$ be a convex and $1/\tau$-smooth function in the sense of Definition 2.7. Suppose $\hat{w}$ denotes a global minimiser of E, i.e. $\hat{w} = \arg\min_{w \in \mathbb{R}^n} E(w)$. Then the iterates of Algorithm 1 satisfy*

$$E(w^k) - E(\hat{w}) \leq \frac{C}{k}\,, \tag{2.25}$$

for a constant $C > 0$ that is independent of k. As a direct consequence, we observe $\lim_{k\to\infty} E(w^k) = E(\hat{w})$.

As usual, *the proof of this statement is non-examinable* and is left to the interested reader. In order to prove Theorem 2.3 we need to verify the following two lemmas first. The first one verifies a convenient property of smooth functions E: if E is $1/\tau$-smooth, then a function $J := \frac{1}{2\tau}\|\cdot\|^2 - E$ is automatically convex.

Lemma 2.3. *Let $E : \mathcal{C} \to \mathbb{R}$ be a $1/\tau$-smooth function over a convex domain $\mathcal{C} \subset \mathbb{R}^n$, for a positive constant $\tau > 0$. Then $J : \mathcal{C} \to \mathbb{R}$, with*

$$J(w) := \frac{1}{2\tau}\|w\|^2 - E(w)\,,$$

is a convex function, for all $w \in \mathcal{C}$.

Proof (non-examinable): As E is $1/\tau$-smooth we conclude

$$\|\nabla E(w) - \nabla E(v)\| \leq \frac{1}{\tau}\|w - v\|\,.$$

Multiplying both sides with $\|w - v\|$ and making use of the Cauchy-Schwartz inequality $\langle x, y\rangle \leq \|x\|\|y\|$ leaves us with

$$\langle \nabla E(w) - \nabla E(v), w - v\rangle \leq \frac{1}{\tau}\|w - v\|^2 = \frac{1}{\tau}\langle w - v, w - v\rangle\,.$$

Subtracting the left-hand-side from the right-hand-side yields the inequality

$$0 \leq \left\langle \frac{1}{\tau}w - \nabla E(w) - \left(\frac{1}{\tau}v - \nabla E(v)\right), w - v\right\rangle = D_J(w, v) + D_J(v, w)\,,$$

for $J := \frac{1}{2\tau}\|\cdot\|^2 - E$. From $D_J(w, v) + D_J(v, w) \geq 0$ for all u, v we can conclude $D_J(y + t(x - y), y) + D_J(y, y + t(x - y)) \geq 0$ for all x, y and $t \in [0, 1]$, which implies

$$\langle \nabla J(y + t(x - y)) - \nabla J(y), x - y\rangle \geq 0\,. \tag{2.26}$$

If we define

$$f(t) := J(y + t(x - y))\,,$$

we observe $f'(t) = \langle \nabla J(y + t(x - y)), x - y\rangle$ and can therefore conclude $f'(t) \geq f'(0)$ from (2.26). Hence, we can estimate

$$\begin{aligned} J(x) = f(1) = f(0) + \int_0^1 f'(t)\,dt &\geq f(0) + f'(0) \\ &= J(y) - \langle \nabla J(y), x - y\rangle\,, \end{aligned}$$

which is true for all x, y. Hence, $D_J(w, v) + D_J(v, w) \geq 0$ for all arguments already implies $D_J(u, v) \geq 0$ for all arguments u, v. Corollary 2.1 then implies convexity of J. □

Before we continue with the actual proof of Theorem 2.3 we also verify the following intermediate result.

Lemma 2.4. *Let the same assumptions hold true as in Theorem 2.3 and suppose w^* is defined as $w^* := \arg\min_{w\in\mathbb{R}^n}\{E(w) + D_J(w, \overline{w})\}$ for some $\overline{w} \in \mathbb{R}^n$. Then the identity*

$$E(w^*) + D_E(w, w^*) + D_J(w, w^*) + D_J(w^*, \overline{w}) = E(w) + D_J(w, \overline{w})$$

holds for any $w \in \mathbb{R}^n$. Here $J : \mathbb{R}^n \to \mathbb{R}$ is defined as $J(w) := \frac{1}{2\tau}\|w\|^2 - E(w)$ and D_E and D_J denote the Bregman distances with respect to the functions E, respectively J.

Proof (non-examinable): As a consequence of Lemma 2.1 we can characterise w^* via the optimality condition

$$0 = \nabla E(w^*) + \nabla J(w^*) - \nabla J(\overline{w})\,.$$

Taking an inner product with $w^* - w$ then yields

$$\begin{aligned} 0 &= -\langle \nabla E(w^*), w - w^*\rangle - \langle \nabla J(w^*) - \nabla J(\overline{w}), w - w^*\rangle\,, \\ &= D_E(w, w^*) - E(w) + E(w^*) - \langle \nabla J(w^*), w - w^*\rangle + \langle \nabla J(\overline{w}), w - w^*\rangle\,, \\ &= D_E(w, w^*) - E(w) + E(w^*) + D_J(w, w^*) - J(w) + J(w^*) + \langle \nabla J(\overline{w}), w - w^*\rangle\,, \\ &= D_E(w, w^*) - E(w) + E(w^*) + D_J(w, w^*) - J(w) + J(w^*) + \langle \nabla J(\overline{w}), w - \overline{w} + \overline{w} - w^*\rangle\,, \\ &= D_E(w, w^*) - E(w) + E(w^*) + D_J(w, w^*) - D_J(w, \overline{w}) + D_J(w^*, \overline{w})\,, \end{aligned}$$

which concludes the proof. □

Example 2.2 (Convergent gradient descent for minimising the MSE)**.** For the MSE function $E(w) := \frac{1}{2s}\|Xw - y\|^2$ we have already verified $\nabla E(w) = \frac{1}{s}X^\top(Xw - y)$. We further verify

$$\|\nabla E(w) - \nabla E(v)\| = \frac{1}{s}\|X^\top X(w - v)\| \leq \frac{\|X^\top X\|}{s}\|w - v\| = \frac{\|X\|^2}{s}\|w - v\|\,,$$

where $\|X\|$ denotes the operator norm of X (based on the Euclidean vector norm). Hence, E is $1/\tau$-smooth for $\tau \leq s/\|X\|^2$. As a consequence of Theorem 2.3 we know that gradient descent, respectively Algorithm 1, applied to the MSE is convergent for any step-size τ with $\tau \leq s/\|X\|^2$.

Remark 2.3. We want to emphasise that Theorem 2.3 not only guarantees convergence of Algorithm 1, but also provides a rate of convergence. This rate is $1/k$, and often this rate of convergence is highlighted with the big $\mathcal{O}$-notation, i.e.

$$E(w^k) - E(\hat{w}) = \mathcal{O}\left(\frac{1}{k}\right)\,.$$

This means that the left-hand-side is proportional to $1/k$. Suppose $D_J(\hat{w}, w^0) = 10$, then we require approximately $k = 1000$ iterations to ensure $E(w^k) - E(\hat{w}) \leq 10^{-2}$ according to Theorem 2.3. Next semester in Machine Learning II we want to address the question of whether we can have a $1/k^2$-convergence rate (or faster). To illustrate the gain in convergence speed, we would only require $k = 32$ instead of $k = 1000$ iterations in order to get the same accuracy, i.e. $E(w^K) - E(\hat{w}) \leq 10^{-2}$.

Remark 2.4. The convergence rate in Remark 2.3 holds for general convex and differentiable functions E. However, for E in Example 2.2 we can show that the convergence is linear and that it can be characterised in terms of the condition number $\kappa(X)$ of X that we have introduced in Section 2.4. Note that for $E(w) = \frac{1}{2s}\|Xw - y\|^2$ we have

$$\begin{aligned}
\frac{1}{2}\|\nabla E(w)\|^2 &= \frac{1}{2s^2}\|X^\top(Xw - y)\|^2 = \frac{1}{2s^2}\|X^\top(Xw - y - (X\hat{w} - y))\|^2 \\
&\geq \frac{\sigma_{d+1}^2}{s^2}\left(\frac{1}{2}\|Xw - y - (X\hat{w} - y)\|^2\right) = \frac{\sigma_{d+1}^2}{s^2}\left(\frac{1}{2}\|Xw - y\|^2 - \frac{1}{2}\|X\hat{w} - y\|^2\right) \\
&= \frac{\sigma_{d+1}^2}{s^2}\left(E(w) - E(\hat{w})\right)\,,
\end{aligned}$$

for any argument w, where $\hat{w}$ is a minimiser of E that satisfies $\nabla E(\hat{w}) = \frac{1}{s}X^\top(X\hat{w} - y) = 0$, and where σ_{d+1} denotes the smallest singular value of X. Note that this inequality is also known as the *Polyak-Łojasiewicz* inequality. If we choose $\tau = s^2/\|X\|^2 = s^2/\sigma_1^2$, where σ_1 denotes the largest singular value of X, we know that $J(w) = \frac{1}{2\tau}\|w\|^2 - E(w)$ is convex, which implies $D_J(w^{k+1}, w^k) \geq 0$ because of Corollary 2.1, which in return implies the inequality

$$E(w^{k+1}) - E(w^k) \leq \langle \nabla E(w^k), w^{k+1} - w^k\rangle + \frac{\sigma_1^2}{2s^2}\|w^{k+1} - w^k\|^2\,.$$

