Information Retrieval In High Dimensional Data

Lecture Notes

Martin Kleinsteuber, Hao Shen, Matthias Seibert, Alexander Sagel

November 8, 2018

Contents

Symbol	meaning
X, Y, Z	random variables, possibly mulivariate
$\mathcal{S}, \mathcal{T}$	Sets
p	dimension of "raw data space"
k	dimension of "reduced data space" or "Feature Space"
$\mathbf{X}, \mathbf{Y}, \mathbf{Z}$	matrices; X often denotes the (centered) observation matrix
$\overline{X}$	centered observation matrix
$\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{u}, \mathbf{v}, \mathbf{w}$	vectors
$(\cdot)^\top$	transpose of a matrix or a vector
$\mathbf{a}^\top \mathbf{b}$	scalar (or *dot*-)product of $\mathbf{a}$ and $\mathbf{b}$
$\operatorname{tr} \mathbf{A}$	trace of $\mathbf{A} \in \mathbb{R}^{n \times n}$, i.e. the sum of the diagonal elements of $\mathbf{A}$
$\operatorname{tr}(\mathbf{A}^\top \mathbf{B})$	scalar product for matrices
$\mathbb{E}[X]$	expected value of X
$\operatorname{Var}[X]$	covariance matrix of the multivariate random variable X; by abuse of notation often also an estimation of the covariance matrix
$\hat{\mu}, \hat{\Sigma}, \ldots$	estimated quantities; often the hat is omitted
pdf	probability density function
$L\big((\cdot), (\cdot)\big)$	loss function
EPE	expected prediction error
$\Pr(X)$	probability measure induced by X
$W(X = x)$	probability for random variable X to take the value of x
Ω	probability space
$\mathcal{F}(\Omega)$	σ-algebra of Ω
μ_X	induced probability measure
$\Pr(Y \mid X)$	conditional probability measure of Y given X
$\operatorname{pr}_{\mathcal{U}}$	projection onto the subspace $\mathcal{U}$
$\operatorname{span}(\mathbf{u}_1, \ldots, \mathbf{u}_k)$	subspace spanned by the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$
$\operatorname{diag}(a_1, \ldots, a_m)$	short for an $m \times m$ matrix with only the diagonal entries $a_1, \ldots, a_m$ being different from zero
I_k	$k \times k$ identity matrix
$\mathbb{1}_n$	vector in $\mathbb{R}^n$ consisting of $\mathbb{1}_n = [1, \ldots, 1]^\top$

1 Introduction

1.1 Information Retrieval and Machine Learning

The problem of extracting information out of high dimensional data is inseparably connected to the question of dimensionality reduction, i.e. to the problem of extracting *a few* reasonable features out of typically high dimensional observations. Consider for example the humans ability to recognize faces on an image. The image, seen as a high dimensional vector of, say 800×600 pixel values, can surely not be stored in a humans brain as *raw* pixel-data. Instead, there must be some features that we extract, which may be e.g. relative distance between eyes, length of the nose, and more abstract the interplay of different face regions as a whole. The ability of storing and recalling these few abstract features make it possible to recognize a face regardless of varying background, sunglasses or partial occlusions and to distinguish between different faces. There are more examples in the wide field of data analysis, where the extraction of features allows to squeeze information out of high dimensional data, ranging from gene data classification to audio signal processing, from data visualization to the analysis of electroencephalography (EEG) data.

Formally, the problem of dimensionality reduction is the following: Given a p-dimensional real valued random variable $X = \begin{bmatrix} X_1 \dots X_p \end{bmatrix}^\top$, find a map or algorithm

$$f : \mathbb{R}^p \to \mathbb{R}^k \text{ with } k \ll p,$$

such that $S = f(X)$ contains "as much information from X as possible". In the spirit of the above mentioned example, $\mathbb{R}^p$ will be referred to as *raw data space* and $\mathbb{R}^k$ as *reduced data space* or *feature space*.

For example the preservation of information might be measured by the variance, so that the variance of S should reflect the variance of X. This can also be interpreted as to eliminate redundancy in the data. Consider the following example: the temperature is measured, one time in degree Celcius (which would be the first entry X_1 of the random variable) and in degree Fahrenheit (X_2). It is obvious that this information can be reduced to one variable, say $S_1 = X_1$, even without any loss of information.

The matrix $\mathbf{X} \subset \mathbb{R}^{p \times n}$ where its (i, j)-entry x_{ij} denotes the realization of the random variable X_i at observation j is called *observation matrix*. Its columns are realizations of the p-dimensional random variable X.

The expected value is referred to by $\mathbb{E}(X) = \mu \in \mathbb{R}^p$. Since we are dealing with a multivariate random variable, the variance is now expressed by the *covariance matrix* (also called the variance-covariance matrix) which is defined as

$$\Sigma = \mathrm{Var}(X) = \mathbb{E}((X - \mu)(X - \mu)^\top) \in \mathbb{R}^{p \times p}. \tag{1.1}$$

Its (i,j)-entry is the covariance between the i^{th} and the j^{th} random variables. The covariance matrix is symmetric, i.e. $\Sigma = \Sigma^\top$, and positive semidefinite[1], i.e. $\Sigma \geq 0 \Leftrightarrow x^\top \Sigma x \geq 0 \ \forall x$.

Example 1.1. Consider two constant random variables $X_1 \equiv \text{const.}, X_2 \equiv \text{const.}$. This means we have a 2-dimensional random variable with the covariance matrix $\Sigma = 0$. This example shows that Σ is not necessarily positive definite.

As the actual distribution of a random variable is typically unknown, the expectation value is usually estimated on the base of n observations:

$$\frac{1}{n}\sum_{j=1}^{n}\begin{bmatrix} x_{1j} \\ \vdots \\ x_{pj} \end{bmatrix} = \tfrac{1}{n}\mathbf{X}\,\mathbb{1}_n := \hat{\mu} \tag{1.2}$$

Using this estimated expectation and the Kronecker product[2] denoted by $\otimes$, the centered observation matrix $\bar{\mathbf{X}}$ may be computed as follows:

$$\bar{\mathbf{X}} = \mathbf{X} - \hat{\mu} \otimes \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \tag{1.3}$$

With the centered observation matrix $\bar{\mathrm{X}}$, the covariance matrix $\Sigma = \mathrm{Cov}(X)$ can be estimated by

$$\widehat{\Sigma} = \frac{1}{n-1}\bar{\mathbf{X}}\bar{\mathbf{X}}^\top.$$

Since in practice n tends to be large, it is also possible to use the approximation $\frac{1}{n}\bar{\mathbf{X}}\bar{\mathbf{X}}^\top$.

[1] in contrast to positive definite, i.e. $x^\top \Sigma x > 0 \ \forall x \neq 0$ and $x^\top \Sigma x = 0 \Leftrightarrow x = 0$

[2] The Kronecker product of two matrices $\mathbf{A} \otimes \mathbf{B}$ with $\mathbf{A} = \{a_{ij}\} \in \mathbb{R}^{k\times l}, \mathbf{B} = \{b_{ij}\} \in \mathbb{R}^{m\times n}$ is a $(km \times ln)$-matrix $\mathbf{C}$, such that $\mathbf{C} = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1l}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{k1}\mathbf{B} & \cdots & a_{kl}\mathbf{B} \end{bmatrix}$.

1.2 Preliminaries on Random Variables

We want to recall some basic definitions and notations from probability theory that we will need occasionally throughout this lecture notes. It will be sufficient for our purposes to consider *real multidimensional* random variables that are either continuous or discrete. More formally, let $X\colon \Omega \to \mathbb{R}^p$ be a random variable and denote its density with respect to the usual Lebesgue measure as $p_X(x)$. We will use the very sloppy but very convenient notation $X \in \mathbb{R}^p$ to indicate that the random variable X takes values in $\mathbb{R}^p$.

For (absolutely) continuous random variables, the density is a continuous function from $\mathbb{R}^p$ to $\mathbb{R}$. In case of a discrete random variable which takes values x_i with probability p_i, we employ the Dirac-Delta-Function[3] to describe its density, namely

$$p_X(x) = \sum_i p_i \delta(x - x_i). \tag{1.4}$$

So, if $\mathcal{A} \subset \mathbb{R}^p$ is a subset[4], the probability that X takes values in $\mathcal{A}$ is given by

$$Pr(X \in \mathcal{A}) = \int_{\mathcal{A}} p_X(x)\mathrm{d}x. \tag{1.5}$$

Note, that in the case of a discrete random variable, this expression is just

$$Pr(X \in \mathcal{A}) = \int_{\mathcal{A}} \sum_i p_i \delta(x - x_i)\mathrm{d}x = \sum_{\{i \mid x_i \in \mathcal{A}\}} p_i. \tag{1.6}$$

By knowing the joint density $p_{X,Y}(x, y)$ of two random variables $X \in \mathbb{R}^p$ and $Y \in \mathbb{R}^k$, it is possible to deduce the individual densities of X and Y, respectively. These are called the *marginal* densities, and they are given by

$$p_X(x) = \int_{\mathbb{R}^k} p_{X,Y}(x, y)\mathrm{d}y, \tag{1.7}$$

$$p_Y(y) = \int_{\mathbb{R}^p} p_{X,Y}(x, y)\mathrm{d}x. \tag{1.8}$$

If the joint density function is given, the knowledge of a certain realization of one of the two variables, say X, allows to infer information about the distribution of Y. The resulting density function is called *conditional density function*, and, if the realisation of X is $x \in \mathbb{R}^p$, it is given by

$$p_{Y|X=x}(y) = \frac{p_{X,Y}(x, y)}{p_X(x)}. \tag{1.9}$$

[3] The Dirac-Delta-Function fulfills the condition that $\delta(t) = 0$ for $t \neq 0$ and $\int_{\mathbb{R}^p} \delta(t)\mathrm{d}t = \mathbb{1}_p$. i.e. δ has an infinitely high peak at 0.

[4] Formally, this set has to be *measurable w.r.t. the Borel σ-algebra*, but if you do not know what measurable is, all subsets that you can imagine satisfy this condition.

There are two quantities that play a prominent role in describing statistical properties of a random variable $X \in \mathbb{R}^p$. These are the first and the second moment, also known as the *expectation value*

$$\mathbb{E}[X] = \int_{\mathbb{R}^p} x p_X(x) \mathrm{d}x =: \mu \tag{1.10}$$

and the *variance/covariance*

$$\mathrm{Var}[X] = \int_{\mathbb{R}^p} (x-\mu)(x-\mu)^\top p_X(x) \mathrm{d}x. \tag{1.11}$$

Note, that $\mu \in \mathbb{R}^p$ and that $\mathrm{Var}[X]$ is a positive semidefinite matrix in $\mathbb{R}^{p\times p}$.

Exercise: Show that the variance/covariance matrix is positive semidefinite.

	x_1	x_2	x_3	x_4	$p_y(Y)\downarrow$
y_1	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{4}$
y_2	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{32}$	$\frac{1}{32}$	$\frac{1}{4}$
y_3	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{4}$
y_4	$\frac{1}{4}$	0	0	0	$\frac{1}{4}$
$p_x(X)\rightarrow$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$	1

Table 1.1: This table shows an exemplary joint probability distribution.

Example 1.2. An example of a joint probability distribution of a two dimensional discrete random variable is given in Table 1.1. The marginal densities are denoted by $p_Y(y)$ and $p_X(x)$, respectively. As an exercise, compute the conditional density of X given $Y = y_2$. The answer is given in this footnote[5].

[5] Answer: $p_{X|Y=y_2}(x) = \sum_i p_i \delta(x - x_i)$, with $p_1 = 1/4$, $p_2 = 1/2$, $p_3 = 1/8$, $p_4 = 1/8$.

2 Statistical Decision Making and Machine Learning

The basic problem is that for some observation of the random variable $X \in \mathbb{R}^p$, we want to obtain the "most probable" value of the random variable Y. For simplicity, we assume that Y takes realizations in $\mathbb{R}$. We further assume for the moment that the induced joint probability density $p_{X,Y}(x, y)$ is given.

Example 2.1. X is a (vectorized) image of a person. In Figure 2.1 below this observation is represented a picture that contains just one pixel with varying levels of gray.

$$Y = \begin{cases} 1: & X \text{ is picture of person A} \\ 0: & X \text{ is not} \end{cases}$$

The aim is to predict Y for a given X. If person A is wearing predominantly dark clothes

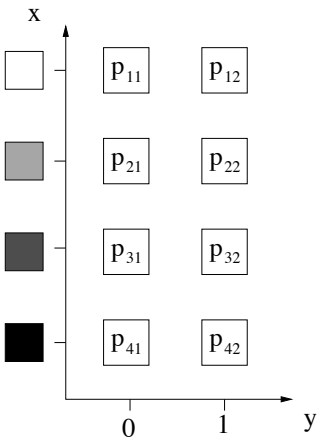

Figure 2.1: Joint PDF of the two discrete random variables X and Y.

it is reasonable to assume that the respective probabilities for the events behave like this: $p_{42} > p_{32} > p_{22} > p_{12}$ and $p_{11} > p_{21} > p_{31} > p_{41}$.

In order to formalize what we mean by "most probable", we consider the *quadratic loss function*[1]

$$L\big(Y, f(X)\big) = \big(Y - f(X)\big)^2. \tag{2.1}$$

[1]Note, that also other loss functions are possible and also do make sense indeed, and that the choice of the squared distance is only due to convenience reasons.

We want to choose f such that the expected value of the loss function, the so called Expected Prediction Error

$$\mathrm{EPE}(f) = \mathbb{E}\Big[L\big(Y, f(X)\big)\Big]$$

is minimized. A simple computation that uses to tools described in subsection 1.2 yields

$$\begin{aligned}\mathrm{EPE}(f) = \mathbb{E}\Big[L\big(Y, f(X)\big)\Big] = \mathbb{E}\Big[\big(Y - f(X)\big)^2\Big] = \int_{\mathbb{R}^p \times \mathbb{R}} \big(y - f(x)\big)^2 p_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y = \\ = \int_{\mathbb{R}^p} \underbrace{\int_{\mathbb{R}} \big(y - f(x)\big)^2 p_{Y|X=x}(y)\mathrm{d}y}_{=:\ \mathbb{E}_{Y|X=x}\left[(Y-f(X))^2\right]} p_X(x)\mathrm{d}x = \mathbb{E}_X \mathbb{E}_{Y|X=x}\Big[\big(Y - f(X)\big)^2\Big]\end{aligned}$$

and hence $f(X) = \arg\min_c \mathbb{E}_{Y|X=x}\big[(Y-c)^2\big]$. Due to the linearity of $\mathbb{E}[\cdot]$, this reduces to

$$f(X) = \arg\min_c \Big(\mathbb{E}_{Y|X=x}\big[Y^2\big] - 2c\ \mathbb{E}_{Y|X=x}\big[Y\big] + c^2\Big). \tag{2.2}$$

The quadratic term can easily be minimized with the optimal value

$$f(X) = \mathbb{E}_{Y|X=x}\big[Y\big]. \tag{2.3}$$

Theorem 2.2. *The best prediction of Y given X is the* conditional mean*, if* best *is measured with respect to the squared loss.*

As stated above, the conditional mean as best prediction relies on the fact that we use the squared loss as a quality measure. One can show that if we employ the absolute value $|Y - f(X)|$ as a loss function - the so-called ℓ_1*-loss* - instead, the best prediction is the *conditional median*. Although a rigorous proof of this statement is beyond the scope of this lecture notes, we can easily see that the median is optimal for the minimization of the *empirical* ℓ_1-loss

$$\arg\min_c \ \tfrac{1}{n} \sum_i |y_i - c|. \tag{2.4}$$

All we need is a result from non-smooth convex optimization which states that for convex functions with subgradients, the minimum c^* is achieved if and only if the subgradient at c^* contains 0. In our case, the subgradient of the *empirical* ℓ_1-loss with respect to c is $\frac{1}{n}\sum_i \mathrm{sign}(y_i - c)$ for $c \neq y_i$ and $\left\{\frac{1}{n}\sum_{i\neq j} \mathrm{sign}(y_i - c) + t| - 1 \leq t \leq 1\right\}$ for $c = y_j$. So we see that the subgradient at c contains zero if an only if there are as many y_i's that are *smaller* than c than there are y_i's that are *larger*. In case of odd n, c has to coincide with the $(n+1)/2$-th greatest value of the y_i's. This is exactly the definition of the median.