If we now replace $w^{k+1} - w^k$ with the gradient descent update formulate (2.23), i.e. $w^{k+1} - w^k = -\tau\nabla E(w^k)$ for $\tau = s^2/\sigma_1^2$ and $\nabla E(w^k) = \frac{1}{s}X^\top(Xw^k - y)$, we observe

$$E(w^{k+1}) - E(w^k) \leq -\frac{s^2}{2\sigma_1^2}\|\nabla E(w^k)\|^2\,.$$

Using the Polyak-Łojasiewicz inequality $\frac{1}{2}\|\nabla E(w^k)\|^2 \geq \frac{\sigma_{d+1}^2}{s^2}\left(E(w^k) - E(\hat{w})\right)$ then yields

$$\begin{aligned} E(w^{k+1}) - E(w^k) &\leq \frac{\sigma_{d+1}^2}{\sigma_1^2}\left(E(w^k) - E(\hat{w})\right), \\ &= \kappa(X)^{-2}\left(E(w^k) - E(\hat{w})\right), \end{aligned}$$

where $\kappa(X)^{-2} = 1/\kappa(X)^2$ denotes the squared inverse of the condition number $\kappa(X)$ of X. Adding $E(w^k)$ and subtracting $E(\hat{w})$ on both sides of the inequality leaves us with

$$E(w^{k+1}) - E(\hat{w}) \leq \left(1 - \kappa(X)^{-2}\right)\left(E(w^k) - E(\hat{w})\right).$$

Applying the result recursively for $k = 0, \ldots, K-1$ then yields

$$E(w^K) - E(\hat{w}) \leq \left(1 - \kappa(X)^{-2}\right)^K \left(E(w^0) - E(\hat{w})\right).$$

Hence, for $E(w) = \frac{1}{2s}\|Xw - y\|^2$ the convergence rate can be linear but depends on the condition number κ. If κ is very large, κ^{-2} will be very small, so that $(1 - \kappa(X)^{-2})^K$ will remain close to one, which yields very slow convergence. However, if κ is reasonably small, we can easily get a better convergence rate than $1/k$. Suppose $E(w^0) - E(\hat{w}) = 10$ and $\kappa = 2$, then $K = 25$ iterations are sufficient to ensure $E(w^K) - E(\hat{w}) \leq 10^{-2}$, which is much less than the $K = 1000$ iterations that we considered in Remark 2.3.

2.7.2 Gradient descent and the LASSO

In this section we want to discuss how we can use gradient descent to solve the LASSO problem (2.22). The key challenge is the non-differentiability of the one-norm $\|\cdot\|_1$. We therefore have to make this problem differentiable. We can do this by using the following neat trick. We can rewrite the modulus-function $|\cdot|$ in the one-norm as

$$|z| = \max_{p \in [-1,1]} zp,$$

for any arbitrary scalar $z \in \mathbb{R}$. We now modify the modulus function by subtracting a multiple of a quadratic of the additional variable p and define

$$|z|_\tau := \max_{p \in [-1,1]} zp - \frac{\tau}{2}|p|^2, \tag{2.27}$$

for some scalar parameter $\tau > 0$. A nice thing about this modification is that it has a closed-form solution that we can compute by computing $\hat{p} = \arg\max_{p \in [-1,1]} zp$, which reads

$$\hat{p} = \begin{cases} 1 & z > \tau \\ \frac{z}{\tau} & |z| \leq \tau \\ -1 & z < -\tau \end{cases},$$

(left as a coursework exercise) and inserting $\hat{p}$ into (2.27). This yields

$$|z|_\tau = \begin{cases} |z| - \frac{\tau}{2} & |z| > \tau \\ \frac{1}{2\tau}|z|^2 & |z| \leq \tau \end{cases},$$

which is also known as the Huber loss. This function is differentiable, and we can replace the one-norm $\|w\|_1 = \sum_{j=0}^{d} |w_j|$ in (2.22) with the funtion $H_\tau(w) = \sum_{j=0}^{d} |w_j|_\tau$, i.e.

$$w_\alpha^\tau = \arg\min_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2} \|\Phi(X)w - y\|^2 + \alpha H_\tau(w) \right\} . \tag{2.28}$$

Since all terms in Problem (2.28) are differentiable, we can solve (or approximate a solution of) (2.28) with Algorithm 1. There are obviously a few open questions, such as convexity of the Huber loss, Lipschitz-continuity of the overall gradient and whether minimisers of (2.28) and (2.22) coincide, which we won't address for now. The reason for this is that there is a minor but powerful modification of gradient descent known as proximal gradient descent or forward-backward splitting that is much for suitable for the minimisation of problems of the form (2.22).

2.7.3 Proximal gradient descent

Problems like the LASSO are problems where we minimise the sum of two functions. More precisely, they are of the form

$$w = \arg\min_{w \in \mathcal{C}} \{E(w) + R(w)\} , \tag{2.29}$$

where $E : \mathbb{R}^n \to \mathbb{R}$ is a convex and continuously differentiable function, while $R : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is a proper, convex and lower semi-continuous function. Please do not worry too much about the assumptions on R; these are obviously important from a mathematical point of view but are not really relevant for this module. I have included them for the interested reader, but you can very well survive this module without knowing what those assumptions mean. As already mentioned, typical examples of (2.29) include the LASSO problem (2.22), i.e.

$$w = \arg\min_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2s} \|Xw - y\|^2 + \alpha \|w\|_1 \right\} ,$$

but also other problems such as constrained MSE minimisation, i.e.

$$w = \arg\min_{w \in \mathcal{C}} \left\{ \frac{1}{2s} \|Xw - y\|^2 \right\} = \arg\min_{w \in \mathbb{R}^{d+1}} \left\{ \frac{1}{2s} \|Xw - y\|^2 + \chi_{\mathcal{C}}(w) \right\} ,$$

where $\chi_{\mathcal{C}} : \mathbb{R}^{d+1} \to \mathbb{R} \cup \{\infty\}$ is the characteristic function over the convex set $\mathcal{C} \subset \mathbb{R}^{d+1}$, i.e.

$$\chi_{\mathcal{C}}(w) := \begin{cases} 0 & w \in \mathcal{C} \\ \infty & w \notin \mathcal{C} \end{cases} .$$

We aim to minimise (2.29) with a modification of Algorithm 1 of the form

$$w^{k+1} = \arg\min_{w \in \mathcal{C}} \left\{ E(w) + R(w) + D_J(w, w^k) \right\} ,$$

Algorithm 2 Proximal gradient descent

Specify: Continuously differentiable, convex function $E : \mathbb{R}^n \to \mathbb{R}$, proper, lower semi-continuous and convex function $R : \mathcal{C} \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, step-size $\tau > 0$, index K

Initialise: $w^0 \in \mathcal{C}^n$

Iterate:

1: **for** $k = 0, \dots, K-1$ **do**

2: $\quad w^{k+1} = \operatorname{prox}_{\tau R}\left(w^k - \tau \nabla E(w^k)\right)$

3: **end for**

return w^K.

again for the function $J(w) := \frac{1}{2\tau}\|w\|^2 - E(w)$. This method yields the so-called proximal gradient method that reads

$$\begin{aligned} w^{k+1} &= \arg\min_{w \in \mathcal{C}} \left\{ E(w^k) + \langle \nabla E(w^k), w - w^k \rangle + R(w) + \frac{1}{2\tau}\|w - w^k\|^2 \right\}, \\ &= \arg\min_{w \in \mathcal{C}} \left\{ E(w^k) + \langle \nabla E(w^k), w - w^k \rangle + R(w) + \frac{1}{2\tau}\|w - w^k\|^2 \right\}, \\ &= \arg\min_{w \in \mathcal{C}} \left\{ \frac{1}{2} \left\| w - \left(w^k - \tau \nabla E(w^k)\right) \right\|^2 + \tau R(w) \right\}, \\ &= \operatorname{prox}_{\tau R}\left(w^k - \tau \nabla E(w^k)\right), \end{aligned}$$

for the short-hand notation

$$\operatorname{prox}_{\tau R}(z) := \arg\min_{x \in \mathbb{R}^n} \left\{ \frac{1}{2}\|x - z\|^2 + \tau R(x) \right\}.$$

The mapping $\operatorname{prox}_{\tau R} : \mathbb{R}^n \to \mathcal{C}$ is known as the proximal operator or proximal map with respect to R, and the success of the proximal gradient method relies on the simplicity of this map. In the examples mentioned above, the proximal maps read

$$\left(\operatorname{prox}_{\tau\alpha\|\cdot\|_1}(z)\right)_j = \begin{cases} z_j - \tau\alpha & z_j > \tau\alpha \\ 0 & |z_j| \leq \tau\alpha \\ z_j + \tau\alpha & z_j < -\tau\alpha \end{cases} \quad \text{and} \quad \operatorname{prox}_{\tau\alpha\chi_{\mathcal{C}}}(z) = \operatorname{proj}_{\mathcal{C}}(z),$$

where $\operatorname{proj}_{\mathcal{C}}$ denotes the orthogonal projector onto the set $\mathcal{C}$. If $\mathcal{C} := \{x \in \mathbb{R} \,|\, x \geq 0\}$ for example, then $\operatorname{proj}_{\mathcal{C}}(z) = \max(0, z)$. We have summarised the proximal gradient descent algorithm in Algorithm 2. Convergence can be deduced in very similar fashion as for gradient descent, but we leave this for the Machine Learning II course in the next semester where this becomes more relevant.