2.1 General Setting for Supervised Decision Making and the Generalization Error

Let us resume the above section on a higher level perspective. We have introduced a loss function that, based on training samples $(x_i, y_i), i = 1, \ldots, N$, allows to learn the *best* prediction function f out of a given function class $\mathcal{F}$. Such methods are called *supervised learning methods*, since they require a training set where to each sample x_i we have a corresponding y_i. The general problem statement is:

Let $(X, Y) \in \mathbb{R}^p \times \mathbb{R}$ be a random variable and let $\mathcal{F}$ be a class of functions from $\mathbb{R}^p \to \mathbb{R}$ that can be parameterized by finitely many parameters, say $\Theta \in \mathbb{R}^M$. Moreover, let $L \colon \mathbb{R} \times \mathbb{R} \to \mathbb{R}_0^+$ be a Loss function, that measures the deviation of Y and $f(X)$. The aim of a supervised prediction method is then to find $\hat{f} \in \mathcal{F}$ that minimizes the expected prediction error $\mathbb{E}[L(Y, f(X)]$.

In fact, there are three issues that we need to clarify in order to build a supervised machine learning method.

- We have to specify the loss function L.
- We have to specify the function class $\mathcal{F}$.
- Since in practice, we do not know the joint distribution of (X, Y), we need training samples $(x_i, y_i), i = 1, \ldots, N$ to approximate the expected prediction error.

Once this is done, we arrive at the optimization problem

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)). \tag{2.5}$$

These optimization problems can be more or less hard to solve, but in principle, they can be fed into an optimization toolbox in the above form.

Example: Linear Regression. For linear regression, we choose L to be the quadratic loss, i.e. $L(Y, f(X)) = (Y - f(X))^2$, and we choose $\mathcal{F}$ to be the class of affine functions

$$f(x) = \theta_0 + \sum_{k=1}^{p} \theta_k x^{(k)}, \tag{2.6}$$

where $x^{(k)}$ are the entries of the vector x. Given N training samples, the optimization problem then is

$$\hat{\Theta} = \arg\min_{\theta_0, \ldots, \theta_p} \frac{1}{N} \sum_{i=1}^{N} (y_i - \theta_0 - \sum_{k=1}^{p} \theta_k x_i^{(k)})^2. \tag{2.7}$$

Definition: The difference between the empirical loss and the expected loss for a given function f

$$G_N(f) = \mathbb{E}[L(Y, f(X)] - \frac{1}{N}\sum_{i=1}^{N} L(y_i, f(x_i)) \tag{2.8}$$

is called the generalization error of f.

We know by the law of large numbers, that G_N tends to 0 as N tends to infinity. However, very often the number of training samples is limited, and moreover, the speed of converges highly depends on the (unknown) probability distribution of (X, Y). One goal in Machine Learning is therefore to bound the generalization error *with high probability*.

In practice, we use cross validation to check how well f fits the data.

TODO: Explain cross validation!!

TODO: Explain Overfitting!!

2.2 k-nearest neighbors

In practice, we face the problem that we do not know the joint distribution of X and Y and we thus have to estimate the conditional mean in Eq. (2.3). Typically, this would be done with the relative frequency of Y given the particular realization x. However, in the continuous case the problem arises, that even among many observations (x_i, y_i), there is probably none with $x_i = x$. This problem is resolved by enlarging the region around x and taking all observations into account where x_i is *near* the desired x. This leads to the concept of k-nearest neighbors.

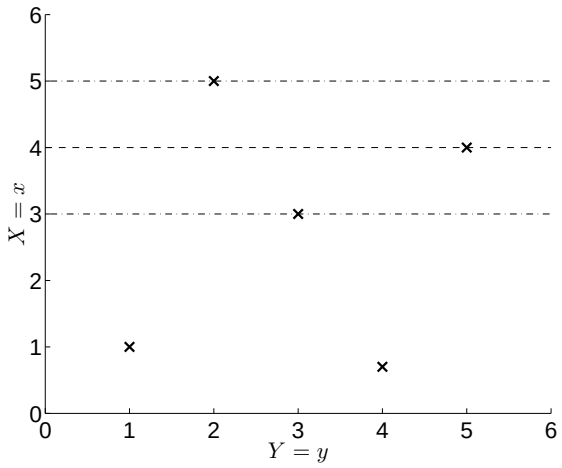

Figure 2.2: Given the events $(1,1), (2,5), (3,3), (4,1), (5,4)$ for the bivariate distribution (X, Y). While $\mathbb{E}_{Y|X=4}[Y] = 5$, averaging over the 3 nearest neighbors provides $\hat{f}_{k=3}(4) = (2+3+5)/3 = 10/3$.

The approach of k-nearest neighbors is to estimate f by

$$\hat{f}_k(x) = \text{average}\big(\ y_i \mid x_i \in N_k(x)\big), \tag{2.9}$$

where $N_k(x)$ denotes the set of the k nearest neighbors of x. There are two approximations made here:

1. Expectation is approximated by averaging.
2. Conditioning at a point is approximated by conditioning in some region (around that point).

If N is the number of observations, the following is to be noted. If N increases, the x_i come close to x. Moreover, if k increases, the average tends towards the expectation value. More precisely, under mild conditions on the joint probability measure, we have

$$\lim_{N,k\to\infty,\,\frac{k}{N}\to 0} \hat{f}(x) = \mathbb{E}_{Y|X=x}[Y]. \tag{2.10}$$

However, the sample size N is generally limited, and therefore not large enough.
There are two ways to overcome this problem:

- by imposing model assumptions on f, e.g. $f(x) \approx x^{\mathrm{T}}\beta$ gives linear regression.
- by "reducing" the dimension of X. This leads to the motivation for dimensionality reduction:
  Find $f\colon \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix} \mapsto \begin{bmatrix} s_1 \\ \vdots \\ s_k \end{bmatrix}$, such that f maintains the "intrinsic" distance of observations.

2.3 Curse of dimensionality

This section will give the motivation for dimensionality reduction as a whole. The term *curse of dimensionality* was coined by Bellman in 1961. It refers to several phenomena that occur when analyzing data in high dimensional space and interfere with statistical decision making.

Let $X \in \mathbb{R}^p$ be random variable. There are several things to be observed:

First, the absolute noise in X increases with the number of features (i.e. with p) since the noise accumulates over all dimensions.

Second, the number of observations required for estimating the density function of X increases dramatically with the dimension p. Density functions are usually estimated by the help of the relative frequency of an incidence. As an example assume that N observations in a 1-dimensional space are required to estimate a certain density function up to a predefined accuracy. If the observation space is increased to 2 dimensions, the number of observations has to be increased to order N^2 to maintain the same accuracy. For a 3-dimensional space around N^3 observations are required, and so forth. This means that the number of measurements required to give the same accuracy for an estimation of a density function increases exponentially with the dimension of the measurement space.

One cause for the curse of dimensionality is the *empty space phenomenon* (Scott), which states that high dimensional spaces are inherently sparse. Given data points uniformly distributed in a 10-dimensional unit sphere, the probability that a point is closer than 0.9 to the center is less than 35%. Therefore, the tail in high-dimensional distributions is far more important than in one-dimensional ones. Given a high dimensional multivariate random variable the fact that one component lying in the tail of its distribution is enough to cause the whole sample to lie in the tail of the common density function. The 1-dimensional case is illustrated in Figure 2.3.

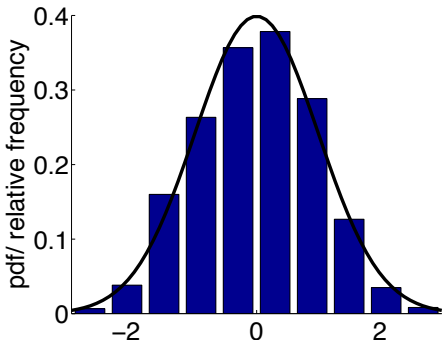

Figure 2.3: With one-dimensional random variables most observations will be around the mean value. This is not the case for higher dimensions.

3 Logistic Regression

The general setting when talking about logistic regression is that we have data points $X \in \mathbb{R}^p$ and output variables $Y \in \{-1, +1\}$. This is a so-called binary classification problem.

The task then is to find the function f in a predefined class of functions $\mathcal{F}$ such that sign $f(X)$ predicts Y as good as possible. A commonly used loss function to measure the 'accuracy' of the predicting functions is motivated by the number of misclassifications, i.e. if the sign of $f(X)$ does not coincide with the sign of the true output Y. The so called $0,1$-loss function

$$L_{0,1}(Y, f(X)) = \begin{cases} 1 & \text{if } Y \operatorname{sign} f(X) \leq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

does the job.

In order to find an optimal prediction function $f \in \mathcal{F}$ for a given set of training samples $(x_i, y_i)_{i=1,\dots,n}$ the goal is to find $f \in \mathcal{F}$ that minimizes the empirical expected loss

$$\frac{1}{n} \sum_{i=1}^{n} L_{0,1}(y_i, f(x_i)). \tag{3.2}$$

However, it is numerically infeasible to find a minimum of this problem. Even just considering the class of affine functions $\mathcal{F}_{\text{aff}}$, this is hard to solve numerically, since the loss function is neither continuous nor differentiable. In order to make it easier to solve we approximate the (non-continuous, non-convex) function $L_{0,1}$ by a convex loss function.

Interlude: Convexity

Definition 3.1. Let $\mathcal{C} \subset \mathbb{R}^n$ be a convex set, i.e. for any pair of elements $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C}$, the point $t\mathbf{x}_2 + (1-t)\mathbf{x}_1$ is also an element of $\mathcal{C}$ for all $t \in [0,1]$. A function $f\colon \mathcal{C} \to \mathbb{R}$ is called *convex* if $tf(\mathbf{x}_2) + (1-t)f(\mathbf{x}_1) \geq f(t\mathbf{x}_2 + (1-t)\mathbf{x}_1)$ for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C}$, $t \in [0,1]$. It is called *strictly convex* if the inequality is strict.

Example: $f\colon \mathbb{R}^+ \to \mathbb{R},\ x \mapsto 1/x$ is convex.

Theorem 3.2. *If f, g are convex, then so are*

- $h = \max(f, g)$
- $h = f + g$
- $h = g \circ f$ *if g is non-decreasing*

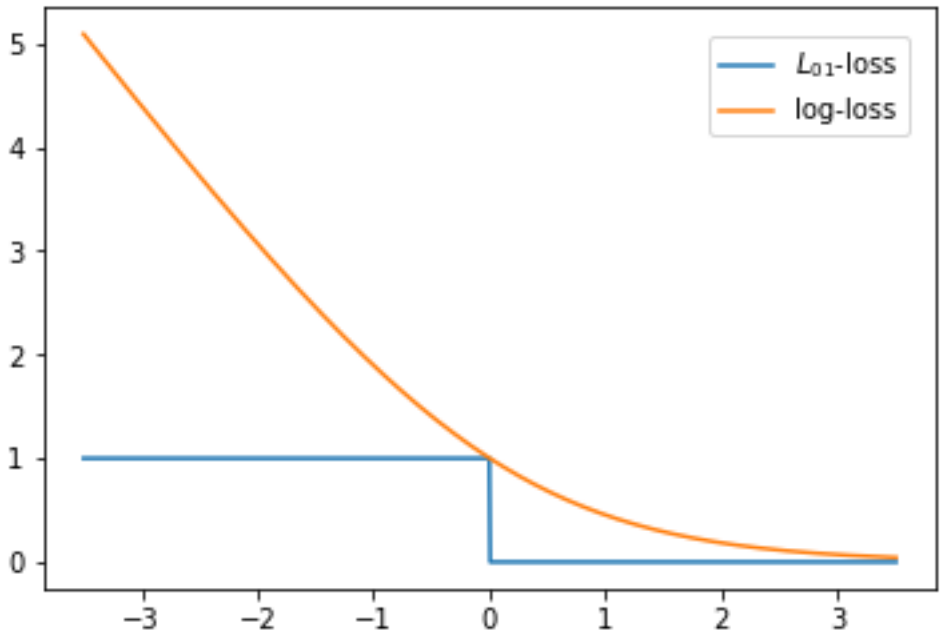

Figure 3.1: $L_{0,1}$ and log-loss with $t := yf(x)$ on the horizontal axis

Theorem 3.3. *A local minimum of a strictly convex function coincides with its global minimum. It is unique if it exists.*

For the problem at hand, that means for us if we choose f to be affine (i.e. $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$, which is convex by definition) and as the loss function we use the convex log-loss function $\ell(t) = \log(1 + e^{-t})$. The concatenation of both is convex. To see this compute the second derivative and investigate its Hessian. It turns out, that it has only non-negative eigenvalues. The choice of the log-loss is motivated by the fact that it can be interpreted as a convex approximation of the $L_{0,1}$-loss as can be seen in Fig. 3.1. Given some training data $\{(\mathbf{x}_i, y_i)\}_{i=1,\dots,N}$, we can find the optimal parameters w and b by solving the optimization problem

$$\min_{\mathbf{w}\in\mathbb{R}^p, b\in\mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i(\mathbf{w}^\top \mathbf{x}_i + b))). \tag{3.3}$$

Convexity of the cost function For simplicity, we only consider the cost function for linear f. The extension to affine f is straightforward. Let us denote it by

$$F(\mathbf{w}) = \sum_{i=1}^{n} \log(1 + \exp(-y_i(\mathbf{w}^\top \mathbf{x}_i))).$$

We also use the auxiliary function $g(z) = 1/(1+e^{-z})$, where $g'(z) = g(z)(1-g(z))$. Then, the first and second partial derivative for F are

$$\frac{\partial}{\partial w^{(j)}} F(\mathbf{w}) = -\sum_i y_i x_i^{(j)} (1 - g(y_i \mathbf{w}^\top \mathbf{x}_i))$$

and

$$\frac{\partial}{\partial w^{(j)} \partial w^{(k)}} F(\mathbf{w}) = \sum_i y_i^2 x_i^{(j)} x_i^{(k)} g(y_i \mathbf{w}^\top \mathbf{x}_i)(1 - g(y_i \mathbf{w}^\top \mathbf{x}_i)),$$

where $y_i^2 = 1$. To show that the function is non-negative definite, we need to show that $a^\top \nabla^2 F a \geq 0$ for all a. We define the auxiliary variables $P_i = g(y_i \mathbf{w}^\top \mathbf{x}_i)(1 - g(y_i \mathbf{w}^\top \mathbf{x}_i))$ and $\rho_i^{(j)} = x_i^{(j)} \sqrt{P_i}$. Then

$$a^\top \nabla^2 F a = \sum_i \sum_{j,k} a_i a_j x_i^{(j)} x_i^{(k)} P_i = \sum_i a^\top \rho_i \rho_i^\top a \geq 0.$$

This holds for any concatenation of convex and affine functions.

In the following, we give a probabilistic interpretation of this optimization problem. First of all, note that the conditional probability of $Y = y$ given the observation $\mathbf{x}$ is defined as

$$Pr(Y = y|x) = \exp\big(-\ell(y, f(x))\big) = \frac{1}{1 + \exp(-y(\mathbf{w}^\top \mathbf{x} + b))}. \tag{3.4}$$

In order to find a solution to (3.3), it is common practice to use gradient based methods. The simplest form is gradient descent where at a given iteration point, we compute the gradient and then take a step in the negative direction of that gradient. The gradient of the function $F(\mathbf{w}, b) = \sum_i \log(1 + \exp(-y_i(\mathbf{w}^\top \mathbf{x}_i + b)))$ can be determined by computing the partial derivatives

$$\begin{aligned}
\frac{\partial}{\partial w^{(k)}} F(\mathbf{w}, b) &= \sum_{i=1}^{n} \frac{\exp(-y_i(\mathbf{w}^\top \mathbf{x}_i + b))}{1 + \exp(-y_i(\mathbf{w}^\top \mathbf{x}_i + b))} (-y_i x_i^{(k)}) \\
&= \sum_{i=1}^{n} \frac{1}{1 + \exp(y_i(\mathbf{w}^\top \mathbf{x}_i + b))} (-y_i x_i^{(k)}) \\
= \sum_{i|y_i=1} \frac{1}{1 + \exp(\mathbf{w}^\top \mathbf{x}_i + b)} (-x_i^{(k)}) &+ \sum_{i|y_i=-1} \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_i + b)} (x_i^{(k)}).
\end{aligned}$$

The factors $1/(1 + \exp(\mathbf{w}^\top \mathbf{x}_i + b))$ and $1/(1 + \exp(-\mathbf{w}^\top \mathbf{x}_i + b))$ are the probability of incorrect prediction. (cf. (3.4))

Thus, when we take a step in the direction of the negative gradient, we go in the 'opposite direction' of these mistakes. This is the reason why gradient descent methods are also called *mistake driven methods*. Mistakes of the current model (here given by

some weights $(\mathbf{w}, b)$) are used to improve it. The gradient points in the direction to best minimize mistakes in the current model.

In summary:
Logistic regression is a supervised classification method where the decision function is affine and the loss is measured with $L(y, f(\mathbf{x})) = \log(1 + e^{-yf(\mathbf{x})})$. The best parameters $\mathbf{w}^\star, b^\star$ are found by minimizing the empirical expected loss, i.e. $\min_{\mathbf{w},b} \frac{1}{N} \sum \log(1 + e^{-y_i(\mathbf{w}^\top \mathbf{x}_i + b)})$. Once the optimal $\mathbf{w}^\star, b^\star$ are determined, we can classify a new data point $\mathbf{x}_{\text{new}}$ by computing $\text{sign}(\mathbf{w}^{\star\top} \mathbf{x}_{\text{new}} + b^\star)$. We can also compute the probability that this classification is correct via Eq. (3.4).

3.1 Alternative approach to logistic regression

First, note that the name logistic regression can be misleading, since in fact logistic regression is not a regression method, but a classification method. While the previous section was more optimization-motivated, here we provide a more statistical approach to logistic regression.

Example: We try to predict the probability of death given some parameters. Let x_1 be the age of a person, x_2 the gender (0 corresp. to male, 1 corresp. to female), and x_3 the cholesterol level. We assume that we can combine these values in a linear fashion so get a real value that is in some way correlated to the probability of death

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 = \mathbf{w}^\top \mathbf{x}$$

with $\mathbf{x} = [1, x_1, x_2, x_3]^\top$, $\mathbf{w} = [w_0, w_1, w_2, w_3]^\top$. The values w_i are called weights, w_0 is called the offset. The resulting value is in $\mathbb{R}$. To shape this into a probability, we need a function σ that squashes this value into the interval $[0, 1]$. A function that achieves this is the logistic function

$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{3.5}$$

The resulting model is $P(\text{death}|x) = \sigma(\mathbf{w}^\top \mathbf{x})$.

More general, we consider the training data for a binary classifiaction problem $D = \{(\mathbf{x}_1, z_1), \ldots, (\mathbf{x}_n, z_n)\}$, $\mathbf{x}_i \in \mathbb{R}^d$, $z_i \in \{0, 1\}$ and model the dependency of input and output variable as $z_i \propto \text{Bernoulli}(\sigma(\mathbf{w}^\top \mathbf{x}))$, where we assume the z_i to be independent.

To train this model, we want to find the maximum likelihood estimate of $\mathbf{w}$ given D, i.e.

$$\mathbf{w}_{\text{MLE}} = \arg\max_{\mathbf{w}} Pr(D|\mathbf{w})$$

with

$$Pr(D|\mathbf{w}) = \prod_{i=1}^{n} Pr(z_i|\mathbf{x}_i, \mathbf{w}) = \prod_{i=1}^{n} \sigma(\mathbf{w}^\top \mathbf{x}_i)^{z_i} (1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))^{1-z_i}. \tag{3.6}$$

For optimization purposes, it is common to use the negative log of the above conditioned probability, namely

$$L(\mathbf{w}) = -\log Pr(D|w) = -\sum_{i=1}^{n} z_i \log(\sigma(\mathbf{w}^\top \mathbf{x}_i)) + (1 - z_i)\log(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i)). \tag{3.7}$$

The respective gradient w.r.t. $\mathbf{w}$ is given by

$$\mathbf{g} = \nabla_{\mathbf{w}} L(\mathbf{w}) = \sum_{i=1}^{n} (\sigma(\mathbf{w}^\top \mathbf{x}_i) - z_i)\mathbf{x}_i = \mathbf{X}(\sigma(\mathbf{X}^\top \mathbf{w}) - \mathbf{z}) \tag{3.8}$$

with $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$. The corresponding Hessian w.r.t. $\mathbf{w}$ is given as

$$\mathbf{H} = \nabla^2_{\mathbf{w}} L(\mathbf{w}) = \mathbf{X}\mathbf{B}\mathbf{X}^\top \tag{3.9}$$

with $\mathbf{B} = \operatorname{diag}(\sigma(\mathbf{w}^\top \mathbf{x}_i)(1 - \sigma(\mathbf{w}^\top \mathbf{x}_i))) \in \mathbb{R}^{n \times n}$. The Hessian is positive semi-definite (Task: show this) and therefore L is convex.

Assume that the Hessian is invertible. Then the iteration of Newton's method has the form

$$\begin{aligned} \mathbf{w}_{t+1} =& \mathbf{w}_t - \mathbf{H}^{-1}\mathbf{g} \\ =& \mathbf{w}_t - (\mathbf{X}\mathbf{B}\mathbf{X}^\top)^{-1}\mathbf{X}(\sigma(\mathbf{X}^\top \mathbf{w}) - \mathbf{z}) \\ =& (\mathbf{X}\mathbf{B}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{B}\mathbf{r}_t \end{aligned} \tag{3.10}$$

with $\mathbf{r}_t = \mathbf{X}^\top \mathbf{w}_t - \mathbf{B}^{-1}(\sigma(\mathbf{X}^\top \mathbf{w}_t) - \mathbf{z})$. This is the solution to the weighted least squares problem $\arg\min_{\mathbf{w}} \sum_i b_i (r_i - \mathbf{w}^\top \mathbf{x}_i)^2$.

Exercise. Show, that Equation (3.7) is equivalent to Equation (3.3) up to the scalar factor $1/n$.

Proof. First, note that $\log(\sigma(\mathbf{w}^\top \mathbf{x})) = -\log(1 + \exp(-\mathbf{w}^\top \mathbf{x}))$ and $\log(1 - \sigma(\mathbf{w}^\top \mathbf{x})) = -\log(1 + \exp(\mathbf{w}^\top \mathbf{x}))$. Thus, Equation (3.7) is equivalent to

$$L(\mathbf{w}) = \sum_{i=1}^{n} z_i \log(1 + \exp(-\mathbf{w}^\top \mathbf{x}_i)) + (1 - z_i)\log(1 + \exp(\mathbf{w}^\top \mathbf{x}_i)). \tag{3.11}$$

Since the z_i are either 0 or 1, we can rewrite the sum as

$$L(\mathbf{w}) = \sum_{z_i=1} \log(1 + \exp(-\mathbf{w}^\top \mathbf{x}_i)) + \sum_{z_i=0} \log(1 + \exp(\mathbf{w}^\top \mathbf{x}_i)). \tag{3.12}$$

Now, comparing the labels z_i with the labels y_i in (3.3), we see that $z_i = 1 \Leftrightarrow y_i = 1$ and $z_i = 0 \Leftrightarrow y_i = -1$, so

$$\begin{aligned} L(\mathbf{w}) &= \sum_{y_i=1} \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)) + \sum_{y_i=-1} \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)) \\ &= \sum_{i=1}^{n} \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)), \end{aligned} \tag{3.13}$$

which coincides with Equation (3.3) up to the factor $1/n$. □

3.2 Linear Separability and Logistic Regression

It is important to be aware that logistic regression can overfit on linearly separable training sets. When the two classes 1 and -1 are linearly separable, we can find a hyperplane $(\mathbf{w}_s, b_s)$ such that the following inequalities hold.

$$y_i(\mathbf{w}_s^\top \mathbf{x}_i + b_s) > 0 \quad \forall i. \tag{3.14}$$

Consider the following theorem.

Theorem 3.4. *For linearly separable, non-empty training sets, the loss function*

$$F(\mathbf{w}, b) = \sum_{i=1}^{n} \log(1 + \exp(-y_i(\mathbf{w}^\top \mathbf{x}_i + b))) \tag{3.15}$$

has no global minimum in $\mathbb{R}^{p+1}$.

Proof. Let us first characterize a global minimum of F. It is a point $(\mathbf{w}^*, b^*) \in \mathbb{R}^{p+1}$ such that

$$F(\mathbf{w}, b) \geq F(\mathbf{w}^*, b^*) \quad \forall (\mathbf{w}, b) \in \mathbb{R}^{p+1} \tag{3.16}$$

holds. If the training set is non-empty, then the loss function is strictly positive. Therefore, the minimal value of F is some positive number ε, i.e.

$$F(\mathbf{w}^*, b^*) = \varepsilon > 0. \tag{3.17}$$

We will now show the lack of the global minimum by contradiction. Assume that there is a point $(\mathbf{w}_s, b_s)$ and a real number $\varepsilon > 0$ such that the following to conditions hold.

$$y_i(\mathbf{w}_s^\top \mathbf{x}_i + b_s) > 0 \ \forall i \quad \text{and} \quad F(\mathbf{w}, b) \geq \varepsilon \ \forall (\mathbf{w}, b) \in \mathbb{R}^{p+1}. \tag{3.18}$$

For each i, define the following scalar value

$$\xi_i = y_i(\mathbf{w}_s^\top \mathbf{x}_i + b_s). \tag{3.19}$$

Consider the function

$$f_i(h) = \log(1 + \exp(-h\xi_i)). \tag{3.20}$$

It is easy to see that ξ_i is strictly positive for any i and therefore $f_i(h)$ approaches 0 as h approaches ∞. The same is true if we consider the sum over i, i.e.

$$\lim_{h \to \infty} \sum_{i=1}^{n} f_i(h) = F(h\mathbf{w}_s, hb_s) = 0. \tag{3.21}$$

In words, for any $\varepsilon > 0$, we can find a real number $\eta > 0$, such that the following inequality holds for any $h \geq \eta$.

$$F(h\mathbf{w}_s, hb_s) < \varepsilon. \tag{3.22}$$

This directly contradicts the assumption (3.18), because we can choose $\mathbf{w} = h\mathbf{w}_s$ and $b = hb_s$ with $h \geq \eta$. □

The proof shows that, once a separating hyperplane is found, the value of the loss function can be always further decreased by increasing the magnitude of the hyperplane parameters. Note that this is true for *any* hyperplane that separates the classes. In practice, an optimization algorithm could choose a hyperplane with a "bad" position and orientation and increase the magnitude of the parameters until the maximum number of iterations is reached.

To prevent this scenario, it is common to penalize the magnitude of $(\mathbf{w}, b)$ by introducing a regularizer, e.g. by fixing a real constant $\lambda > 0$ and adjusting the original cost function as

$$\tilde{F}(\mathbf{w}, b) = F(\mathbf{w}, b) + \lambda(\|\mathbf{w}\|^2 + b^2). \tag{3.23}$$

4 Principal Component Analysis and Its Modern Interpretations

Unsupervised Learning Methods: Input variable is $X \in \mathbb{R}^n$, output variable is the *reduced variable* $S \in \mathbb{R}^k$. S is computed from X via some map f that minimizes a particular Loss function $L(f(X))$. Loss functions in unsupervised learning aim at reducing "the volume of the distribution of the input variable". By this, we mean that samples of $f(X)$ concentrate in a much smaller volume than samples of X. In many cases, this is done by doing *unsupervised dimensionality reduction*, i.e. by choosing k smaller than n. Also clustering, where S consists of finitely many cluster centers, reduces the volume of the input distribution.

Among all unsupervised dimensionality reduction techniques Principal Component Analysis (PCA) is the most famous one. We refer to unsupervised learning methods, if the data does not have to be labeled (by a *supervisor*) before we employ it for the learning algorithm.

The success of PCA is due to its simplicity and broad applicability in many real world data analysis tasks. This may be the reason that it has many aliases, namely the discrete Karhunen–Loéve transform, the Hotelling transform or proper orthogonal decomposition. Its core assumption is that the distribution of the raw data is concentrated around some lower dimensional plane, or, equivalently, that most of the variance in the data can be described by the variance of its projection onto this plane.

4.1 A geometric interpretation

The geometric interpretation of PCA is that it reduces the dimension by projecting the centered data onto a lower dimensional subspace and describes it with coordinates of an appropriate basis of that subspace. Figure 4.1 illustrates this interpretation.

We can formalize this task as follows. Let $k < p$ (usually $k \ll p$) and $\mathbf{X} = \begin{bmatrix}\mathbf{x}_1, \dots, \mathbf{x}_n\end{bmatrix} \in \mathbb{R}^{p\times n}$ be the centered observation matrix. Recall, that centered means $\sum_i \mathbf{x}_i = 0$. The orthogonal[1] projection $\pi_{\mathcal{U}} \colon \mathbb{R}^p \to \mathbb{R}^p$ onto a k-dimensional subspace $\mathcal{U} \subset \mathbb{R}^p$ can be described with help of an orthogonal basis of $\mathcal{U}$. More precisely, if we denote this basis by the matrix $\mathbf{U}_k \in \mathbb{R}^{p\times k}$ (i.e. with orthonormal columns $\mathbf{U}_k^\top \mathbf{U}_k = \mathbf{I}_k$) the projection is given by

$$\pi_{\mathcal{U}}(\mathbf{x}) = \mathbf{U}_k \mathbf{U}_k^\top \mathbf{x}. \tag{4.1}$$

[1] If nothing else is mentioned, *orthogonal* is always with respect to the standard scalar product.