2.7.4 Coordinate descent

An interesting observation when minimising convex functions with vector-valued arguments like (2.3), (2.18) or (2.22) is that we can also minimise these functions with respect to each individual argument of the vector-valued argument. Suppose, we have a proper, convex and lower semi-continuous function $F : \mathbb{R}^{1+d} \to \mathbb{R} \cup \{\infty\}$ with vector-valued argument $w = (w_0, w_1, \dots, w_d)^\top$. We can then construct an iterative optimisation procedure by minimising F with respect to each

Algorithm 3 Coordinate descent

Specify: Proper, convex and lower semi-continuous function $F : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, index K
Initialise: $w^0 \in \mathbb{R}^{1+d}$
Iterate:

1: **for** $k = 0, \dots, K-1$ **do**
2: **for** $j = 0, \dots, d$ **do**
3: $w_j^{k+1} = \arg\min_{w \in \mathbb{R}} \left\{ F(w_0^{k+1}, w_1^{k+1}, \dots, w_{j-1}^{k+1}, w, w_{j+1}^k, \dots, w_d^k) \right\}$
4: **end for**
5: **end for**

return w^K.

individual *coordinate* w_j for $j \in \{0, \dots, d\}$ whilst keeping all other coordinates fixed, before moving to the next coordinate. Cycling through the coordinates multiple times then yields what is known as *coordinate descent*, i.e.

$$w_j^{k+1} = \arg\min_{w \in \mathbb{R}} \left\{ F(w_0^{k+1}, w_1^{k+1}, \dots, w_{j-1}^{k+1}, w, w_{j+1}^k, \dots, w_d^k) \right\}, \tag{2.30}$$

for $j \in \{0, \dots, d\}$ and $k \in \mathbb{N}$. A nice property of (2.30) is that we are guaranteed a decrease of the function F after each coordinate descent updddate, i.e.

$$F(w_0^{k+1}, w_1^{k+1}, \dots, w_{j-1}^{k+1}, w_j^{k+1}, w_{j+1}^k, \dots, w_d^k) \leq F(w_0^{k+1}, w_1^{k+1}, \dots, w_{j-1}^{k+1}, w_j^k, w_{j+1}^k, \dots, w_d^k),$$

simply because of the definition of the coordinate descent iteration. Hence, we can conclude a decrease of the function F after each iteration, i.e.

$$F(w_0^{k+1}, w_1^{k+1}, \dots, w_d^{k+1}) \leq F(w_0^k, w_1^k, \dots, w_d^k).$$

However, this does not necessarily mean that coordinate descent is guaranteed to find a global minimiser of F, and you are invited to construct a counter example. The good news is that it can be shown that (2.30) is guaranteed to converge to a global minimiser if the function F is of the form of (2.3), (2.18) or (2.22). The coordinate descent algorithm is summarised in Algorithm 3. In the following, we are taking a closer look at coordinate descent in the context of LASSO regression (2.22).

Example 2.3 (Solving the LASSO with coordinate descent)**.** Note that we can write the function that we minimise in (2.22) as

$$\begin{aligned} F(w_0, \dots, w_l, \dots, w_d) &= \frac{1}{2} \sum_{i=1}^{s} \left| \sum_{j=0}^{d} x_i^j w_j - y_i \right|^2 + \alpha \sum_{j=0}^{d} |w_j|, \\ &= \frac{1}{2} \sum_{i=1}^{s} \left| x_i^l w_l - \left(y_i - \left(\sum_{j=0}^{l-1} x_i^j w_j + \sum_{j=l+1}^{d} x_i^j w_j \right) \right) \right|^2 + \alpha \left(|w_l| + \sum_{j \neq l}^{d} |w_j| \right). \end{aligned}$$

Since $\alpha \sum_{j \neq l}^{d} |w_j|$ does not depend on w_l, we can drop it for the minimisation and write

$$\begin{aligned} w_l^{k+1} &= \arg\min_{w \in \mathbb{R}} \left\{ F(w_0^{k+1}, w_1^{k+1}, \dots, w_{l-1}^{k+1}, w, w_{l+1}^k, \dots, w_d^k) \right\} \\ &= \arg\min_{w \in \mathbb{R}} \left\{ \frac{1}{2} \sum_{i=1}^{s} \left| x_i^l w - \left(y_i - \left(\sum_{j=0}^{l-1} x_i^j w_j^{k+1} + \sum_{j=l+1}^{d} x_i^j w_j^k \right) \right) \right|^2 + \alpha |w| \right\}, \end{aligned} \tag{2.31}$$

for $l \in \{0, \dots, d\}$ and $k \in \mathbb{N}$. Despite being a nonlinear and non-differentiable minimisation problem, the nice thing about (2.31) is that this minimisation problem has a closed-form solution that reads

$$w_l^{k+1} = \frac{1}{\sum_{i=1}^s x_i^{2l}} \begin{cases} \sum_{i=1}^s x_i^l z_i^l - \alpha & \sum_{i=1}^s x_i^l z_i^l > \alpha \\ 0 & \left|\sum_{i=1}^s x_i^l z_i^l\right| \leq \alpha \\ \sum_{i=1}^s x_i^l z_i^l + \alpha & \sum_{i=1}^s x_i^l z_i^l < -\alpha \end{cases}, \tag{2.32}$$

for $z_i^l := y_i - \left(\sum_{j=0}^{l-1} x_i^j w_j^{k+1} + \sum_{j=l+1}^d x_i^j w_j^k\right)$. Hence, we can use (2.32) in combination with Algorithm 3 to numerically compute a solution of the LASSO problem (2.22).

2.8 Deep learning

So far, all supervised machine learning models that we have considered were regression models that were either linear or nonlinear in the input arguments x but linear in the weights w. We now want to shift our focus to a particular class of models that are nonlinear in both the input arguments and the weights: so-called *deep neural networks*.

Deep neural networks

In mathematical terms, a deep neural network is simply a function that is a composition of simple, parametrised functions and potentially many of them, i.e.

$$f_w(x) := \varphi_L\left(\varphi_{L-1}\left(\cdots \varphi_1\left(\varphi_0(x, w_1), w_2\right)\cdots, w_{L-1}\right), w_L\right). \tag{2.33}$$

Here $\{\varphi_l\}_{l=1}^L$ is a family of L so-called *activation functions* that are parametrised with weights $w := \{w_l\}_{l=1}^L$. To be more precise, (2.33) is a deep neural network with L layers.
Typical activation functions are affine-linear transformations, i.e.

$$\varphi(x, W, b) := W^\top x + b\,.$$

In terms of terminology, $W \in \mathbb{R}^{n\times m}$ is a *weight* matrix and $b \in \mathbb{R}^{m\times 1}$ is the *bias* vector. This way $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ maps inputs $x \in \mathbb{R}^n$ onto outputs $\varphi(x) \in \mathbb{R}^m$.
An example for a nonlinear activation function is the Heaviside function

$$\varphi(x) = H(x) := \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}.$$

Together with an affine-linear transformation, Heaviside functions paved the way for the definition of the *perceptron*, a simplified model loosely based on a neuron.

Example 2.4 (Perceptron)**.** An artificial neuron known as *perceptron* is defined as the nonlinear activation function

$$\varphi(x, w, b) := H(w^\top x + b) = \begin{cases} 1 & w^\top x \geq -b \\ 0 & w^\top x < -b \end{cases},$$

for a weight vector $w \in \mathbb{R}^n$ and a bias $b \in \mathbb{R}$. Note that this function is a composition and itself could already be seen as two-layer neural network of the form

$$f(x, w, b) = \varphi_1(\varphi_0(x, w, b))$$

with $\varphi_1(x) := H(x)$ and $\varphi_0(x, w, b) := w^\top x + b$.

Another popular activation function is the so-called *rectifier* $\varphi : \mathbb{R} \to \mathbb{R}_{\geq 0}$, defined as

$$\varphi(x) := \max(0, x)\,.$$

Simple activation functions such as the Heaviside function or the rectifier can easily be extended to activation functions operating on vectors $x \in \mathbb{R}^n$ by defining $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ with

$$\varphi(x) := (\varphi(x_1), \varphi(x_2), \ldots, \varphi(x_n))^\top\,.$$

We will often abuse notation and use the same notation for the scalar and vector-valued variants of such simple activation functions. When we write a vector-valued rectifier $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ as

$$\varphi(x) = \max(0, x)\,,$$

we mean $\varphi(x) = (\max(0, x_1), \max(0, x_2), \ldots, \max(0, x_n))$ but write $\max(x, 0)$ for the sake of notational simplicity.

In combination with the affine-linear transformations they form what is known as *Rectified Linear Unit (ReLU)*, i.e.

$$\varphi(x, W, b) := \max\left(0,\, W^\top x + b\right)\,,$$

where we have used the simplified notation for the vector-valued rectifier.

Example 2.5 (ReLU neural networks)**.** Based on the previous considerations, an L-layer neural network with ReLU activation functions has the form

$$\varphi(x, w) = \max\left(0,\, W_L^\top \max\left(0,\, W_{L-1}^\top \max\left(\ldots \max\left(0, W_1^\top x + b_1\right)\ldots\right) + b_{L-1}\right) + b_L\right)\,,$$

or

$$\varphi(x, w) = \max\left(0, W_L^\top x^L + b_L\right)\,,$$

for

$$x^l := \begin{cases} \max\left(0, W_l^\top x^{l-1} + b_l\right) & l \in \{2, \ldots, L\} \\ \max\left(0, W_1^\top x + b_1\right) & l = 1 \end{cases}\,.$$

Here the parameters w are defined as the collection of all weight matrices and bias vectors, i.e. $w = \left\{\{W_l\}_{l=1}^L, \{b_l\}_{l=1}^L\right\}$.

One last notable activation function that we want to discuss is the *softmax* activation function, which is defined as $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ with

$$\varphi(x_1, \ldots, x_n) = \left(\frac{\exp(x_1)}{\sum_{j=1}^n \exp(x_j)}, \ldots, \frac{\exp(x_n)}{\sum_{j=1}^n \exp(x_j)}\right)^\top\,.$$

This activation function is extremely useful as it allows us to map arguments onto the (probability) simplex, i.e. $\varphi(x_1, \ldots, x_n)_i \geq 0$ for all $i \in \{1, \ldots, n\}$ and $\sum_{i=1} \varphi(x_1, \ldots, x_n)_i = 1$. The name stems from the fact that this activation function can be seen as a smooth approximation of the argmax function.