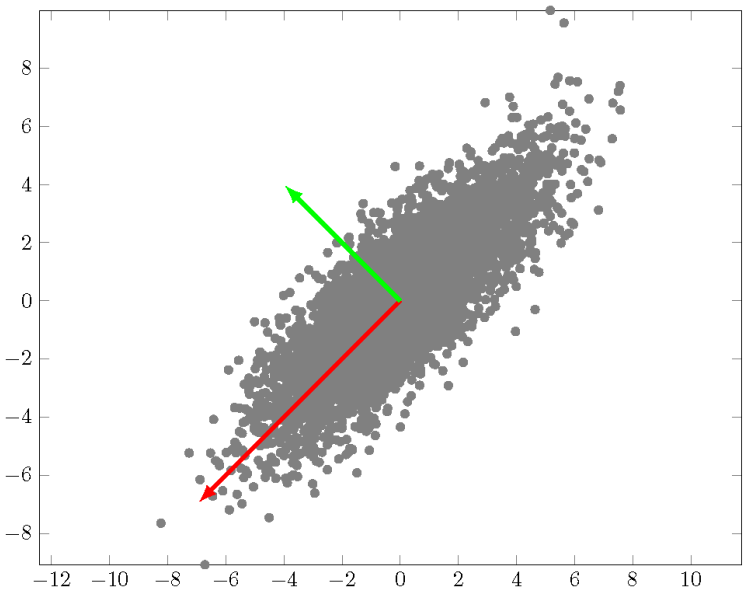

Figure 4.1: Geometric interpretation of PCA: Find a k-dimensional subspace that captures most of the variance of the data. The first PC is displayed in red, while the second is shown in green.

We thus can formulate the task as to find a matrix $\mathbf{U}_k \in \mathbb{R}^{p\times k}$ with orthonormal columns that minimizes the sum of squared projection errors

$$J(\mathbf{U}_k) = \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{U}_k\mathbf{U}_k^\top \mathbf{x}_i\|_2^2. \tag{4.2}$$

The following result states that such a matrix can easily[2] be found.

Theorem 4.1. *Let the singular value decomposition of the centered observation matrix* $\mathbf{X}$ *be given as*

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top, \tag{4.3}$$

with singular values $d_1 \geq \cdots \geq d_n \geq 0$. *If we denote by* $\mathbf{U}_k$ *the first* k *columns of* $\mathbf{U}$, *then* $\mathbf{U}_k$ *minimizes* (4.2). *Moreover, the empirical covariance matrix of the reduced* k *variables* $\mathbf{S} := \mathbf{U}_k^\top \mathbf{X}$ *is diagonal, i.e. the features which are extracted by PCA are uncorrelated.*

[2]The reason why we can state that this matrix can *easily been found* is due to the fact that researchers have developed efficient and reliable algorithms in the past decades for computing the Singular Value Decomposition of a matrix. It does not mean that in general such a matrix can be computed by hand.

Proof. First, Let $\mathbf{Q}_k \in \mathbb{R}^{p\times k}$ with $\mathbf{Q}_k^\top \mathbf{Q}_k = \mathbf{I}_k$. Note that since $\|\mathbf{x}_i - \mathbf{Q}_k\mathbf{Q}_k^\top \mathbf{x}_i\|_2^2 = \|\mathbf{x}_i\|_2^2 - \mathbf{x}_i^\top \mathbf{Q}_k\mathbf{Q}_k^\top \mathbf{x}_i$, minimizing J is equivalent to maximizing $\sum_i \mathbf{x}_i^\top \mathbf{Q}_k\mathbf{Q}_k^\top \mathbf{x}_i$. Then

$$\begin{aligned}\sum_{i=1}^{n} \mathbf{x}_i^\top \mathbf{Q}_k\mathbf{Q}_k^\top \mathbf{x}_i &= \operatorname{tr}(\mathbf{Q}_k^\top \mathbf{X}\mathbf{X}^\top \mathbf{Q}_k)\\ &= \operatorname{tr}(\mathbf{Q}_k^\top \mathbf{U}\mathbf{D}\mathbf{D}^\top \mathbf{U}^\top \mathbf{Q}_k)\\ &= \operatorname{tr}\left(\mathbf{U}^\top \mathbf{Q}_k\mathbf{Q}_k^\top \mathbf{U}^\top \begin{bmatrix} d_1^2 & & \\ & \ddots & \\ & & d_n^2 \end{bmatrix}\right).\end{aligned}$$

The $n \times n$ matrix $\mathbf{U}^\top \mathbf{Q}_k\mathbf{Q}_k^\top \mathbf{U}$ is a rank k projection matrix, hence its diagonal entries are all between 0 and 1. To see this, let $\mathbf{P}$ be an orthogonal projection matrix. Its i-th diagonal entry is $\mathbf{e}_i^\top \mathbf{P}\mathbf{e}_i$, where $\mathbf{e}_i$ is the i-th standard basis vector. Since $\mathbf{P}$ is an orthogonal projection matrix, we have

$$\mathbf{e}_i^\top \mathbf{P}\mathbf{e}_i = \mathbf{e}_i^\top \mathbf{P}^2\mathbf{e}_i = \|\mathbf{P}\mathbf{e}_i\|^2. \tag{4.4}$$

Since $\mathbf{P}$ is an orthogonal projection, $(\mathbf{I}-\mathbf{P})$ is the projection onto the orthogonal complement and every vector in $\mathbb{R}^n$ can be written as a unique composition of elements from the projection spaces. In particular, this is true for standard basis vectors, i.e.

$$\begin{aligned}1 =&\|\mathbf{e}_i\|^2 = \|\mathbf{P}\mathbf{e}_i + (\mathbf{I}-\mathbf{P})\mathbf{e}_i\|^2\\ =&\|\mathbf{P}\mathbf{e}_i\|^2 + \|(\mathbf{I}-\mathbf{P})\mathbf{e}_i\|^2 + 2\langle \mathbf{P}\mathbf{e}_i, (\mathbf{I}-\mathbf{P})\mathbf{e}_i\rangle\\ =&\|\mathbf{P}\mathbf{e}_i\|^2 + \|(\mathbf{I}-\mathbf{P})\mathbf{e}_i\|^2.\end{aligned} \tag{4.5}$$

The last equality holds since $\mathbf{P}\mathbf{e}_i$ and $(\mathbf{I}-\mathbf{P})\mathbf{e}_i$ are orthogonal to one another. The summands $\|\mathbf{P}\mathbf{e}_i\|^2$ and $\|(\mathbf{I}-\mathbf{P})\mathbf{e}_i\|^2$ are both positive and add up to 1. Therefore, $\|\mathbf{P}\mathbf{e}_i\|^2$ lies in the interval $[0,1]$ for all i. Besides, the trace of $\mathbf{P}$, i.e. the sum of all diagonal elements is equal to k. So the maximum value we can achieve is to set $\mathbf{Q}_k = \mathbf{U}_k$, for then this projector is $\mathbf{U}^\top \mathbf{U}_k\mathbf{U}_k^\top \mathbf{U} = \begin{bmatrix} \mathbf{I}_k & 0 \\ 0 & 0 \end{bmatrix}$.

To see that the reduced variables $\mathbf{S} := \mathbf{U}_k^\top \mathbf{X}$ are uncorrelated, note first that since $\mathbf{X}$ is centered, so is $\mathbf{S}$, and therefore its empirical covariance matrix is given by

$$\operatorname{cov}(\mathbf{S}) := \frac{1}{n}\mathbf{S}\mathbf{S}^\top = \frac{1}{n}\mathbf{U}_k^\top \mathbf{X}\mathbf{X}^\top \mathbf{U}_k = \frac{1}{n}\begin{bmatrix} d_1^2 & & \\ & \ddots & \\ & & d_k^2 \end{bmatrix}. \tag{4.6}$$

The (i,j)-entry of the matrix $\mathbf{S}$ is called the *j-th score* for the *i-th principal component*, the complete matrix is called *score matrix*. The matrix $\mathbf{U}$ is also often referred to as

loadings matrix. [3] Note, that $\mathbf{S}$ can directly be obtained by the SVD of $\mathbf{X}$, since

$$\mathbf{S} = \mathbf{U}_k^\top \mathbf{U}\mathbf{D}\mathbf{V}^\top = \begin{bmatrix} d_1 & & \\ & \ddots & \\ & & d_k \end{bmatrix} \mathbf{V}_k^\top = \mathbf{D}_k \mathbf{V}_k^\top, \tag{4.7}$$

where $\mathbf{V}_k$ stands for the first k columns of $\mathbf{V}$. □

While the reduced variables can be computed via $\mathbf{U}_k^\top \mathbf{X}$ this is not feasible in practical applications since both $\mathbf{U}_k$ and $\mathbf{X}$ are dense ($(2p-1)nk$ operations required to compute $\mathbf{S}$). Using $\mathbf{D}_k \mathbf{V}_k^\top$ is much cheaper since $\mathbf{D}_k$ is diagonal and $\mathbf{V}_k^\top$ only has k rows (nk operations required to compute $\mathbf{S}$).

4.2 Statistical Interpretation

Other than minimizing the reconstruction error $J(\mathbf{U}_k)$, as defined in (4.2), PCA can be interpreted from a statistical point of view. If $X \in \mathbb{R}^p$ denotes a multidimensional random variable, PCA seeks for an orthogonal transformation to a new coordinate system, i.e.

$$Y = \mathbf{U}^\top X \tag{4.8}$$

with $\mathbf{U}^\top \mathbf{U} = I_p$, such that the covariance matrix of Y is diagonal and that the variances of the components decreases, i.e. $\operatorname{var}(Y_1) \geq \cdots \geq \operatorname{var}(Y_p)$.

Since the covariance matrix $\operatorname{var}(X) = \mathbb{E}[(X-\mu)(X-\mu)^\top]$, with $\mu = \mathbb{E}[X]$, is symmetric positive semidefinite, it can be diagonalized and ordered by an orthogonal matrix $\mathbf{U}$, i.e.

$$\begin{aligned} \mathbf{D} =& \mathbf{U}^\top \operatorname{var}(X)\mathbf{U} = \mathbb{E}[\mathbf{U}^\top (X-\mu)(X-\mu)^\top \mathbf{U}] \\ =& \mathbb{E}[(\mathbf{U}^\top X - \mathbf{U}^\top \mu)(\mathbf{U}^\top X - \mathbf{U}^\top \mu)^\top] = \operatorname{var}(Y) \end{aligned} \tag{4.9}$$

is diagonal with decreasing, positive entries. If we consider the *empirical* covariance matrix instead, then $\mathbf{U}$ is given by the SVD of the centered observation matrix as the left-singular vectors, cf. (4.3).

4.3 Error Model Interpretation

Consider the observed data $\mathbf{X}$ as the superposition of some *clean* data $\mathbf{L}$ that lies in some k-dimensional subspace and some *additional noise* $\mathbf{N}$, i.e.

$$\mathbf{X} = \mathbf{L} + \mathbf{N}. \tag{4.10}$$

[3] The definition of the loadings matrix is not consistent in the literature. Sometimes, the matrix $\mathbf{L} := \mathbf{U}\mathbf{D}$ is called the loading matrix, and $\mathbf{U}$ is denoted as the *matrix of principal axes/directions*.

Formally, requiring that the data $\mathbf{L}$ lies in some k-dimensional subspace is equivalent to requiring that the rank of $\mathbf{L}$ is lower or equal to k. We shall see below that another way of looking at PCA is to recover $\mathbf{L}$ given that the noise $\mathbf{N}$ is (entry-wise) i.i.d. Gaussian, i.e. each entry of the matrix $\mathbf{N}$ is drawn independently according to the Gaussian distribution with mean 0 and variance 1. Under this model assumption, the maximum likelihood estimation of $\mathbf{L}$ given the observations $\mathbf{X}$ is

$$\hat{\mathbf{L}} = \arg \min_{\operatorname{rank} \mathbf{L} \leq k} \|\mathbf{X} - \mathbf{L}\|_F. \tag{4.11}$$

Classical PCA provides the solution of this problem.

Theorem 4.2. *Let $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ be the singular value decomposition of the observation matrix $\mathbf{X}$ with singular values $d_1 \geq \cdots \geq d_p \geq 0$. If we denote by $\mathbf{U}_k$ the first k columns of $\mathbf{U}$ and by $\mathbf{V}_k$ the first k columns of $\mathbf{V}$, then $\hat{\mathbf{L}} = \mathbf{U}_k \operatorname{diag}(d_1, \ldots, d_k)\mathbf{V}_k^\top$ minimizes (4.11).*

Proof. Let $\mathbf{L}$ be a minimizer of (4.11). Observe that the rank of a matrix $\mathbf{L}$ is lower or equal to k if and only if its columns $\mathbf{l}_i$ lie in a k-dimensional subspace, say $\mathcal{L}$. Now each column $\mathbf{l}_i$ has to be the projection of $\mathbf{x}_i$ onto $\mathcal{L}$, since otherwise, one could replace the i-th column of $\mathbf{L}$ without increasing $\|\mathbf{X} - \mathbf{L}\|_F^2 = \sum_i \|\mathbf{x}_i - \mathbf{l}_i\|^2$. Thus,

$$\min_{\operatorname{rk}\mathbf{L} \leq k} \|\mathbf{X} - \mathbf{L}\|_F^2 = \min_{\dim(\mathcal{L}) = k} \sum_i \|\mathbf{x}_i - \pi_{\mathcal{L}}(\mathbf{x}_i)\|^2, \tag{4.12}$$

and we know from Theorem 4.1, that $\mathcal{L}$ is spanned by the first k singular vectors $\mathbf{U}_k$ of $\mathbf{X}$. Thus the best rank-k approximation is given by

$$\hat{\mathbf{L}} = \mathbf{U}_k \mathbf{U}_k^\top \mathbf{X} = \mathbf{U}_k \operatorname{diag}(d_1, \ldots, d_k)\mathbf{V}_k^\top. \tag{4.13}$$

□

4.4 Relation to Autoencoders

PCA is closely related to a particularly simple form of neural networks, the so-called *autoencoders*. The aim of an autoencoder is to reconstruct the inputs as good as possible after they have been passed through a lower dimensional space, or, more formally: Realizations $x_1, \ldots, x_n$ of a p-dimensional random variable are first mapped to a lower dimensional space $\mathbb{R}^k$ using a function f, and these images are then mapped to $\mathbb{R}^p$ again, trying to best fit the original input

$$g \circ f(\mathbf{x}_i) \approx \mathbf{x}_i. \tag{4.14}$$

Autoencoders try to find the best pair of functions (f, g) for this job, and the idea is that a reasonable part of the interesting information is contained in the reduced data $f(\mathbf{x}_i)$.

Let us now assume that f and g are linear and represented by matrices $\mathbf{V} \in \mathbb{R}^{k \times p}$ and $\mathbf{W} \in \mathbb{R}^{p \times k}$. Then $g \circ f(\mathbf{x}_i) = \mathbf{W}\mathbf{V}\mathbf{x}_i$, and if we measure the reconstruction error via the sum of squared distances, i.e.

$$J(\mathbf{W}, \mathbf{V}) = \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{W}\mathbf{V}\mathbf{x}_i\|^2 \tag{4.15}$$

one can show that the optimal $\mathbf{V}$ is just given by the first k left singular vectors of the observation matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$.

Theorem 4.3. *Let $\mathbf{U}_k$ be the first k left singular vectors of the observation matrix $\mathbf{X}$. Then $\mathbf{V} = \mathbf{U}_k^\top$ and $\mathbf{W} = \mathbf{U}_k$ minimize the reconstruction error of the linear auotencoder (4.15).*

Proof. Due to the dimensions of $\mathbf{V}$ and $\mathbf{W}$, the squared matrix $\mathbf{W}\mathbf{V}$ can at most have a rank of k. So all $\mathbf{W}\mathbf{V}\mathbf{x}_i$ lie in a k-dimensional subspace and thus form a matrix of at most rank k. We have seen in Theorem 4.2 that the minimum of L is achieved by $\mathbf{V} = \mathbf{U}_k^\top$ and $\mathbf{W} = \mathbf{U}_k$. □

There are interesting extensions of this approach by allowing nonlinear functions. Very successful in practical applications are for example functions of the form $f(\mathbf{x}) := \sigma(\mathbf{V}\mathbf{x})$, where $\mathbf{V}$ is a matrix as in the above case and σ is an activation function that operates component-wise and is zero for negative arguments while leaving positive arguments unchanged. In such nonlinear settings, there are no closed form solutions to the resulting optimization problem, and we need techniques from optimization, like e.g. gradient descent methods, to approximate an optimal solution.