A general nonlinear regression model with deep neural networks then simply reads as minimising the empirical risk (2.20) of the form

$$w_t = \operatorname*{arg\,min}_{w\in\mathbb{R}^n} \left\{ \frac{1}{s} \sum_{i=1}^{s} \ell_i(f_w(x_i), y_i) \right\}, \tag{2.34}$$

for a family of loss functions $\{\ell_i\}_{i=1}^s$ with $\ell_i : \mathbb{R}^n \to \mathbb{R}^m$ for each $i \in \{1, \dots, s\}$. As for the previous examples we can for instance choose $\ell_i(z) := \frac{1}{2}|z - y_i|^2$ in order to minimise the mean-squared error, but with a deep neural network as the model function, i.e.

$$\begin{aligned} w_t &= \operatorname*{arg\,min}_{w\in\mathbb{R}^n} \left\{ \frac{1}{2s} \sum_{i=1}^{s} |f_w(x_i) - y_i|^2 \right\}, \\ &= \operatorname*{arg\,min}_{w\in\mathbb{R}^n} \left\{ \frac{1}{2s} \sum_{i=1}^{s} |x_i^L - y_i|^2 \right\}, \end{aligned}$$

for

$$x_i^l = \varphi_l(x_i^{l-1}, w^l), \qquad \text{for} \qquad \forall i \in \{1, \dots, s\}, \forall l \in \{1, \dots, L\},$$

assuming $x_i^0 = x_i$ for all $i \in \{1, \dots, s\}$. Other choices such as the mean absolute error can certainly be used as well, and some will also be explored next semester. An important question that we want to address in the following section is the optimisation of the parameters w_t, also referred to as training or empirical risk minimisation as discussed in Section 2.5.

2.8.1 Training deep learning models

In the previous section we learned that training the parameters of a deep neural network is equivalent to minimising objective functions of the form (2.34). In principle, there is no reason why we cannot use algorithms such as gradient descent (Algorithm 1) and modifications such as stochastic gradient descent or subgradient descent that we will learn more about in the next semester. However, the key difference is that the objective in (2.34) is in general not a convex function anymore. This means that the convergence results that we have derived in earlier sections do not apply. This does not mean that there do not exist other conditions that could guarantee convergence, but this is beyond the scope of this module. We nevertheless will apply the same algorithms, keeping in mind that we do not necessarily have convergence guarantees as in the convex setting.

If we have a differentiable deep neural network with differentiable activation functions, we can (try to) train a deep neural network simply by minimising (2.34) via gradient descent, i.e. Algorithm 1. In order to do so, we are required to compute the gradient of the objective function w.r.t. the network parameters w. This can be done via backpropagation, which is a fancy name for applying the chain rule to the particular network architecture.
In the following, we focus on architectures of the form

$$x_i^l = \sigma(z_i^l), \tag{2.35a}$$

$$z_i^l = W_l^\top x_i^{l-1} + b_l, \tag{2.35b}$$

for $x_i^0 = x_i$, nonlinear activation functions σ and all $i \in \{1, \dots, s\}$ and $l \in \{1, \dots, L\}$ and the empirical risk minimisation problem for the empirical risk function

$$L(W_1, \dots, W_L, b_1, \dots, b_L) = \frac{1}{s} \sum_{i=1}^{s} \ell(x_i^L, y_i). \tag{2.36}$$

Algorithm 4 Backpropagation

Specify: Activation function σ, samples $\{(x_i, y_i)\}_{i=1}^s$, weight and bias dimensions, and no. of layers L

Iterate:

for $i = 1, \dots, s$ **do**
  for $l = 1, \dots, L$ **do**
    Forward pass: compute $z_i^l = W_l^\top x_i^{l-1} + b_l$
    Forward pass: compute $x_i^l = \sigma(z_i^l)$
  end for
end for
for $i = 1, \dots, s$ **do**
  for $l = L, \dots, 1$ **do**
    Backward pass: compute $\delta_i^l = \begin{cases} \sigma'(z_i^L) \odot \frac{1}{s} \nabla_1 \ell(x_i^L, y_i) & l = L \\ \sigma'(z_i^l) \odot W_{l+1} \delta^{l+1} & l \in \{1, \dots, L-1\} \end{cases}$
  end for
end for
Partial derivatives: compute $\frac{\partial L}{\partial b_j^l} = \delta_j^l$, for all $j \in \{1, \dots, n_l\}$
Partial derivatives: compute $\frac{\partial L}{\partial w_{jk}^l} = \delta_j^l x_k^{l-1}$, for all $j \in \{1, \dots, n_l\}$ and $k \in \{1, \dots, n_{l-1}\}$.

return $\{W_l\}_{l=1}^L$ and $\{b_l\}_{l=1}^L$.

Other backpropagation rules for more general architectures can be derived in similar fashion. The following lemma characterises the partial derivatives of the empirical risk function with respect to the parameters.

Lemma 2.5. *When we define the quantity*

$$\delta_j^l := \frac{\partial L}{\partial x_j^l}$$

for $j \in \{1, \dots, n_l\}$ and $l \in \{2, \dots, L\}$, we can show that the partial derivatives of L with respect to the weights and biases satisfy

$$\begin{aligned} \delta_i^l &= \begin{cases} \sigma'(z_i^L) \odot \frac{1}{s} \nabla_1 \ell(x_i^L, y_i) & l = L \\ \sigma'(z_i^l) \odot W_{l+1} \delta^{l+1} & l \in \{1, \dots, L-1\} \end{cases}, \\ \frac{\partial L}{\partial b_j^l} &= \delta_j^l \,, \\ \frac{\partial L}{\partial w_{jk}^l} &= \delta_j^l x_k^{l-1} \,. \end{aligned}$$

Here $\odot$ denotes the Hadamard product, which is simply a pointwise multiplication, and $\nabla_1 \ell$ is the gradient of ℓ with respect to the first argument.

Theorem 2.4 (Backpropagation)**.** *The gradient of the function* (2.36) *subject to the neural network constraint* (2.35) *with respect to the parameters $\{W_l\}_{l=1}^L$ and $\{b_l\}_{l=1}^L$ can be computed via the backpropagation Algorithm 4.*

As a consequence, we can train the network parameters via Algorithm 1 aka gradient descent where we use the gradient that we have computed with Algorithm 4.

2.9 Classification

For the remainder of this chapter we move on from regression problems to classification problems. Classification is the task of associating a certain class from a number of pre-defined classes to the input of a function. Suppose we have a set of s input and output samples $\{(x_i, y_i)\}_{i=1}^s$, then the goal of classification is to find a function $f : \mathbb{R}^d \to \{C_1, C_2, \dots, C_n\}$ that approximately satisfies

$$f(x_i) \approx y_i \,,$$

for all $i \in \{1, \dots, s\}$. You may say: hold on, this looks exactly like the general supervised learning formulation that we have introduced in (2.1) and this is correct! The only difference compared to supervised regression is that the function f no longer maps onto continuous values, but onto a discrete set of values $\{C_1, C_2, \dots, C_n\}$. Here $\{C_j\}_{j=1}^n$ are the so-called *class labels* that are numerical values associated with the n individual classes. Note that each y_i has to take on one of those values as well, i.e. $y_i \in \{C_1, \dots, C_n\}$ for all $i \in \{1, \dots, s\}$. If there are only two classes to map to, i.e. the range of the function is $\{C_1, C_2\}$, then we speak of a *binary classification* problem. For more than two classes we speak of a *multiclass classification* problem. In the following, we introduce a very basic classification method known as nearest neighbour classification, discuss its limitations and then continue to introduce other classification models such as logistic regression and support vector machines.

2.9.1 Nearest neighbour classification

One of the most simple classification ideas is to classify a new data sample x based on classes of the K-nearest neighbours of that sample in the training set. We can do this by assigning a probability to the unknown output label based on the labels of the K nearest neighbours. This probability is of the form

$$\rho(y = c \,|\, x, K) := \frac{1}{K} \sum_{l \in \mathcal{N}_K(x)} \iota(y_l = c) \,, \tag{2.37}$$

with ι defined as

$$\iota(z) := \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{if } z \text{ is false} \end{cases} . \tag{2.38}$$

Here $\mathcal{N}_K$ denotes the neighbourhood of x, which includes the K nearest neighbours of x. There are obviously many different ways of measuring distances between data points. One example is to simply measure the Euclidean distances between data points. In that case the neighbourhood can be defined as $\mathcal{N}_K(x) := \big\{x_{i(1)}, \dots, x_{i(K)} \,\big|\, \|x - x_{i(j)}\| \leq \|x - x_l\| \,, \forall\, l \in \{1, \dots, s\} \setminus \{i(1), \dots, i(K)\}\big\}$.

The definition (2.38) assigns a probability to the event that a label y equals a class label $c \in \{C_0, \dots, C_n\}$. This is done by computing the averages among the K nearest neighbours with identical class label. You can find particular examples in the corresponding video lecture that accompanies these lecture notes. Once we have computed all probabilities for the different class labels, we assign the class label with the highest probability to the output of our classifier f, i.e.

$$f(x) := \underset{c \in \{C_0, C_1, \dots, C_n\}}{\arg\max} \; \rho(y = c \,|\, x, K) \,. \tag{2.39}$$

The entire strategy is known as the *K-nearest neighbours classification*. The number of neighbours K is a hyperparameter that has to be determined with model selection strategies such a cross validation.

In the following we want to shed some light on why nearest neighbour classification works well for lower dimensional problems (small d) but fails to deliver meaningful results when the dimension d increases.

The curse of dimensionality

Nearest neighbour classification is a very simple classification strategy that, unfortunately, falls victim to the so-called *curse of dimensionality*. In the context of supervised machine learning, the curse of dimensionality usually takes the following form.

(a) "Generalising correctly becomes exponentially harder as the dimensionality grows because fixed-size training sets cover a dwindling fraction of the input space."

(b) In high-dimensions, data-points are far from each other. Consequently, "as the dimensionality increases, the choice of nearest neighbour becomes effectively random."