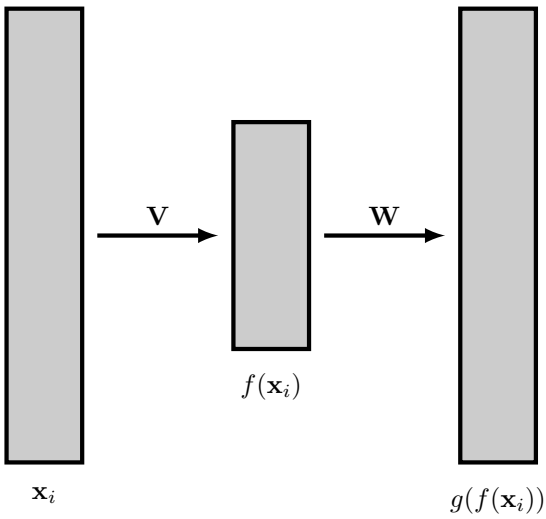

Figure 4.2: A simple autoencoder with $\mathbf{V} \in \mathbb{R}^{k \times p}$, $\mathbf{W} \in \mathbb{R}^{p \times k}$, $k < p$.

5 (Deep) Feedforward Neural Networks

5.1 Definition and Motivation of FNNs

Roughly speaking, Feedforward Neural Networks (FNN) are particular function classes that are very powerful in minimizing any kind of expected loss, at the price of training a huge amount of parameters. To be more precise, consider an input variable $\mathcal{X} \in \mathbb{R}^p$ and a function class $\mathcal{F}$ out of which we want to find a function f that minimizes the expectation of a certain loss function L. Consider for example the simple Loss function $L(f(\mathcal{X})) = \|f(\mathcal{X}) - \mathcal{X}\|_2^2$ for autoencoders that aims at reconstructing samples of $\mathcal{X}$. Here, f consists of the concatenation of an encoder function (that maps into a low dimensional space) and a decoder function (mapping back into the raw data space).

Another example from supervised Learning is regression, where we have a joint distribution $(\mathcal{X}, \mathcal{Y})$ of input and output variables and the aim is to find the best $f \in \mathcal{F}$ that minimizes the expectation of $L(f(\mathcal{X}), \mathcal{Y}) = \|f(\mathcal{X}) - \mathcal{Y}\|_2^2$. We discuss multiclass classification subsequent to this section.

If all functions in $\mathcal{F}$ can be described by a set of parameters, say $\Theta \in \mathbb{R}^N$, and if some samples, say n, for training are given, then these learning problems result in a minimization process

$$\hat{\Theta} = \arg\min_{\Theta \in \mathbb{R}^N} \frac{1}{n} \sum_i L(f_\Theta(\mathbf{x}_i)). \tag{5.1}$$

A very important class of functions in many applications are the so-called *feed-forward neural networks (FNN)*. FNNs are concatenations of linear functions[1]

$$\varphi_{\mathbf{W}} \colon \mathbb{R}^p \to \mathbb{R}^m, \mathbf{h} \mapsto \mathbf{W}\mathbf{h} \tag{5.2}$$

followed by so called *activation functions* that operate component-wise on a vector. Examples for activation functions are the *Rectified Linear Unit ReLU*

$$\sigma(t) := \max\{0, t\}, \tag{5.3}$$

and others, cf. here Wikipedia. By a slight abuse of notation, we also denote the vector activation function by

$$\sigma \colon \mathbb{R}^m \to \mathbb{R}^m, \quad \mathbf{x} \mapsto \begin{bmatrix} \sigma(x_1) \\ \vdots \\ \sigma(x_m) \end{bmatrix}. \tag{5.4}$$

A feedforward neural network is a function

$$f \colon \mathbb{R}^p \to \mathbb{R}^o, \quad \mathbf{x} \mapsto \sigma_l \circ \varphi_{\mathbf{W}_l} \circ \cdots \circ \sigma_1 \circ \varphi_{\mathbf{W}_1}(\mathbf{x}), \tag{5.5}$$

[1] This setting also includes *affine* functions, since we may simply append the additional component 1 to our input vector.

where as usual $\circ$ denotes the concatenation of functions and different function classes $\mathcal{F}$ are defined by different activation function, the number of the *layers* l and the dimensions of the matrices $\mathbf{W}_i \in \mathbb{R}^{n_i \times m_i}$. One usually calls such a FNN *deep* if the number of layers is greater than three. Note, that the output dimension o is determined by the loss function, since the output of the FNN serves as input to the loss.

5.2 Training FNNs

Training FNNs in practice is an art in itself, and there are many tricks and regularization techniques that decide on failure or success of learning a powerful FNN. In this section, we focus on the fundamental method of finding the optimal FNN for the general problem (5.1), i.e. the optimal weights $\hat{\Theta} = (\hat{\mathbf{W}}_1, \dots \hat{\mathbf{W}}_L)$ within a given class of FNNs. It is in fact, a gradient descent method that iteratively updates the weights and the method is known as *backpropagation* in the literature. In the following, we describe how it works in principal.

The most important tool that we need here from our undergrad math courses is *the Chain rule and Jacobi matrices.* Recall, that if $g\colon \mathbb{R}^k \to \mathbb{R}^l$ and $h\colon \mathbb{R}^l \to \mathbb{R}^m$ are two functions with g being differentiable at $\mathbf{x}$ and h differentiable at $\mathbf{y} = g(\mathbf{x})$ with Jacobi matrices $\mathbf{J}_g(\mathbf{x})$ and $\mathbf{J}_h(\mathbf{y})$, then the function $h \circ g\colon \mathbb{R}^k \to \mathbb{R}^m$ is differentiable at $\mathbf{x}$ with Jacobi matrix $\mathbf{J}_{h\circ g}(\mathbf{x}) = \mathbf{J}_h(g(\mathbf{x})) \cdot \mathbf{J}_g(\mathbf{x})$.

Examples.

- (*Linearity of derivatives.*) If in the above setting, h is a simple linear transformation given by matrix multiplication with, say $\mathbf{W}$, then

$$\mathbf{J}_{\mathbf{W}\cdot g}(\mathbf{x}) = \mathbf{W} \cdot \mathbf{J}_g(\mathbf{x}) \tag{5.6}$$

- Since the i-th output only depends on x_i, the Jacobi-matrix of σ in (5.4) is a squared diagonal matrix of the form

$$\mathbf{J}_\sigma(\mathbf{x}) = \begin{bmatrix} \sigma'(x_1) & & \\ & \ddots & \\ & & \sigma'(x_m) \end{bmatrix}. \tag{5.7}$$

- In order to define the Jacobi matrix of the function *multiplication with a vector from the right*

$$\mathrm{mult}(\mathbf{x})\colon \mathbb{R}^{m\times n} \to \mathbb{R}^m, \quad \mathbf{W} \mapsto \mathbf{W}\mathbf{x}, \tag{5.8}$$

  we first need to embed the mn variables, here given in matrix structure, into $\mathbb{R}^{mn}$. This can be done in various ways, but if we choose to embed one row after the

other, i.e.

$$\pi\colon \mathbf{W} = \begin{bmatrix} \mathbf{w}_1^\top \\ \vdots \\ \mathbf{w}_m^\top \end{bmatrix} \mapsto \begin{bmatrix} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_m \end{bmatrix} \in \mathbf{R}^{mn}, \tag{5.9}$$

then the Jacobi matrix has the well-arranged form

$$\mathbf{J}_{\mathrm{mult}(\mathbf{x})} = \begin{bmatrix} \mathbf{x}^\top & & \\ & \ddots & \\ & & \mathbf{x}^\top \end{bmatrix} \in \mathbb{R}^{m \times mn}. \tag{5.10}$$

Note, that due to the linearity of $\mathrm{mult}(\mathbf{x})$, the Jacobi matrix does not depend on $\mathbf{W}$.

In order to compute the gradient of the cost function (5.1) with respect to the weights $\Theta = (\mathbf{W}_l, \dots \mathbf{W}_1)$, we note that it is the average of the gradients of the loss function for one input data $\mathbf{x}$, i.e.

$$F(\mathbf{W}_l, \dots, \mathbf{W}_1) := L \circ \sigma_l \circ \varphi_{\mathbf{W}_l} \circ \cdots \circ \sigma_1 \circ \varphi_{\mathbf{W}_1}(\mathbf{x}). \tag{5.11}$$

It is therefore sufficient to compute the gradient of F, which depends on one input signal, and then average over all training data $\mathbf{x}_i$. For convenience, we denote

$$\mathbf{h}_j := \sigma_j \circ \varphi_{\mathbf{W}_j} \circ \cdots \circ \sigma_1 \circ \varphi_{\mathbf{W}_1}(\mathbf{x}) \tag{5.12}$$

the output after the j-th layer of the FNN. For $0 < j < l$, $\mathbf{h}_j$ is called the *j-th hidden layer of the FNN*, $\mathbf{h}_l$ is called *the output layer* and $\mathbf{h}_0 := \mathbf{x}$ is the input to the FNN.

We denote by $\frac{\partial}{\partial \mathbf{W}_j} F \in \mathbb{R}^{1 \times m_j n_j}$ the Jacobi-Matrix[2] of the function

$$\mathbf{W}_j \mapsto F(\mathbf{W}_l, \dots, \mathbf{W}_j, \dots, \mathbf{W}_1). \tag{5.13}$$

Using the chain rule and the examples from the last section, the derivatives with respect to the different weight matrices $\mathbf{W}_i$ are given by

$$\begin{aligned}
\frac{\partial}{\partial \mathbf{W}_l} F =& \mathbf{J}_L(\mathbf{h}_l) \cdot \mathbf{J}_{\sigma_l}(\mathbf{W}_l \mathbf{h}_{l-1}) \cdot \mathbf{J}_{\mathrm{mult}(\mathbf{h}_{l-1})} \\
\frac{\partial}{\partial \mathbf{W}_{l-1}} F =& \mathbf{J}_L(\mathbf{h}_l) \cdot \mathbf{J}_{\sigma_l}(\mathbf{W}_l \mathbf{h}_{l-1}) \cdot \mathbf{W_l} \cdot \mathbf{J}_{\sigma_{l-1}}(\mathbf{W}_{l-1} \mathbf{h}_{l-2}) \cdot \mathbf{J}_{\mathrm{mult}(\mathbf{h}_{l-2})} \\
& \dots \\
\frac{\partial}{\partial \mathbf{W}_j} F =& \mathbf{J}_L(\mathbf{h}_l) \cdot \mathbf{J}_{\sigma_l}(\mathbf{W}_l \mathbf{h}_{l-1}) \cdot \mathbf{W_l} \cdot \mathbf{J}_{\sigma_{l-1}}(\mathbf{W}_{l-1} \mathbf{h}_{l-2}) \cdot \mathbf{J}_{l-1} \cdot \dots \cdot \mathbf{J}_{\mathrm{mult}(\mathbf{h}_{j-1})},
\end{aligned}$$

[2]Since F is real-valued, this is also the transpose of the gradient.

In practice, the initial weights are often chosen at random from a normal distribution, and then individually updated with the above gradients and a step size $\alpha > 0$. Algorithms and methods for step-size selection are beyond the scope of this lecture. Note, that we have to "reshape" the gradients, i.e. the transpose of $\frac{\partial}{\partial \mathbf{W}_j} F$, in matrix form by inverting the matrix embeddings π_j from (5.9). We finally have the update rule

$$\mathbf{W}_j \leftarrow \mathbf{W}_j - \alpha \pi_j^{-1} \left(\frac{\partial}{\partial \mathbf{W}_j} F \right)^\top . \tag{5.14}$$

5.3 Multiclass Classification with FNNs

Multiclass classification considers the problem of assigning an input $\mathcal{X} \in \mathbb{R}^p$ to one out of multiple, say C, classes. We model the problem via the random variable $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X} \in \mathbb{R}^p$ and $\mathcal{Y} \in \{\mathbf{e}_1, \dots, \mathbf{e}_C\}$ is one of the C standard basis vectors in $\mathbb{R}^C$. A realization $\mathbf{y} = \mathbf{e}_c$ means that the event *belongs to class* c is true. This modelling of the output variable is also known as *one-hot-encoding.*

The idea behind multiclass classification with FNNs is that $\mathcal{X}$ serves as input to the network and the output is a vector in $\mathbb{R}^C$ that should reflect the probability of the class distributions given $\mathcal{X}$. More precisely, if $\mathbf{x}$ is a realization of $\mathcal{X}$, and if f denotes the FNN, then the c-th component of the output vector $\mathbf{h}_l := f(\mathbf{x})$ should approximately be the probability that $\mathcal{Y}$ belongs to class c, given $\mathbf{x}$, i.e.

$$\mathbf{h}_l \approx \begin{bmatrix} Pr(\mathcal{Y} = \mathbf{e}_1 | \mathcal{X} = \mathbf{x}) \\ \vdots \\ Pr(\mathcal{Y} = \mathbf{e}_C | \mathcal{X} = \mathbf{x}) \end{bmatrix} . \tag{5.15}$$

The motivation of the last activation function σ_l is thus to output a probability distribution for the C classes, meaning in practice that the entries of the output vector are between 0 and 1, and that they add up to 1. A prominent choice here is the so called *softmax*, given by

$$\sigma \colon \mathbb{R}^C \to \mathbb{R}^C, \quad \begin{bmatrix} a_1 \\ \vdots \\ a_C \end{bmatrix} \mapsto \frac{1}{\sum_c \exp a_c} \begin{bmatrix} \exp a_1 \\ \vdots \\ \exp a_C \end{bmatrix} . \tag{5.16}$$

Exercise: Compute the Jacobi-Matrix of the softmax function.

The loss function that we need for training the network has to measure how well the predicted distribution corresponds to the distribution observed through our training set $(\mathbf{x}_i, \mathbf{y}_i)$, $i = 1, \dots, n$. Note, that these observed distributions are deterministic, meaning that $y_c := Pr(\mathcal{Y} = \mathbf{e}_c | \mathcal{X} = \mathbf{x}_i) = 1$ if $\mathbf{x}_i$ is labelled with class c and 0 else. Generally, one common way of comparing two probability distributions $\mathbb{P} = (p_1, \dots p_C)$

and $\mathbb{Q} = (q_1, \dots q_C)$ over the same underlying set of C events is by using the so called *cross-entropy*, cf. Wikipedia,

$$H(\mathbb{P}, \mathbb{Q}) = -\sum_{c=1}^{C} p_c \log q_c. \tag{5.17}$$

In our case with one distribution being deterministic, this reduces to the loss function

$$L(f(\mathcal{X}), \mathbf{y}_c) = -\log f(\mathcal{X})_c \tag{5.18}$$

where f is a FNN with softmax as output and $f(\mathcal{X})_c$ denotes its c's entry. For training, as usual, we use the empirical expectation of the loss on our training data, leading to the optimization problem

$$(\hat{\mathbf{W}}_l, \dots, \hat{\mathbf{W}}_1) = \arg\min_{(\mathbf{W}_l, \dots, \mathbf{W}_1)} \left\{ -\tfrac{1}{n} \sum_i \log f_{(\mathbf{W}_l, \dots, \mathbf{W}_1)}(\mathbf{x}_i)_{c_i} \right\}. \tag{5.19}$$

6 Kernels and the Kernel Trick

The success of the machine learning algorithms that we have discussed so far all rely on assumption on the distribution of the input data. PCA for example works better, the more the data is distributed around a linear subspace. Or in Linear Discriminant Analysis, where we assume Gaussian distributions of the classes that even have the same Covariance matrix.