These quotes are taken from Pedro Domingos review article "A few useful things to know about machine learning". In the following, we want to dissect what those claims mean mathematically. For the first claim, imagine that we have m data inputs $\{x_i\}_{i=1}^m$ that all lie in the d-dimensional unit cube $[0,1]^d$ with a total volume of $1^d = 1$. Now we consider a sub-cube $[a, a+r]^d \subset [0,1]^d$ with $0 \leq a$ and $a+r \leq 1$ with volume $r^d \leq 1$. This sub-cube can be seen as our training set, containing s training samples. The question we want to address here is the following: how large does this cube have to be in order to cover a certain fraction α of the m data samples (in expectation)? In other words: how do we need to choose the length r of the sub-cube such that $\alpha = r^d$? The straight-forward answer to this question is

$$r = \sqrt[d]{\alpha}\,.$$

If we are in a $d = 10$ dimensional space for example, covering only $\alpha = 1\%$ of the data already requires a cube with length $r \approx 0.63$. To cover $\alpha = 10\%$ of the data in the overall cube, a length of $r \approx 0.8$ is required. In other words, for a fixed sub-cube with fixed length $r < 1$ increasing the dimension d dramatically reduces the fraction of data samples that the sub-cube covers.

For the second claim, we consider the d-dimensional unit cube again, but this time our sub-cube is located around the centre point $(\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2})$, i.e. we have $[(1-r)/2, (1+r)/2]^d \subset [0,1]^d$ with $0 \leq r \leq 1$. Suppose m input samples $\{x_i\}_{i=1}^s$ are uniformly distributed in the cube $[0,1]^d$. What is the chance that a random sample is in the overall cube $[0,1]^d$ but not in the sub-cube $[(1-r)/2, (1+r)/2]^d$? This chance is simply $1-r^d$. Considering m i.i.d. samples, the chance that none of these m samples is in the sub-cube then becomes

$$\left(1 - r^d\right)^m\,.$$

For a fixed probability ρ, we can solve this equation for r, i.e.

$$r = \sqrt[d]{1 - \sqrt[m]{\rho}}\,.$$

To have a probability of 50% (which means $\rho = 0.5$) that none of $m = 500$ samples in a $d = 10$ dimensional space is in the sub-cube, the length of the sub-cube only has to be $r \approx 0.52$. Again, keeping r fixed and increasing the dimension d will quickly increase the probability ρ to values close to 100%, rendering the concept of nearest neighbours useless in higher dimensions.

All these considerations tell us that the K-nearest neighbours classification strategy suffers from the curse of dimensionality, which is why we have to consider other classification strategies that do not rely on the concept of neighbours. In the following sections we will consider three alternative classification strategies.

2.9.2 Logistic regression

One could think of solving classification problems with the same tools that we have used for tackling regression problems, i.e. mean-squared-error regression. After all, the only difference is that the output of the trained prediction function $f(x, w)$ is supposed to be discrete and not continuous, which we could achieve by thresholding the function output. However, as you find out in the weekly lecture videos, this strategy is not working in practice, since the mean-squared error is not really related to the objective of minimising the number of misclassified samples. Starting with the concept of binary classification, it seems reasonable to transform the prediction $f(x, w)$ into a probability. In order, to do so, we consider $\sigma(f(x, w))$ instead of $f(x, w)$, where $\sigma : (-\infty, \infty) \to [0, 1]$ is the so-called *logistic function* defined as

$$\sigma(z) := \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z} \,. \tag{2.40}$$

We can then define the following probabilities for the events that the output $f(x, w)$ belongs the class with class label zero or the class with class label one:

$$\rho(1|x) := \sigma(f(x, w)) \,, \tag{2.41a}$$
$$\rho(0|x) := 1 - \sigma(f(x, w)) \,. \tag{2.41b}$$

From the definition of σ we instantly observe $\rho(1|x) \geq 0$, $\rho(0|x) \geq 0$ and $\rho(1|x) + \rho(0|x) = 1$. Hence, when we speak of ρ we can indeed speak of a probability. Now assume we have s pairs of samples $\{(x_i, y_i)\}_{i=1}^s$ where $y_i \in \{0, 1\}$ for all $i \in \{1, \ldots, s\}$ that are iid and follow the probability density function that we have just defined in (2.41). The corresponding likelihood then reads

$$\rho\left(y \mid X, w\right) = \prod_{i=1}^{s} \rho(y_i|x_i) \,, \tag{2.42}$$

where X and y are abbreviations of the matrix and vector that we obtain from the samples $\{y_i\}_{i=1}^s$ and $\{x_i\}_{i=1}^s$. Note that we can rewrite (2.42) as

$$\begin{aligned} \rho\left(y \mid X, w\right) &= \prod_{\{i \,|\, y_i = 1\}} \rho(y_i = 1|x_i) \prod_{\{i \,|\, y_i = 0\}} \rho(y_i = 0|x_i) \,, \\ &= \prod_{i=1}^{s} \sigma(f(x_i, w))^{y_i} \left(1 - \sigma(f(x_i, w)\right)^{1 - y_i} \,. \end{aligned}$$

As in the regression case with the normal distribution, we can obtain parameters $\hat{w}$ that maximise the likelihood (2.42) by minimising the negative log-likelihood, i.e.

$$\begin{aligned} \hat{w} &= \arg\min_w \left\{ -\log\left(\rho\left(y \mid X, w\right)\right) \right\} \,, \\ &= \arg\min_w \left\{ -\log\left(\prod_{i=1}^{s} \sigma(f(x_i, w))^{y_i} \left(1 - \sigma(f(x_i, w))\right)^{1 - y_i} \right) \right\} \,, \\ &= \arg\min_w \left\{ -\sum_{i=1}^{s} \left[y_i \log\left(\sigma(f(x_i, w))\right) + (1 - y_i) \log\left((1 - \sigma(f(x_i, w)))\right) \right] \right\} \,, \\ &= \arg\min_w \left\{ \sum_{i=1}^{s} \left[\log\left(1 + \exp\left(f(x_i, w)\right)\right) - y_i f(x_i, w) \right] \right\} \,. \end{aligned}$$

This alternative form of regression is known as *logistic regression* as it is based on the logistic function (2.40). We can choose any model function f – linear or polynomial basis function, or even a neural network – that we like, as long as it maps onto the real numbers. To determine a unique minimiser, a model linear with respect to the weights w has its advantages, as we will discuss later. Before we discuss how to compute an argument that minimises this expression numerically, we want to discuss how to extend logistic regression to classification problems with more than two classes first.

2.9.3 Multinomial logistic regression

In order to derive a logistic regression problem that can deal with more than two classes, we need to come up with a probability model that can associate the highest probability to the label that corresponds to the correct class. We can do this with the help of the so-called *softmax*-function. Assume for $K > 2$ classes that we have a model function $f(x, w_1, \dots, w_K)$ that depends on multiple weight vectors $\{w_k\}_{k=1}^K$, and more importantly, that maps onto a K-dimensional vector rather than a scalar output. Given such a function, we compose it with the softmax function $\sigma : \mathbb{R}^K \to [0,1]^K$ that is defined as

$$\sigma(z_1, z_2, \dots, z_K)_k := \frac{\exp(z_k)}{\sum_{j=1}^K \exp(z_j)}\,, \qquad \forall\, k \in \{1, \dots, K\}\,.$$

We can then associate a probability for each class based on this composition, i.e.

$$\rho(y_i = k \,|\, x_i, w_1, \dots, w_K) := \sigma\left(f(x_i, w_1, \dots, w_K)\right)_k = \frac{\exp(f(x_i, w_1, \dots, w_K)_k)}{\sum_{j=1}^K \exp(f(x_i, w_1, \dots, w_K)_j)}\,,$$

for all $k \in \{1, \dots, K\}$. By definition of the softmax-function, we observe $\rho(y_i = k \,|\, x_i, w_1, \dots, w_K) \geq 0$ for all $k \in \{1, \dots, K\}$ as well as $\sum_{k=1}^K \rho(y_i = k \,|\, x_i, w_1, \dots, w_K) = 1$; hence, we can talk of ρ as a probability. We can then proceed as in the binary logistic regression case and define a likelihood for s data samples $\{(x_i, y_i)\}_{i=1}^s$ via

$$\rho(\hat{y} = y \,|\, X, W) := \prod_{i=1}^s \rho(\hat{y}_i = y_i \,|\, x_i, w_1, \dots, w_K)\,,$$

for the short-hand notations $y = (y_1, \dots, y_s)^\top$, $X = \begin{pmatrix} x_1 & \dots & x_s \end{pmatrix}^\top$ and $W = \begin{pmatrix} w_1 & w_2 & \dots & w_K \end{pmatrix}$. We can simplify this likelihood to

$$\rho(\hat{y} = y \,|\, X, W) = \prod_{\{i \,|\, y_i = 1\}} \rho(\hat{y}_i = 1 \,|\, x_i, w_1, \dots, w_K) \cdots \prod_{\{i \,|\, y_i = K\}} \rho(\hat{y}_i = K \,|\, x_i, w_1, \dots, w_K)\,;$$

We can use the indicator function $\mathbf{1}_{y_i = k}$ defined as

$$\mathbf{1}_{y_i = k} := \begin{cases} 1 & y_i = k \\ 0 & \text{otherwise} \end{cases}$$

to further simplify the likelihood to

$$\rho(\hat{y} = y \,|\, X, W) := \prod_{i=1}^s \prod_{k=1}^K \rho(\hat{y}_i = k \,|\, x_i, w_1, \dots, w_K)^{\mathbf{1}_{y_i = k}}\,.$$

As before, we can maximise this likelihood by choosing the parameters W such that they minimise the negative log-likelihood, i.e.

$$\begin{aligned}
\hat{W} &= \arg\min_W -\log\left(\rho(\hat{y} = y \,|\, X, W)\right) \,, \\
&= \arg\min_W -\log\left(\prod_{i=1}^{s}\prod_{j=1}^{K} \rho(\hat{y}_i = k \,|\, x_i, w_1, \ldots, w_K)^{\mathbf{1}_{y_i=k}}\right) \,, \\
&= \arg\min_W -\sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} \log\left(\rho(\hat{y}_i = k \,|\, x_i, w_1, \ldots, w_K)\right) \,, \\
&= \arg\min_W -\sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} \log\left(\frac{\exp(f(x_i, w_1, \ldots, w_K)_k)}{\sum_{j=1}^{K}\exp(f(x_i, w_1, \ldots, w_K)_j)}\right) \,, \\
&= \arg\min_W \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} \left(\log\left(\sum_{j=1}^{K}\exp(f(x_i, w_1, \ldots, w_K)_j)\right) - f(x_i, w_1, \ldots, w_K)_k\right) \,, \\
&= \arg\min_W \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} \log\left(\sum_{j=1}^{K}\exp(f(x_i, w_1, \ldots, w_K)_j)\right) - \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} f(x_i, w_1, \ldots, w_K)_k \,, \\
&= \arg\min_W \sum_{i=1}^{s} \log\left(\sum_{k=1}^{K}\exp(f(x_i, w_1, \ldots, w_K)_k)\right) - \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} f(x_i, w_1, \ldots, w_K)_k \,.
\end{aligned}$$

Hence, with the more compact notation $f(x, W)$ we can estimate optimal parameters $\hat{W}$ via

$$\hat{W} = \arg\min_W \sum_{i=1}^{s} \log\left(\sum_{k=1}^{K}\exp(f(x_i, W)_k)\right) - \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k} f(x_i, W)_k \,.$$

A typical choice for f based on polynomial augmentation would be

$$\begin{aligned}
f(x, W) &= \begin{pmatrix}\langle\phi(x), w_1\rangle & \langle\phi(x), w_2\rangle & \ldots & \langle\phi(x), w_K\rangle\end{pmatrix} \\
&= \phi(x)^\top W \,.
\end{aligned}$$

If we store all input samples $\{x_i\}_{i=1}^s$ in a matrix $X = \begin{pmatrix}x_1 & \ldots & x_s\end{pmatrix}^\top$, we can write

$$f(X, W) = \Phi(X)W$$

in matrix form. The *multinomial logistic regression* problem then reads

$$\hat{W} = \arg\min_{W\in\mathbb{R}^{(1+d)\times K}} \sum_{i=1}^{s} \log\left(\sum_{k=1}^{K}\exp(\langle\phi(x_i), w_k\rangle)\right) - \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k}\langle\phi(x_i), w_k\rangle \,,$$

or

$$\hat{W} = \arg\min_{W\in\mathbb{R}^{(1+d)\times K}} \sum_{i=1}^{s} \log\left(\sum_{k=1}^{K}\exp((\Phi(X)W)_{ik})\right) - \sum_{i=1}^{s}\sum_{k=1}^{K} \mathbf{1}_{y_i=k}(\Phi(X)W)_{ik}\rangle$$

in matrix form. The key question that remains for both the binary and the multinomial logistic regression problem is: how do we solve these minimisation problems computationally?

We will focus on the binary logistic regression case with polynomial data model $f(x, w) = \langle \phi(x), w \rangle$; the multinomial logistic regression case can be covered in almost identical fashion, just with different computations. The first thing that we observe is that the objective- (or cost-)function

$$L(w) := \sum_{i=1}^{s} \left[\log\left(1 + \exp\left(\langle \phi(x_i), w \rangle\right)\right) - y_i \langle \phi(x_i), w \rangle \right] \tag{2.43}$$

is differentiable. So can we maybe compute the gradient, set the gradient to zero and solve for the weights w as we did for the mean-squared error regression? Computing the partial derivative of L with respect to w_l, for $l \in \{0, \ldots, d\}$, yields

$$\begin{aligned} \frac{\partial L}{\partial w_l} &= \sum_{i=1}^{s} \phi(x_i)_l \left(\sigma\left(\langle \phi(x_i), w \rangle\right) - y_i\right) , \\ &= \sum_{i=1}^{s} \Phi(X)_{il} \left(\sigma\left((\Phi(X)w)_i\right) - y_i\right) . \end{aligned}$$

Here σ denotes the logistic function as defined in (2.40). Hence, the entire gradient of L reads

$$\nabla L(w) = \Phi(X)^\top \left(\sigma(\Phi(X)w) - y\right)$$

in column-vector form. Here, $\sigma(\Phi(X)w)$ is short-hand notation for applying the logistic function point-wise to every component of the vector $\Phi(X)w$. Setting the gradient to zero and solving it for the corresponding argument $\hat{w}$ would therefore require the solution of the equation

$$0 = \Phi(X)^\top \left(\sigma(\Phi(X)\hat{w}) - y\right) . \tag{2.44}$$

Because of the non-linearity of σ, we cannot simply solve (2.44) for $\hat{w}$ and a fixed but arbitrary matrix $\Phi(X)$. Having computed the gradient, we can, however, try to approximate a solution of

$$\hat{w} = \underset{w \in \mathbb{R}^{1+d}}{\arg\min} \{L(w)\} \tag{2.45}$$

via gradient descent, as defined in Algorithm 1. For this particular choice of cost function L, the gradient descent iterate becomes

$$w^{k+1} = w^k - \tau\, \Phi(X)^\top \Big(\sigma(\Phi(X)w^k) - y\Big) .$$

The question that we need to address is whether gradient descent will convergence to a solution of (2.45). Note that in order to apply Theorem 2.3, we only need to verify convexity of L and $1/\tau$-smoothness. With Corollary 2.2 we can verify that the logistic regression function is convex.

Lemma 2.6 (Convexity of the binary logistic regression problem)**.** *The function $L : \mathbb{R}^{1+d} \to \mathbb{R}$ as defined in Equation* (2.43) *is convex.*

Proof. We basically only need to show that the function $f(z) := \log(1 + \exp(z))$ is convex; then we could conclude that L is a sum of convex functions (and therefore convex itself), as linear functions are convex and since compositions of convex functions and linear functions are convex.

We verify that f is convex by showing that $f''(z) \geq 0$ for all $z \in \mathbb{R}$. We had already computed the first derivative that reads

$$f'(z) = \frac{1}{1+\exp(-z)} = \sigma(z)\,;$$

It can be easily verified that the second derivative reads

$$f''(z) = \sigma(z)\,(1-\sigma(z))\ .$$

Since $\sigma(z) \in [0,1]$ for all $z \in \mathbb{R}$, we immediately see that $f''(z) \geq 0$ for all z. Hence, f is convex and as a consequence, L is convex. □

In order to converge successfully to a minimiser of the logistic regression problem with gradient descent, we only need to specify a step-size $\tau > 0$ that is sufficiently small so that the objective values decrease monotonically (if such a step-size exists). Without proof, we use that the logistic function σ is a $\frac{1}{4}$-Lipschitz continuous function, i.e.

$$|\sigma(x) - \sigma(y)| \leq \frac{1}{4}|x-y|\,,$$

for all $x, y \in \mathbb{R}$. Then we can conclude that the gradient $\nabla L(w)$ is $1/\tau$-Lipschitz continuous, i.e.

$$\begin{aligned}
\|X^\top(\sigma(Xw)-y) - X^\top(\sigma(Xv)-y)\| &= \|X^\top(\sigma(Xw)-\sigma(Xv))\|\,,\\
&\leq \|X\|\|\sigma(Xw)-\sigma(Xv)\|\,,\\
&\leq \frac{\|X\|}{4}\|Xw-Xv\|\,,\\
&\leq \frac{\|X\|^2}{4}\|w-y\|\,,
\end{aligned}$$

for $\tau = 4/\|X\|^2$ and all $w, v \in \mathbb{R}^{1+d}$. Hence, gradient descent is guaranteed to converge to a global minimum of L for $\tau < 4/\|X\|^2$ as a consequence of Lemma 2.3 and Theorem 2.3.

2.9.4 Support-vector machines (SVMs)

One limitation of logistic regression is that the hyperplane that spans the decision boundary is not necessarily optimal in the sense that it maximises the distance between the closest data points on each side of the decision boundary. This feature can, however, be achieved in the context of binary classification with *Support Vector Machines (SVMs)* that utilise a different data model. For a more detailed motivation we refer to the lecture videos & slides. For a set of data points $\{(x_i, y_i)\}_{i=1}^s$, with $y_i \in \{-1, 1\}$ for all $i \in \{1, \ldots, s\}$, and a linear model function $f(x, w)$ with parameters $w \in \mathbb{R}^{1+d}$, the key idea is to maximise the distance r of the closest data points to the hyper-plane, but to ensure that the each data point ends up on the correct side of the decision boundary at the same time. Mathematically, for a linear model $f(x, w) = \langle \phi(x), w\rangle$ this distance r can be characterised via

$$r = \frac{f(x,w)}{\|w\|}\,,$$

while ensuring that each data point is on the correct side of the decision boundary can mathematically be described as the constraint

$$y_i\, f(x_i, w) - 1 \geq 0\,,$$

for all $i \in \{1, \dots, s\}$. In order to maximise r, we can minimise $\|w\|$ (or $\|w\|^2$) subject to the previous constraint, i.e.

$$\min_{w_0, w_2, \dots, w_d} \|w\|^2 \qquad \text{subject to} \qquad y_i \, f(x_i, w) - 1 \geq 0 \,, \quad \forall i \in \{1, \dots, s\} \,. \tag{2.46}$$

If the data points cannot be separated linearly, Problem (2.46) doesn't have a solution. To overcome this limitation, we can relax Problem (2.46) to

$$\min_{w \in \mathbb{R}^{1+d}} \sum_{i=1}^{s} \max\left(0, 1 - y_i \, \langle x_i, w \rangle\right) + \frac{\alpha}{2} \|w\|^2 \,, \tag{2.47}$$

for $w = \begin{pmatrix} w_0 & w_2 & \dots & w_d \end{pmatrix}^\top$ and a balancing (or regularisation) parameter $\alpha > 0$. The solution to Problem (2.47) is known as the *soft-margin SVM*; the solution to Problem (2.46) as *hard-margin SVM*. In the following we want to discuss how to simplify (2.47) to make it computationally more tractable.