One way of extending methods to better take into account other, more complicated, distribution of the input data is to employ the so-called *Kernel Trick*. It allows to generalize all methods that essentially only have standard inner products as input data.

More precisely, consider a ML-algorithm with input data that can either be unlabelled, i.e. $\mathbf{x}_1, ..., \mathbf{x}_n$ or labelled, i.e. $(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)$. Assume moreover that the algorithm actually only uses inner products $\langle \mathbf{x}_i, \mathbf{x}_j \rangle := \mathbf{x}_i^\top \mathbf{x}_j$ of the input data. Then, by replacing $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ with some function $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ that is *a suitable generalization of an inner product* (namely a Kernel, the definition will follow right away!) is called the Kernel Trick, cf. Fig. 6.1. The resulting learning method is typically named by prefixing the term *Kernel*. This trick usually allows to extend methods that are based on linearity assumptions on the distribution of the data to more complex, non-linear distribution.

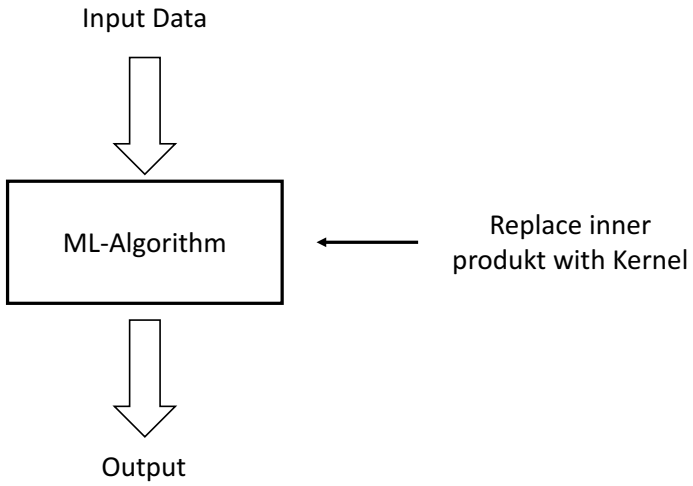

Figure 6.1: Illustration of the Kernel Trick: Replace the standard inner product in a Machine Learning algorithm by a kernel to obtain the 'Kernel'-version of the method.

So here is the definition of a Kernel. It generalizes the standard inner product.

Definition 6.1. Sollten wir hier nicht für allgemeine Mengen Kernel definieren, anstatt

nur $\mathbb{R}^p$?

A (positive semidefinite) Kernel is a function $\kappa\colon \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ such that for all finite sets $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ the $n \times n$-matrix

$$\mathbf{K} := \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_n, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{6.1}$$

is symmetric and positive semidefinite. The matrix $\mathbf{K}$ is called *Gram-Matrix* of κ and $\mathbf{X}$. For a given function κ it is usually difficult to tell whether it is a kernel function or not. Necessary conditions, however, like symmetry and positivity are easily checked.

Example 6.2. The function $\kappa(\mathbf{x}, \mathbf{y}) = \|\mathbf{x}\|\|\mathbf{y}\| - 1$ cannot be a Kernel, since there exists a finite set, namely $\{\mathbf{0}\} \subset \mathbb{R}^p$, such that the associated Gram-Matrix (which is 1×1 in this case) $\mathbf{K} = -1$ is negative definite.

Or consider the function $\kappa(\mathbf{x}, \mathbf{y}) = \mathrm{e}^{-\|\mathbf{x}-\mathbf{y}\|-\|\mathbf{y}\|}$. It is easily seen that in general, $\kappa(\mathbf{x}, \mathbf{y}) \neq \kappa(\mathbf{y}, \mathbf{x})$, and thus it cannot be a Kernel function.

A few common kernels are

- the linear Kernel $\kappa(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y} + c$, $c \geq 0$;
- the polynomial Kernel $\kappa(\mathbf{x}, \mathbf{y}) = (\alpha \mathbf{x}^\top \mathbf{y} + c)^d$, $c, \alpha, d \geq 0$;
- the Gaussian Kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})$, $\sigma > 0$;
- the exponential Kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|}{2\sigma^2})$, $\sigma > 0$;

It follows straightforwardly from the properties of positive semi-definite matrices that, if κ_1 and κ_2 are Kernels, and if $c > 0$, then so are

- $c\kappa_1$
- $c + \kappa_1$
- $\kappa_1 + \kappa_2$
- $\kappa_1 \kappa_2$.

Moreover, for any real valued function $f\colon \mathbb{R}^p \to \mathbb{R}$, we can construct a kernel via $\kappa := f(\mathbf{x}) \cdot f(\mathbf{y})$. Note, that in this case, the corresponding Gram-Matrix has a rank of at most one.

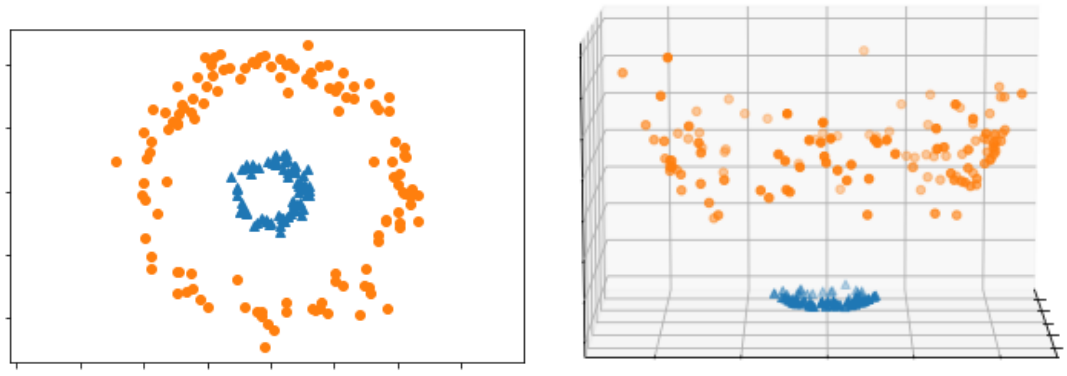

Figure 6.2: Dataset in $\mathbb{R}^2$ and mapped with $\phi\colon \mathbb{R}^2 \to \mathbb{R}^3$, $\phi(x_1, x_2) = [x_1, x_2, x_1^2 + x_2^2]^\top$.

Mercer's Theorem

Theorem 6.3 (Mercer)**.** *For any symmetric function $\kappa\colon \mathcal{X} \times \mathcal{X}$ which is square integrable in $\mathcal{X} \times \mathcal{X}$ and satisfies $\int_{\mathcal{X}\times\mathcal{X}} f(x)\kappa(x,y)f(y)dxdy \geq 0$ for all $f \in L_2(\mathcal{X})$ there exists functions ϕ_i and scalars $\lambda_i \geq 0$ such that*

$$\kappa(x,y) = \sum_i \lambda_i \phi_i(x)\phi_i(y) \qquad \text{for all } x, y \in \mathcal{X}. \tag{6.2}$$

A kernel is a continuous function that takes two variables x, y and maps them to a real value, such that $\kappa(x,y) = \kappa(y,x)$. A kernel is positive semi-definite if and only if $\int\int f(x)\kappa(x,y)f(y)dxdy \geq 0$. In association with a kernel κ, we can define an integral operator T_κ, which, when applied to a function $f(x)$, generates another function:

$$T_\kappa(f(x)) = \int \kappa(x,y)f(y)dy = [T_\kappa f](x).$$

This is a linear function and therefore has eigenvalues λ_i and eigenfunctions $\phi_i(\cdot)$. They are defined as

$$T_\kappa(\phi_i(x)) = \int \kappa(x,y)\phi(y)dy = \lambda_i\phi_i(x)$$

The eigenvalues λ_i are non-negative and the eigenfunctions $\phi_i(x)$ are orthonormal in the sense that $\int \phi_i(x)\phi_j(x)dx = \delta_{ij}$. The eigenfunctions corresponding to non-zero eigenvalues for a set of basis function so that the kernel can be decomposed in terms of them via

$$\kappa(x,y) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x)\phi_i(y). \tag{6.3}$$

7 Kernel Principal Component Analysis

Standard PCA reduces the dimensionality of the observed data by projecting it onto a linear subspace. The projection is chosen in a way that the error measured in the squared standard Euclidean norm is minimized, which can also be interpreted as a way to reduce white Gaussian noise. One very important application is to use PCA as a preprocess for classification, since classifiers perform better in the feature space where this noise is reduced.

The major drawback of standard PCA is that it heavily relies on the approximate linear structure of the data. In many applications, this is a far too strict assumption. Kernel PCA (K-PCA) is an extension of standard PCA that does not have these shortcomings. The crucial idea behind K-PCA is that it implicitly assumes the existence of a nonlinear map

$$\phi\colon \mathbb{R}^p \to \mathcal{H}, \tag{7.1}$$

where $\mathcal{H}$ is a very high dimensional vector space (that could even be infinite dimensional) with an inner product[1] $\langle \cdot, \cdot \rangle$. There is no harm for us to think of $\mathcal{H}$ as some $\mathbb{R}^P$ with very large P. As a preliminary step, we reformulate the well known standard PCA in a way such that it only involves inner products of our data.

7.1 Linear PCA expressed with inner products

Let $\mathbf{X} \in \mathbb{R}^{p \times n}$ be the centered data matrix (this will be a crucial assumption for the following derivations) and let $\mathbf{K} := \mathbf{X}^\top \mathbf{X}$ be the $(n \times n)$-matrix consisting of all inner products of the data. More precisely, the (i, j) entry of $\mathbf{K}$ is the inner product $\mathbf{x}_i^\top \mathbf{x}_j$ of the i-th and the j-th observation.

Recall, that if $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$ is the SVD of the data matrix, then the first k principle components of a data vector $\mathbf{y} \in \mathbb{R}^p$ are given by $\mathbf{U}_k^\top \mathbf{y}$. In the case at hand only the inner products are given and it is thus not possible to obtain $\mathbf{U}_k$ directly. However, the eigenvalue decomposition of $\mathbf{K}$ allows for an elegant solution to this problem. Remember that we have $\mathbf{K} := \mathbf{X}^\top \mathbf{X}$ and substitute $\mathbf{X}$ by its SVD. This yields

$$\mathbf{K} = \mathbf{V}\mathbf{\Sigma}^\top \mathbf{\Sigma}\mathbf{V}^\top. \tag{7.2}$$

Remember that $\mathbf{V}$ is orthogonal and $\mathbf{\Sigma}^\top \mathbf{\Sigma}$ is diagonal. Thus, Equation (7.2) is the eigenvalue decomposition of $\mathbf{K}$, due to the uniqueness of the EVD, and by computing the k largest eigenvalues of $\mathbf{K}$ with their respective eigenvectors, we obtain $\sigma_1^2, \cdots, \sigma_k^2$

[1] Formally, a Hilbert space $\mathcal{H}$ is a real or complex inner product space that is also a complete metric space with respect to the distance function induced by the inner product.

and $\mathbf{V}_k$. We will assume that $\sigma_k > 0$, since otherwise we could reduce the dimension of the targeted subspace without losing any information on our data. For simplicity, we define the diagonal matrix $\mathbf{\Sigma}_k = \operatorname{diag}(\sigma_1, \ldots, \sigma_k)$.

This allows us to to compute the principal components $\mathbf{U}_k^\top \mathbf{y}$ for a given observation $\mathbf{y}$ by only using inner products. For a new measurement $\mathbf{y}$ we have

$$\begin{aligned} \mathbf{V}_k^\top \mathbf{X}^\top \mathbf{y} &= \mathbf{V}_k^\top \mathbf{V} \mathbf{\Sigma} \mathbf{U}^\top \mathbf{y} \\ &= \big[I_k \mid 0 \big] \mathbf{\Sigma} \mathbf{U}^\top \mathbf{y} \\ &= \big[\mathbf{\Sigma}_k \mid 0 \big] \mathbf{U}^\top \mathbf{y} \\ &= \mathbf{\Sigma}_k \mathbf{U}_k^\top \mathbf{y}. \end{aligned} \tag{7.3}$$

By multiplying this equation with $\mathbf{\Sigma}_k^{-1}$, we obtain

$$\mathbf{U}_k^\top \mathbf{y} = \mathbf{\Sigma}_k^{-1} \mathbf{V}_k^\top \mathbf{X}^\top \mathbf{y} = \mathbf{\Sigma}_k^{-1} \mathbf{V}_k^\top \begin{bmatrix} \mathbf{x}_1^\top \mathbf{y} \\ \vdots \\ \mathbf{x}_n^\top \mathbf{y} \end{bmatrix}. \tag{7.4}$$

Note, that the right hand side of this equation can be computed by data that only involves inner products of the data, namely the Gram matrix $\mathbf{K}$ and the inner products $\mathbf{x}_n^\top \mathbf{y}$.

Centering the data. So far, we have assumed that the data is centered, i.e. that the Gram matrix $\mathbf{K}$ arises from *centered* data. This assumption was crucial, since otherwise the derivation for Equation (7.4) would not hold. Now let us assume that we have no centered data available, and that the Gram matrix $\mathbf{K}$ is built from non-centered data. The good news is that it is possible to deduce the Gram matrix that corresponds to the centered data, say $\tilde{\mathbf{K}}$, directly from $\mathbf{K}$ via the formula

$$\tilde{\mathbf{K}} = \mathbf{H}\mathbf{K}\mathbf{H}, \tag{7.5}$$

where $\mathbf{H} = (\mathbf{I}_n - \frac{1}{n} \mathbb{1}_n \mathbb{1}_n^\top)$.