We follow a similar trick as in Section 2.7.2 where we have reformulated the absolute value function in terms of a dual variable. We can do the same trick for the ramp function $\max(0, z)$, i.e. we can write $\max(0, z)$ as

$$\max(0, z) = \max_{\lambda \in [0,1]} \lambda z \,.$$

Replacing the ramp function in (2.47) with this dual characterisation yields the min-max problem

$$\min_{w \in \mathbb{R}^{1+d}} \left\{ \max_{\lambda \in [0,1]^s} \sum_{i=1}^{s} \lambda_i \left(1 - y_i \langle x_i, w \rangle\right) + \frac{\alpha}{2} \|w\|^2 \right\} . \tag{2.48}$$

Note that the underlying function $L(w, \lambda)$ defined as

$$L(w, \lambda) := \sum_{i=1}^{s} \lambda_i \left(1 - y_i \langle x_i, w \rangle\right) + \frac{\alpha}{2} \|w\|^2 - \chi_{[0,1]^s}(\lambda)$$

with

$$\chi_{[0,1]^s}(\lambda) = \begin{cases} 0 & \forall i \in \{1, \dots, s\} \,:\, \lambda_i \in [0, 1] \\ \infty & \exists i \in \{1, \dots, s\} \,:\, \lambda_i \notin [0, 1] \end{cases} ,$$

is convex in the first argument (for fixed second argument) and concave in the second argument (for fixed first argument), where convexity and concavety are defined as in Definition 2.2, respectively Definition 2.3. This statement is left as an exercise to the curious reader. The following theorem states that it makes no difference for such functions whether one first maximises and then minimises the function, or if one first minimises and then maximises the function.

Theorem 2.5 (Minimax Theorem, von Neumann 1928)**.** *Let $X \subset \mathbb{R}^m$ and $Y \subset \mathbb{R}^n$ be compact, convex sets. If $f : X \times Y \to \mathbb{R}$ is a continuous function that is convex-concave, i.e.*

$$f(\cdot, y) : X \to \mathbb{R} \text{ is convex for fixed } y \,,$$
$$f(x, \cdot) : Y \to \mathbb{R} \text{ is concave for fixed } x \,.$$

Then the max-min inequality is an equality, i.e.

$$\min_{x \in X} \max_{y \in Y} f(x, y) = \max_{y \in Y} \min_{x \in X} f(x, y) \,.$$

Thanks to Theorem 2.5, we can replace (2.48) with the equivalent problem

$$\max_{\lambda\in[0,1]^s}\left\{\min_{w\in\mathbb{R}^{1+d}}\sum_{i=1}^{s}\lambda_i\left(1-y_i\langle x_i,w\rangle\right)+\frac{\alpha}{2}\|w\|^2\right\}. \tag{2.49}$$

What is the advantage of (2.49) over (2.48), you may ask? The advantage is that (2.48) can be solved more easily, as the inner problem becomes differentiable. The gradient of L with respect to w reads

$$\nabla_w L(w,\lambda) = -\sum_{i=1}^{s}\lambda_i y_i x_i + \alpha w\,.$$

Computing $\hat{w}$ with $\nabla_w L(\hat{w},\lambda)=0$ yields

$$\hat{w} = \frac{1}{\alpha}\sum_{i=1}^{s}\lambda_i y_i x_i\,. \tag{2.50}$$

Inserting $\hat{w}$ into $L(w,\lambda)$ then turns (2.49) into

$$\max_{\lambda\in[0,1]^s}\left\{\langle\lambda,\mathbf{1}\rangle-\frac{1}{2\alpha}\|X^\top Y\lambda\|^2\right\}. \tag{2.51}$$

Here Y is short-hand notation for the $s\times s$ diagonal matrix that contains the elements of the vector y on its diagonal, i.e.

$$Y=\operatorname{diag}(y):=\begin{pmatrix} y_1 & 0 & 0 & \cdots & 0\\ 0 & y_2 & 0 & \cdots & 0\\ \vdots & & \ddots & & \vdots\\ 0 & \cdots & 0 & \cdots & y_s\end{pmatrix},$$

and $\mathbf{1}$ is short-hand-notation for the s-dimensional vector of ones, i.e. $\mathbf{1}=\begin{pmatrix}1 & 1 & 1 & \dots & 1\end{pmatrix}^\top\in\mathbb{R}^s$. We can reformulate (2.51) equivalently as minimisation problem; hence, computing the argument $\hat{\lambda}$ that minimises this expression is

$$\hat{\lambda}=\operatorname*{arg\,min}_{\lambda\in[0,1]}\left\{\frac{1}{2\alpha}\|X^\top Y\lambda\|^2-\langle\lambda,\mathbf{1}\rangle\right\}. \tag{2.52}$$

This convex problem can be solved computationally with various algorithms; examples include the proximal gradient method as described in Algorithm 2 or the coordinate descent method summarised in Algorithm 3. Both methods utilise proximal maps, which in this case reduce to the orthogonal projection onto the convex set $[0,1]$, i.e.

$$\left(\operatorname{prox}_{\chi_{[0,1]^s}}(z)\right)_i=\min(1,\max(0,z_i))\,,$$

for all $i\in\{1,\dots,s\}$. Note that by solving (2.52) we automatically find the argument that minimises the original soft-margin SVM problem (2.47) via (2.50).

2.9.5 Semi-supervised binary classification with graphs

In this section we discuss how to model semi-supervised classification problems with the help of undirected, weighted graphs. An undirected, weighted graph is defined as follows.

Definition 2.8. *An undirected* graph G *is a pair* $G = (V, E)$*, where* V *is a set of elements called* vertices*, and* $E = \{x, y \,|\, (x, y) \in V^2 \wedge x \neq y\}$ *is a set of* edges*. A* weighted graph *(or* network*) is a graph in which a number, known as* weight*, is assigned to each edge.*

An example of an undirected, weighted graph can be seen in Figure 2.2.

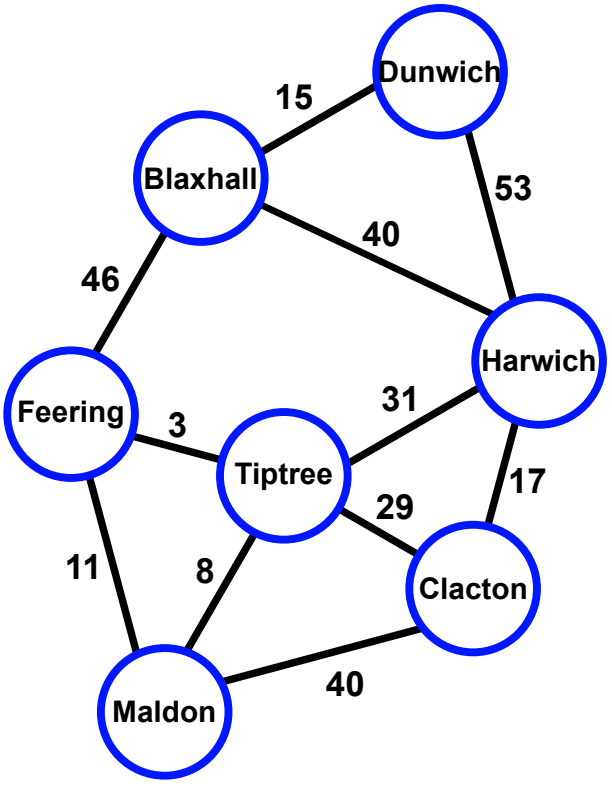


Figure 2.2: An example of a weighted graph of several connected towns in the south east of England. This graph has seven vertices (or nodes), representing different towns, connected by eleven edges. The weights represent the distances between the connected towns. ©Wikimedia commons.

For a (weighted) graph (with weights w) we define a so-called incidence matrix $M_w \in \mathbb{R}^{|E| \times |V|}$, where $|E|$ denotes the number of edges and $|V|$ the number of vertices, as

$$(M_w)_{ev} := \begin{cases} -\sqrt{w_{ev}} & \text{if } v = i \\ \sqrt{w_{ev}} & \text{if } v = j \\ 0 & \text{otherwise} \end{cases},$$

where every edge $e = (i, j)$ connects vertices i and j, with $i < j$. The corresponding *graph-*

Laplacian $L_w \in \mathbb{R}^{|V|\times|V|}$ is then defined as

$$L_w := M_w^\top M_w \,.$$

It is always best to give a concrete example for such an incidence matrix as well as the graph-Laplacian.

Example 2.6. The incidence matrix for the graph in Figure 2.2 reads

$$M_w = \begin{pmatrix} -\sqrt{15} & \sqrt{15} & 0 & 0 & 0 & 0 & 0 \\ -\sqrt{53} & 0 & \sqrt{53} & 0 & 0 & 0 & 0 \\ 0 & -\sqrt{40} & \sqrt{40} & 0 & 0 & 0 & 0 \\ 0 & -\sqrt{46} & 0 & 0 & \sqrt{46} & 0 & 0 \\ 0 & 0 & 0 & -\sqrt{3} & \sqrt{3} & 0 & 0 \\ 0 & 0 & -\sqrt{31} & \sqrt{31} & 0 & 0 & 0 \\ 0 & 0 & 0 & -\sqrt{29} & 0 & \sqrt{29} & 0 \\ 0 & 0 & -\sqrt{17} & 0 & 0 & \sqrt{17} & 0 \\ 0 & 0 & 0 & 0 & -\sqrt{11} & 0 & \sqrt{11} \\ 0 & 0 & 0 & -\sqrt{8} & 0 & 0 & \sqrt{8} \\ 0 & 0 & 0 & 0 & 0 & -\sqrt{40} & \sqrt{40} \end{pmatrix},$$

while the corresponding graph Laplacian is then given as

$$L_w = M_w^\top M_w = \begin{pmatrix} 68 & -15 & -53 & 0 & 0 & 0 & 0 \\ -15 & 101 & -40 & 0 & -46 & 0 & 0 \\ -53 & -40 & 141 & -31 & 0 & -17 & 0 \\ 0 & 0 & -31 & 71 & -3 & -29 & -8 \\ 0 & -46 & 0 & -3 & 60 & 0 & -11 \\ 0 & 0 & -17 & -29 & 0 & 86 & -40 \\ 0 & 0 & 0 & -8 & -11 & -40 & 59 \end{pmatrix}.$$

The graph Laplacian can also be decomposed into a so-called degree matrix D_w and an adjacency matrix A_w via $L_w = M_w - A_w$, which we in the interest of time will not explore any further throughout this lecture.