Proof. Recall from Equation (1.3), that if $\mathbf{X}$ denotes the non-centered data matrix, the centered matrix is given by

$$\overline{\mathbf{X}} = \mathbf{X} - \hat{\mu} \mathbb{1}_n^\top, \tag{7.6}$$

with $\hat{\mu} = \frac{1}{n} \mathbf{X} \mathbb{1}_n$. From this, it follows that

$$\overline{\mathbf{X}} = \mathbf{X} - \frac{1}{n} \mathbf{X} \mathbb{1}_n \mathbb{1}_n^\top = \mathbf{X}(\mathbf{I}_n - \tfrac{1}{n} \mathbb{1}_n \mathbb{1}_n^\top). \tag{7.7}$$

With the shorthand notation for the symmetric matrix $\mathbf{H} := \mathbf{I}_n - \frac{1}{n} \mathbb{1}_n \mathbb{1}_n^\top$ we obtain

$$\tilde{\mathbf{K}} = \overline{\mathbf{X}}^\top \overline{\mathbf{X}} = \mathbf{H}\mathbf{X}^\top \mathbf{X}\mathbf{H} = \mathbf{H}\mathbf{K}\mathbf{H}. \tag{7.8}$$

□

So, if we have the Gram matrix from non-centered data, we can easily compute the Gram matrix that corresponds to centered data (without explicitly centering our $\mathbf{X}$). From this, we can - as above - compute the $\mathbf{V}_k$ and the $\mathbf{\Sigma}_k$. In order to compute the principal components for a new data sample $\mathbf{y}$, we first have to center $\mathbf{y}$ w.r.t. the training samples. Concretely, this means that we have to subtract the empirical mean of the training data $\hat{\mu} = \frac{1}{n}\sum_i \mathbf{x}_i = \frac{1}{n}\mathbf{X}\mathbb{1}_n$ from $\mathbf{y}$. Then we have to replace $\mathbf{X}$ in Eqs. (7.3) and (7.4) with $\overline{\mathbf{X}} = \mathbf{X}\mathbf{H}$. This yields

$$\mathbf{U}_k^\top(\mathbf{y} - \tfrac{1}{n}\mathbf{X}\mathbb{1}_n) = \mathbf{\Sigma}_k^{-1}\mathbf{V}_k^\top(\mathbf{X}\mathbf{H})^\top(\mathbf{y} - \tfrac{1}{n}\mathbf{X}\mathbb{1}_n) = \mathbf{\Sigma}_k^{-1}\mathbf{V}_k^\top\mathbf{k}_y \tag{7.9}$$

with

$$\mathbf{k}_y = \mathbf{H}\begin{bmatrix}\mathbf{x}_1^\top\mathbf{y}\\ \vdots \\ \mathbf{x}_n^\top\mathbf{y}\end{bmatrix} - \tfrac{1}{n}\mathbf{H}\mathbf{K}\mathbb{1}_n.$$

7.2 Transition to Kernel PCA

It is now straightforward to extend classical PCA by simply replacing the inner product $\mathbf{x}^\top\mathbf{y}$ by $\langle\phi(\mathbf{x}), \phi(\mathbf{y})\rangle$. The practical success of K-PCA is due to the fact that for computing $\langle\phi(\mathbf{x}), \phi(\mathbf{y})\rangle$, neither ϕ nor the inner product $\langle\cdot,\cdot\rangle$ is explicitly needed. Instead, it is sufficient to have a function

$$\kappa\colon \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}, \quad (\mathbf{x}, \mathbf{y}) \mapsto \kappa(\mathbf{x}, \mathbf{y}) \tag{7.10}$$

that reflects the properties of $\langle\phi(\mathbf{x}), \phi(\mathbf{y})\rangle$, i.e. that is symmetric and fulfills the positivity property $\kappa(\mathbf{x}, \mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathbb{R}^p$. Substituting $\langle\phi(\mathbf{x}), \phi(\mathbf{y})\rangle$ by $\kappa(\mathbf{x}, \mathbf{y})$, and therefore not needing to know the feature mapping ϕ, is called the *Kernel trick*.

Definition 7.1 (Positive Definite Kernel)**.**

1. Let $S := \{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \subset \mathbb{R}^p$ and $\kappa(\cdot)$ as above. The $(n \times n)$-matrix $\mathbf{K}$ with (i, j)-entries $k_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ is called a *Gram-* or *Kernel-matrix* of S with respect to κ.

2. The function $\kappa(\cdot)$ is called a *Kernel (positive definite Kernel)*, if for all finite, nonempty sets $S \subset \mathbb{R}^p$ the corresponding Gram matrix is positive semidefinite (positive definite).

In the previous subsection we discussed the issue of centering the data in the latent Hilbert space. We face the same problem here. When a new data point comes in, it

is straightforward to compute the *uncentered* vector of inner products of the new data point w.r.t. the training data. We will refer to it as

$$\mathbf{k}^{new} = \begin{bmatrix} \langle \phi(\mathbf{x}_1), \phi(\mathbf{y}) \rangle \\ \vdots \\ \langle \phi(\mathbf{x}_n), \phi(\mathbf{y}) \rangle \end{bmatrix}.$$

The j-th entry of the centered data then is

$$(\mathbf{k}_{cent}^{new})_j = \langle \phi(\mathbf{x}_j) - \tfrac{1}{n} \sum_i \phi(\mathbf{x}_i), \phi(\mathbf{y}) - \tfrac{1}{n} \sum_i \phi(\mathbf{x}_i) \rangle$$

In order to find a more concise expression for this, we use the linearity of the scalar product to get

$$\begin{aligned} &\langle \phi(\mathbf{x}_j) - \tfrac{1}{n} \sum_i \phi(\mathbf{x}_i), \phi(\mathbf{y}) - \tfrac{1}{n} \sum_i \phi(\mathbf{x}_i) \rangle \\ &= \langle \phi(\mathbf{x}_j), \phi(\mathbf{y}) \rangle - \tfrac{1}{n} \langle \sum_i \phi(\mathbf{x}_i), \phi(\mathbf{y}) \rangle \\ -\tfrac{1}{n} \langle \phi(\mathbf{x}_j), \sum_i \phi(\mathbf{x}_i) \rangle &+ \tfrac{1}{n^2} \langle \sum_i \phi(\mathbf{x}_i), \sum_i \phi(\mathbf{x}_i) \rangle, \end{aligned}$$

and it can easily be see that

$$\begin{aligned} \mathbf{k}_{cent}^{new} &= \mathbf{k}^{new} - \tfrac{1}{n} \mathbb{1}_n \mathbb{1}_n^\top \mathbf{k}^{new} - \tfrac{1}{n} \mathbf{K} \mathbb{1}_n + \tfrac{1}{n^2} \mathbb{1}_n \mathbb{1}_n^\top \mathbf{K} \mathbb{1}_n \\ &= \mathbf{H} \mathbf{k}^{new} - \tfrac{1}{n} \mathbf{H} \mathbf{K} \mathbb{1}_n. \end{aligned}$$

In summary, K-PCA for a training set $\mathbf{X} = [\mathbf{x}_1, \dots \mathbf{x}_n]$, $\mathbf{x}_i \in \mathbb{R}^p$ consists of the following steps.

1. Find a suitable Kernel function $\kappa(\cdot)$ and compute the *Gram matrix*

$$\mathbf{K} = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \kappa(\mathbf{x}_1, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_n) \\ \kappa(\mathbf{x}_2, \mathbf{x}_1) & \kappa(\mathbf{x}_2, \mathbf{x}_2) & & \kappa(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_n, \mathbf{x}_1) & \kappa(\mathbf{x}_n, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}.$$

2. Compute the Gram matrix $\tilde{\mathbf{K}} = \mathbf{H}\mathbf{K}\mathbf{H}$ with $\mathbf{H} = \mathbf{I}_n - \frac{1}{n} \mathbb{1}_n \mathbb{1}_n^\top$ corresponding to centered data.

3. Compute the Eigenvalue Decomposition $\tilde{\mathbf{K}} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^\top$. Due to the definition of kernel functions, the matrix $\mathbf{K}$ is positive semi-definite, and therefore the diagonal entries of $\mathbf{\Lambda}$ are non-negative. Hence, we can write $\mathbf{\Lambda} = \mathbf{\Sigma}^2 = \mathrm{diag}(\sigma_1^2, \dots, \sigma_n^2)$.

4. Define the reduced matrices $\mathbf{\Sigma}_k = \text{diag}(\sigma_1, \ldots, \sigma_k) \in \mathbb{R}^{k \times k}$ and $\mathbf{V}_k \in \mathbb{R}^{n \times k}$.

5. The reduced training data then is given by

$$\mathbf{S} = \mathbf{\Sigma}_k \mathbf{V}_k^\top. \tag{7.11}$$

6. For a new data point $\mathbf{y} \in \mathbb{R}^p$, the k first kernel principal components are then computed by

$$\mathbf{s}_{\text{new}} := \mathbf{\Sigma}_k^{-1} \mathbf{V}_k^\top \mathbf{k}_{cent}^{new} \tag{7.12}$$

with

$$\begin{aligned} \mathbf{k}_{cent}^{new} &= \mathbf{H}(\mathbf{k}^{new} - \tfrac{1}{n}\mathbf{K}\mathbb{1}_n) \\ \text{where } \mathbf{k}^{new} &= [\kappa(\mathbf{x}_1, \mathbf{y}), \ldots, \kappa(\mathbf{x}_n, \mathbf{y})]^\top. \end{aligned}$$

8 Support Vector Machines

The idea behind support vector machines is pretty easy. In the simplest case it is assumed that samples of two classes can be *linearly separated*, i.e. one assumes that there exists an affine hyperplane of codimension one which can separate the two classes. SVMs are supervised learning algorithms which, given training samples of the two classes, find the "best" separating hyperplane. Once it is found it is then easy to classify a new data point: Depending on which side of the hyperplane the incoming new data point lies, it is assigned to the respective class.

8.1 Some Geometry

For some $\mathbf{w} \in \mathbb{R}^p \setminus \{0\}$ and some non-negative $b \geq 0$, an *affine hyperplane* in $\mathbb{R}^p$ is defined as the set

$$\mathcal{H}_{\mathbf{w},b} := \{\mathbf{x} \in \mathbb{R}^p \mid \mathbf{w}^\top \mathbf{x} - b = 0\}. \tag{8.1}$$

The vector $\mathbf{w}$ is *normal*, or perpendicular to $\mathcal{H}_{\mathbf{w},b}$, because whenever we have an arbitrary line segment $\mathbf{x_2} - \mathbf{x_1}$ in the hyperplane with $\mathbf{x_1}, \mathbf{x_2} \in \mathcal{H}_{\mathbf{w},b}$, then

$$\mathbf{w}^\top(\mathbf{x_2} - \mathbf{x_1}) = 0. \tag{8.2}$$

From this it also follows easily that two affine hyperplanes $\mathcal{H}_{\mathbf{w},b}$, $\mathcal{H}_{\tilde{\mathbf{w}},\tilde{b}}$ are parallel if and only if $\mathbf{w}$ is a multiple of $\tilde{\mathbf{w}}$.

Any hyperplane separates $\mathbb{R}^p$ in exactly two half-spaces[1]. The *signed distance* from some point $\mathbf{x}$ to $\mathcal{H}_{\mathbf{w},b}$ is defined as

$$\delta(\mathbf{x}, \mathcal{H}_{\mathbf{w},b}) = \frac{\mathbf{w}^\top \mathbf{x} - b}{\|\mathbf{w}\|}. \tag{8.3}$$

The justification of this definition is the following Lemma.

Lemma 8.1 (Signed distance to affine hyperplanes)**.** *The Euclidean distance from* $\mathbf{x}$ *to* $\mathcal{H}_{\mathbf{w},b}$ *is given by* $|\delta(\mathbf{x}, \mathcal{H}_{\mathbf{w},b})|$.

Proof. We use some geometric intuition for the proof. Assume that starting from $\mathbf{x}$, we want to move towards the hyperplane in direction $\mathbf{r}$. In order to find the shortest distance, we have to solve

$$\min \|\mathbf{r}\| \quad \text{s.t.} \quad \mathbf{x} + \mathbf{r} \in \mathcal{H}_{\mathbf{w},b}. \tag{8.4}$$

[1] This follows from the so-called *hyperplane separation theorem* which essentially states that two disjoint convex sets can be separated by a hyperplane. The proof goes back to famous Hermann Minkowski (1864-1909).

The constraint here is equivalent to $(\mathbf{x}+\mathbf{r})^\top \mathbf{w} = b$, or

$$\mathbf{r}^\top \mathbf{w} = b - \mathbf{x}^\top \mathbf{w}. \tag{8.5}$$

From this equation it follows that $\mathbf{r}$ that solves (8.4) has to be a multiple of $\mathbf{w}$. We proof this in the following. Assume that $\mathbf{r}$ is not a multiple of $\mathbf{w}$. Then we can always write it as $\mathbf{r} = \lambda\mathbf{w} + \sum_i \mu_i \mathbf{w}_i^\perp$ for some $\mathbf{w}_i^\perp$ are orthogonal to $\mathbf{w}$. Due to the orthogonality, the equality $(\lambda\mathbf{w} + \sum_i \mu_i \mathbf{w}_i^\perp)^\top \mathbf{w} = \lambda \mathbf{w}^\top \mathbf{w}$ holds, while at the same time $\|\mathbf{r}\| \geq \|\lambda\mathbf{w}\|$ according to the triangle inequality.

This observation yields the minimization problem

$$\min_\lambda \|\lambda\mathbf{w}\| \quad \text{s.t.} \quad \lambda\|\mathbf{w}\|^2 = b - \mathbf{x}^\top \mathbf{w}, \tag{8.6}$$

which has as solution the absolute value of the signed distance

$$\frac{|\mathbf{w}^\top \mathbf{x} - b|}{\|\mathbf{w}\|} = |\delta(\mathbf{x}, \mathcal{H}_{\mathbf{w},b})|. \tag{8.7}$$

□

It is readily seen that the signed distance to the hyperplane is positive in one half space, and negative in the other. We define the *margin* of $\mathcal{H}_{\mathbf{w},b}$ as the set of points that are close to $\mathcal{H}_{\mathbf{w},b}$. More precisely, let

$$\mathcal{H}_+ := \{\mathbf{x} \in \mathbb{R}^p \mid \mathbf{w}^\top \mathbf{x} - b = 1\} \tag{8.8}$$

$$\mathcal{H}_- := \{\mathbf{x} \in \mathbb{R}^p \mid \mathbf{w}^\top \mathbf{x} - b = -1\} \tag{8.9}$$

be two affine hyperplanes parallel to $\mathcal{H}_{\mathbf{w},b}$. Then the margin of $\mathcal{H}_{\mathbf{w},b}$ is defined as the convex hull of $\mathcal{H}_+ \cup \mathcal{H}_-$.

Using Lemma 8.1, we see that the thickness of this margin, i.e. the distance between $\mathcal{H}_+$ and $\mathcal{H}_-$, is given by $\frac{2}{\|\mathbf{w}\|}$. So if we want to find an affine hyperplane $\mathcal{H}_{\mathbf{w},b}$ that allows for "best" separating two classes of data points, and if we quantify "best" by allowing the largest margin while still preventing data points to fall within this margin, we will have to maximize $\frac{2}{\|\mathbf{w}\|}$ under some constraints, which leads us to linear SVM.

8.2 Basic Linear SVM

As mentioned above, SVM is a supervised learning method, so we start with N labeled training data $(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \{-1, 1\}$, $i = 1, \ldots, N$. Here, y_i is either 1 or -1 and indicates to which of the two classes the data belongs. For linear SVM, we have to assume that this data is linearly separable indeed, i.e. we have to assume that an affine hyperplane $\mathcal{H}_{\mathbf{w},b}$ exists that separates the two classes. The constraint that no data point lies within

the margin of $\mathcal{H}_{\mathbf{w},b}$ is equivalent to requiring that all points belonging to class $y_i = 1$ have a positive distance to $\mathcal{H}_+$, and all data belonging to class $y_i = -1$ have a negative distance to $\mathcal{H}_-$, i.e.

$$\mathbf{w}^\top \mathbf{x}_i - b \geq +1 \qquad \text{for } y_i = +1 \tag{8.10}$$

$$\mathbf{w}^\top \mathbf{x}_i - b \leq -1 \qquad \text{for } y_i = -1. \tag{8.11}$$

This can be written more compactly as

$$y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \text{ for all } i = 1, \ldots, N. \tag{8.12}$$

Therefore, the task of finding the affine hyperplane that allows for the largest margin, while still separating the two classes, is described by the optimization problem

$$\max \frac{2}{\|\mathbf{w}\|^2} \quad \text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \text{ for all } i = 1, \ldots, N, \tag{8.13}$$

or, equivalently, to

$$\min \tfrac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \text{ for all } i = 1, \ldots, N. \tag{8.14}$$

In order to tackle this constrained optimization problem we need to introduce optimality conditions for these types of problems.