We can use weighted graphs to model and exploit similarities in datasets. Suppose we are given a set $S := \{x_i\}_{i=1}^s$ of s samples, for which only $r \ll s$ samples in the set $R := \{x_{i(j)}\}_{j=1}^r$ have corresponding (binary) output labels $\{y_j\}_{j=1}^r$ with values in $\{0, 1\}$. Here $i : \{1, \dots, r\} \to \{1, \dots, s\}$ denotes an index function that picks the indices for which labels are known. If r is very small, supervised learning on just r samples may not lead to mappings that have satisfactory predictive powers when applied to the classification of new samples x. However, we can assume that all data points $\{x_i\}_{i=1}^s$ are nodes in a connected graph. The weights between each nodes are determined by a similarity measure, for example of the form

$$w_{ij} = \begin{cases} \exp\left(-\gamma\|x_i - x_j\|^2\right) & \|x_i - x_j\| \le \text{threshold} \\ 0 & \|x_i - x_j\| > \text{threshold} \end{cases},$$

for a constant $\gamma > 0$ and a threshold value that guarantees that not all weights are non-zero. This way we create an undirected weighted graph with connections between nodes where there is similarity in terms of the Euclidean norm. Based on this graph with weights w, we can construct a

corresponding incidence matrix M_w. This type of supervised machine learning is usually referred to as semi-supervised, as we only require a smaller subset R of a set of (training) samples S instead of the entire set of training samples in order to perform the binary classification task.

One could now try to pursue a supervised classification task by solving the following optimisation problem:

$$\hat{v} = \underset{v\in[0,1]^s}{\arg\min} \left\{ \|M_w v\|^2 \text{ subject to } (P_R v)_j = y_j \text{ , for all } j \in \{1,\dots,r\} \right\} . \tag{2.53}$$

Here $P_R : \mathbb{R}^s \to \mathbb{R}^r$ denotes the projection of a vector on the indices specified by the index function i, i.e.

$$(P_R v)_j = v_{i(j)} \,, \qquad \forall j \in \{1,\dots,r\} \,.$$

The label vector $\hat{v}$ is constrained to have values in $[0,1]^s$ and to take on the correct values for the indices with corresponding output labels. The remaining values are determined by ensuring minimal $\|M_w \hat{v}\|^2$ for $\hat{v}$ amongst all possible label vectors $v \in \mathbb{R}^s$.

Note that given the sets S and R, we can easily define the complement $R^\perp := S \setminus R$ of R. If we denote the projection onto this set with $P_{R^\perp} : \mathbb{R}^s \to \mathbb{R}^{s-r}$, we can rewrite v as

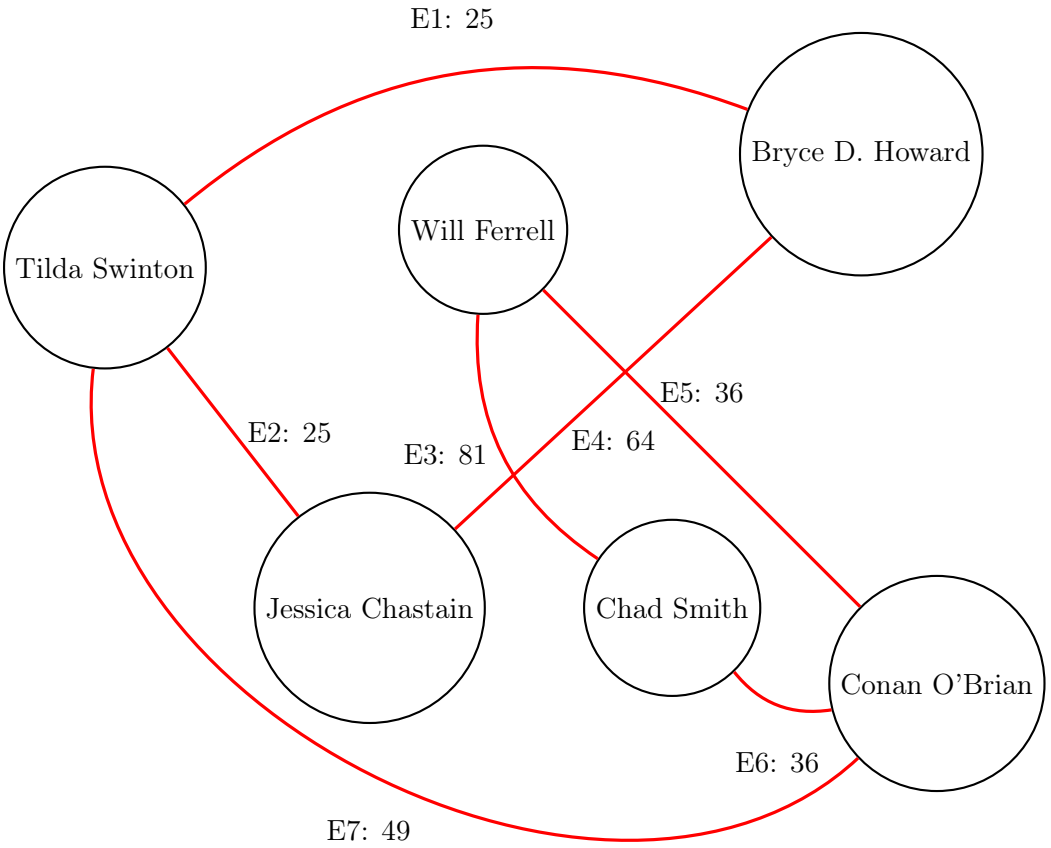
$$v = P_{R^\perp}^\top P_{R^\perp} v + P_R^\top P_R v = P_{R^\perp}^\top P_{R^\perp} v + P_R^\top y \,.$$

Note that we only need to compute $P_{R^\perp}\hat{v}$ instead of $\hat{v}$ as $P_R \hat{v} = y$ is already known. As a direct consequence, we can reformulate (2.53) to

$$\begin{aligned} P_{R^\perp}\hat{v} &= \underset{\tilde{v}\in[0,1]^{s-r}}{\arg\min} \left\{ \left\| M_w \left(P_{R^\perp}^\top \tilde{v} + P_R^\top y \right) \right\|^2 \right\} \\ &= \underset{\tilde{v}\in\mathbb{R}^{s-r}}{\arg\min} \left\{ \left\| M_w \left(P_{R^\perp}^\top \tilde{v} + P_R^\top y \right) \right\|^2 \right\} , \end{aligned} \tag{2.54}$$

where the last equality holds when $y \in [0,1]^r$. In the following, we want to give a small example to illustrate this approach of binary classification.

Example 2.7. We consider the following graph where its weights mimic a similarity measure between the individuals in the nodes:



Wherever there is no edge between two nodes, the corresponding weight, respectively the similarity measure, is zero. Computing the incidence matrix M_w of this weighted graph yields

$$\left(\begin{array}{c|cccccc} \text{E1} & -5 & 0 & 0 & 0 & 5 & 0 \\ \text{E2} & 0 & 0 & 0 & -5 & 5 & 0 \\ \text{E3} & 0 & -9 & 0 & 0 & 0 & 9 \\ \text{E4} & -8 & 0 & 0 & 8 & 0 & 0 \\ \text{E5} & 0 & 0 & -6 & 0 & 0 & 6 \\ \text{E6} & 0 & -6 & 6 & 0 & 0 & 0 \\ \text{E7} & 0 & 0 & -7 & 0 & 7 & 0 \\ \hline & \text{B. D. Howard} & \text{C. Smith} & \text{C. O' Brian} & \text{J. Chastain} & \text{T. Swinton} & \text{W. Ferrell} \end{array}\right).$$

The corresponding graph Laplacian reads

$$L_w = M_w^\top M_w = \begin{pmatrix} 89 & 0 & 0 & -64 & -25 & 0 \\ 0 & 117 & -36 & 0 & 0 & -81 \\ 0 & -36 & 121 & 0 & -49 & -36 \\ -64 & 0 & 0 & 89 & -25 & 0 \\ -25 & 0 & -49 & -25 & 99 & 0 \\ 0 & -81 & -36 & 0 & 0 & 117 \end{pmatrix}.$$

Suppose we want to classify each node according to biological sex, and already know that Jessica Chastain is female (class label $\hat{v}_4 = 1$) and Will Ferrell is male (class label $\hat{v}_6 = 0$). We can therefore formulate (2.54) with the projections

$$P_{R^\perp} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad \text{and} \quad P_R = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

and the data $y = \begin{pmatrix} 1 & 0 \end{pmatrix}^\top$. This leaves us with the solution of the linear system

$$P_{R^\perp} L_w P_{R^\perp}^\top \underbrace{\tilde{v}}_{= P_{R^\perp}\hat{v}} = -P_{R^\perp} L_w P_R^\top y \,,$$

which for this example reads

$$\begin{pmatrix} 89 & 0 & 0 & -25 \\ 0 & 117 & -36 & 0 \\ 0 & -36 & 121 & -49 \\ -25 & 0 & -49 & 99 \end{pmatrix} \tilde{v} = \begin{pmatrix} 64 \\ 0 \\ 0 \\ 25 \end{pmatrix}.$$

The solution to this linear system is (approximately) $\tilde{v} \approx \begin{pmatrix} 0.8912 & 0.0840 & 0.2732 & 0.6128 \end{pmatrix}^\top$, respectively

$$\hat{v} = \begin{pmatrix} 0.8912 & 0.0840 & 0.2732 & 1 & 0.6128 & 0 \end{pmatrix}^\top.$$

We can use this result to classify the remaining nodes by setting all values below 1/2 to zero and above 1/2 to one. In this example, we would (correctly) determine the biological sex of Bryce Dallas Howard as female, of Chad Smith as male, of Conan O' Brian as male, and of Tilda Swinton as female.