8.2.1 Karush-Kuhn-Tucker Conditions

We refer to the text book *Numerical Optimization, 2nd Edition*, Springer 2006, by J. Nocedal and S.J. Wright, for a more detailed insight into the topic of optimization. Consider the general problem

$$\begin{aligned} \min_{\mathbf{z} \in \mathbb{R}^n} \quad & f(\mathbf{z}) \\ \text{s.t.} \quad & c_i(\mathbf{z}) = 0 \text{ for } i \in \mathcal{E}, \\ & \text{and } c_j(\mathbf{z}) \geq 0 \text{ for } j \in \mathcal{I} \end{aligned} \tag{8.15}$$

with the smooth real valued functions f, c_i. Here $\mathcal{E}$ stands for *equality constraints* and $\mathcal{I}$ stands for *inequality constraints*. For some point $\mathbf{z}$ that satisfies the constraints, we define its *active set* as $\mathcal{A}(\mathbf{z}) = \mathcal{E} \cup \{j \,:\, c_j(\mathbf{z}) = 0\}$. In other words, $\mathcal{A}(\mathbf{z})$ are all indices of the constraints where $\mathbf{z}$ *exactly* satisfies the equality conditions.

The Lagrange function of the optimization problem (8.15) is given by

$$L(\mathbf{z}, \boldsymbol{\lambda}) = f(\mathbf{z}) - \sum_{i \in \mathcal{I} \cup \mathcal{E}} \lambda_i c_i(\mathbf{z}). \tag{8.16}$$

Theorem 8.2 (Karush-Kuhn-Tucker (KKT) conditions)**.** *Let* $\mathbf{z}^\star$ *be a solution of* (8.15)*. Under certain conditions on the constraint functions*[2] *there exists a Lagrange multiplier* $\boldsymbol{\lambda}^\star$ *such that*

$$\nabla_{\mathbf{z}} L(\mathbf{z}^\star, \boldsymbol{\lambda}^\star) = 0 \tag{8.17}$$

$$c_i(\mathbf{z}^\star) = 0 \text{ for } i \in \mathcal{E}, \tag{8.18}$$

$$c_i(\mathbf{z}^\star) \geq 0 \text{ for } i \in \mathcal{I} \tag{8.19}$$

$$\lambda_i^\star \geq 0 \text{ for } i \in \mathcal{I} \tag{8.20}$$

$$\lambda_i^\star c_i(\mathbf{z}^\star) = 0 \text{ for } i \in \mathcal{I} \cup \mathcal{E}. \tag{8.21}$$

Since the optimization problem for SVMs is convex (a convex objective function with constraints that define a convex feasible region) the KKT conditions are necessary and sufficient for $\mathbf{z}^\star, \boldsymbol{\lambda}^\star$ to be a solution. The last conditions imply that either constraint i is in the active set, or that the i-th component of the Lagrange multiplier is zero, or possibly both.

8.2.2 Lagrangian Duality

Like before, we will consider the general optimization problem

$$\min f(\mathbf{z}) \quad \text{s.t. } c_i(\mathbf{z}) = 0, i \in \mathcal{E}, \quad c_j(\mathbf{z}) \geq 0, j \in \mathcal{I}.$$

This is also called the *primal* problem. The corresponding Lagrange function is defined as

$$L(\mathbf{z}, \boldsymbol{\lambda}) = f(\mathbf{z}) - \sum_{i \in \mathcal{I} \cup \mathcal{E}} \lambda_i c_i(\mathbf{z})$$

with Lagrangian multipliers (also called *dual variables*) $\lambda_i \geq 0$. Based on the Lagrange function we can create a new function that provides a lower bound on the objective function f. Since the dual variables are all positive, i.e. $\lambda_i \geq 0$, we know that $f(\mathbf{z}) \geq L(\mathbf{z}, \boldsymbol{\lambda})$ for all feasible $\mathbf{z}$. This motivates the definition of the *Lagrange dual function* as

$$g(\boldsymbol{\lambda}) = \inf_{\mathbf{z}} L(\mathbf{z}, \boldsymbol{\lambda}) = \inf_{\mathbf{z}} \left(f(\mathbf{z}) - \sum_{i \in \mathcal{I} \cup \mathcal{E}} \lambda_i c_i(\mathbf{z}) \right). \tag{8.22}$$

The dual function $g(\cdot)$ is concave, even if the original problem is not convex since it is the point-wise infimum of affine functions. The dual form yields lower bounds on the optimal value $p^\star$ of the objective function f. For any $\boldsymbol{\lambda} \geq 0$ we have $g(\boldsymbol{\lambda}) \leq p^\star$.

The Lagrangian dual problem is the maximization problem

$$\max_{\boldsymbol{\lambda}} g(\boldsymbol{\lambda}) \quad \text{s.t.} \quad \lambda_i \geq 0.$$

[2] which are fulfilled in case they are linear, so in particular for the case of linear SVM

Under certain conditions (that hold in the case of SVMs), the minimum of the primal problem coincides with the maximum of the dual problem, i.e. $d^\star = \max g(\boldsymbol{\lambda}) = \inf_{\mathbf{z}} f(\mathbf{z}) = p^\star$. This is called *strong duality*.

Remark. Duality allows us to compute a *lower bound* on the optimal value for any problem, convex or not, using convex optimization. However, the dual function g may not be easy to compute since it is defined as an optimization problem itself. Using duality works best when g can be written in a closed form. Even then it might not be easy to find a solution to the dual problem since not all convex problems are easy to solve.

8.2.3 Linear SVM: Primal and Dual Problem

Following this motivation, for the problem at hand we have to solve the problem

$$\min_{\mathbf{w},b,\boldsymbol{\lambda}\geq 0} \quad L(\mathbf{w},b,\boldsymbol{\lambda}) \tag{8.23}$$

$$\text{with} \quad L(\mathbf{w},b,\boldsymbol{\lambda}) = \tfrac{1}{2}\|\mathbf{w}\|^2 - \sum_i \lambda_i (y_i(\mathbf{w}^\top \mathbf{x}_i - b) - 1) \tag{8.24}$$

$$= \tfrac{1}{2}\|\mathbf{w}\|^2 - \sum_i \lambda_i y_i (\mathbf{w}^\top \mathbf{x}_i - b) + \sum_i \lambda_i. \tag{8.25}$$

This problem is strictly convex, and therefore its solution is unique if it exists. The gradient of the Lagrange function w.r.t. the optimization parameters $(\mathbf{w}, b)$ is given by

$$\nabla_{(\mathbf{w},b)} L(\mathbf{w},b,\lambda) = \begin{bmatrix} \mathbf{w} - \sum_i \lambda_i y_i \mathbf{x}_i \\ \sum_i \lambda_i y_i \end{bmatrix}. \tag{8.26}$$

Therefore, the KKT conditions (8.17) and (8.21) yield

$$\mathbf{w}^\star - \sum_i \lambda_i^\star y_i \mathbf{x}_i = 0 \tag{8.27}$$

$$\sum_i \lambda_i^\star y_i = 0 \tag{8.28}$$

$$\lambda_i^\star (y_i((\mathbf{w}^\star)^\top \mathbf{x}_i - b^\star) - 1) = 0. \tag{8.29}$$

This implies that the solution $\mathbf{w}^\star$ is a linear combination of points that touch the boundary hyperplanes, i.e. those points that lie in $\mathcal{H}_+$ and $\mathcal{H}_-$. These points are the name giving *support points* or *vectors*.

We can now use these equations to derive an easy way for finding the desired optimal hyperplane parameters $\mathbf{w}^\star$ and $b^\star$. Substituting first (8.27) and then and (8.28) into

(8.25) yields (omitting * for improved readability) the equation

$$\begin{aligned}
&\tfrac{1}{2}\|\mathbf{w}\|^2 - \sum_i \lambda_i y_i(\mathbf{w}^\top \mathbf{x}_i - b) + \sum_i \lambda_i \\
=&\tfrac{1}{2}\sum_{i,j} \lambda_i\lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j - \sum_{i,j} \lambda_i\lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + \sum_i \lambda_i y_i b + \sum_i \lambda_i \\
=&\sum_i \lambda_i - \tfrac{1}{2}\sum_{i,j} \lambda_i\lambda_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j.
\end{aligned}$$

This new formulation which only depends on $\boldsymbol{\lambda}$ is the dual form of the problem (cf. (8.22)) and we write it as

$$L_D(\boldsymbol{\lambda}) = \sum_i \lambda_i - \tfrac{1}{2}\boldsymbol{\lambda}^\top \mathbf{H}\boldsymbol{\lambda} \quad \text{s.t.} \quad \lambda_i \geq 0,\ \sum_i \lambda_i y_i = 0,$$

where the entries of $\mathbf{H}$ are defined as the inner products $h_{ij} = y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$. The optimal Lagrangian multipliers are found by solving the maximization problem

$$\max_{\boldsymbol{\lambda}} \left(\sum_i \lambda_i - \tfrac{1}{2}\boldsymbol{\lambda}^\top \mathbf{H}\boldsymbol{\lambda} \right) \quad \text{s.t.} \quad \lambda_i \geq 0,\ \sum_i \lambda_i y_i = 0. \tag{8.30}$$

This is a convex quadratic optimization problem which can be solved by using a quadratic program (QP) solver (e.g. the function `quadprog` in Matlab). The resulting $\boldsymbol{\lambda}^\star$ then yields $\mathbf{w}^\star$ by plugging it into Equation (8.27) and $b^\star$ is obtained via Equation (8.29). Note that the Lagrange multiplier λ_i corresponds to the point $\mathbf{x}_i$. If λ_i is unequal to zero, then Equation (8.29) implies that $y_i(\mathbf{w}^\top \mathbf{x}_i - b) = 1$, i.e. the corresponding $\mathbf{x}_i$ is an element of either $\mathcal{H}_+$ or $\mathcal{H}_-$.

8.3 Soft Margin Linear SVM

Clearly, the above method fails if the training set is not linearly separable, because in this case, there is no point that fulfills the constraints. To overcome this obvious drawback, the *soft margin SVM* allows for some wrongly assigned data samples. It searches for a trade-off between a large margin and a degree of misclassification. This misclassification is quantified by a set of N additional variables ξ_i that are assumed to be nonnegative, leading to the constraints

$$y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 - \xi_i \text{ for all } i = 1, \ldots, N. \tag{8.31}$$

So, in order to gain a large margin while at the same time keeping the misclassifiaction moderate, we consider the optimization problem

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \quad \tfrac{1}{2}\|\mathbf{w}\|_2^2 + c\sum_{i=1}^N \xi_i \tag{8.32}$$

$$\text{s.t.} \quad y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \quad \forall i. \tag{8.33}$$

The free parameter $c > 0$ weighs between a large margin and misclassification. The larger c is chosen, the more the violation of the separation rule is punished. As before, this is a quadratic programming problem that can be solved by using a QP solver. The corresponding KKT conditions are discussed in the following. The Lagrangian for our soft margin SVM reads as

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \tfrac{1}{2}\|\mathbf{w}\|^2 + c\sum_i \xi_i - \sum_i \lambda_i(y_i(\mathbf{w}^\top \mathbf{x}_i - b) - 1 + \xi_i) - \sum_i \mu_i \xi_i, \quad (8.34)$$

where μ_i are the Lagrange multipliers introduced to enforce positivity of ξ_i. The corresponding KKT conditions are

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_i \lambda_i y_i \mathbf{x}_i = 0 \quad (8.35)$$

$$\nabla_b L = \sum_i \lambda_i y_i = 0 \quad (8.36)$$

$$\nabla_{\xi_i} L = c - \lambda_i - \mu_i = 0 \quad (8.37)$$

$$y_i(\mathbf{w}^\top \mathbf{x}_i - b) - 1 + \xi_i \geq 0 \quad (8.38)$$

$$\xi_i \geq 0 \quad (8.39)$$

$$\lambda_i \geq 0 \quad (8.40)$$

$$\mu_i \geq 0 \quad (8.41)$$

$$\lambda_i(y_i(\mathbf{w}^\top \mathbf{x}_i - b) - 1 + \xi_i) = 0 \quad (8.42)$$

$$\mu_i \xi_i = 0 \quad (8.43)$$

While a QP solver could already find the solution to this problem, we will now derive the corresponding dual form since it has an appearance very similar to the separable problem. Also, the dual form is required in order to extend SVMs to work with kernels. First, note that Equation (8.37) implies that $\mu_i = c - \lambda_i$. By plugging this into Equation (8.34) we can already remove the dependency of μ_i. It can also easily be seen that by this substitution ξ_i is eliminated and we obtain

$$\tfrac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_i \lambda_i(y_i(\mathbf{w}^\top \mathbf{x}_i - b) - 1). \quad (8.44)$$

Next, we use the fact from (8.35) that $\mathbf{w} = \sum_i \lambda_i y_i \mathbf{x}_i$ and substitute this into (8.44). By employing the property (8.36), we obtain the dual form

$$L_D(\boldsymbol{\lambda}) = \sum_i \lambda_i - \tfrac{1}{2}\boldsymbol{\lambda}^\top \mathbf{H} \boldsymbol{\lambda} \quad (8.45)$$

with the matrix $\mathbf{H}$ with entries $h_{ij} = y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$ under the constraints $0 \leq \lambda_i \leq c$ and $\sum_i \lambda_i y_i = 0$. Hence, the dual problem has the form

$$\max_{\boldsymbol{\lambda}} L_D(\boldsymbol{\lambda}) \quad \text{s.t.} \quad 0 \leq \lambda_i \leq c, \ \sum_i \lambda_i y_i = 0. \quad (8.46)$$

The solution is then given by $\mathbf{w}^\star = \sum_i \lambda_i y_i \mathbf{x}_i$. Thus, the only difference to the separable case is that the λ_i have an upper bound c. Equation (8.37) and (8.43) imply that $\xi_i = 0$ if $\lambda_i < c$. Thus, any training sample $\mathbf{x}_i$ for which $0 < \lambda_i < c$ is a support vector. Furthermore, for any support vector $\mathbf{x}_i$ Equation (8.42) reduces to $y_i(\mathbf{w}^\top \mathbf{x}_i - b) + 1 = 0$ and can be used to compute $b^\star$. To obtain a more stable solution it is recommended to use the average over all points in the support. Specifically, it is defined as

$$b^\star = \frac{1}{N_{Supp}} \sum_{i \in Supp} \left((\mathbf{w}^\star)^\top \mathbf{x}_i - y_i \right), \tag{8.47}$$

where $Supp$ denotes the support indices and N_{Supp} the number of support vectors.

8.4 Kernel SVM

As we have seen before the linear soft margin SVM problem for a set of trainings samples $(\mathbf{x}_i, y_i)$, $i = 1, \dots, N$, $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in \{-1, 1\}$ can be rewritten in the dual form

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \quad & \sum_i \lambda_i - \tfrac{1}{2}\boldsymbol{\lambda}^\top \mathbf{H}\boldsymbol{\lambda} \\ \text{s.t.} \quad & 0 \leq \lambda_i \leq c, \ \sum_i \lambda_i y_i = 0, \end{aligned} \tag{8.48}$$

with the Gram matrix $\mathbf{H}$ the entries of which are defined as $h_{ij} = y_i y_j \mathbf{x}_i^\top \mathbf{x}_j$. We have talked about the Kernel trick, i.e. using a nonlinear function ϕ that maps the training samples to a high dimensional Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and expressing inner product directly with a kernel function κ, in the chapter about kernel PCA. That is, the kernel function κ maps two points to $\mathbb{R}$ via $\kappa(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathcal{H}}$. The same holds true in this case. The inner product $\mathbf{x}^\top \mathbf{y}$ can be replaced by an appropriate kernel function. Common choices are:

$$\kappa(\mathbf{x}, \mathbf{y}) = (\alpha \mathbf{x}^\top \mathbf{y} + \beta)^\gamma \qquad \text{polynomial kernel} \tag{8.49}$$

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp(-\|\mathbf{x} - \mathbf{y}\|^2 / (2\sigma^2)) \qquad \text{radial basis function kernel} \tag{8.50}$$

$$\kappa(\mathbf{x}, \mathbf{y}) = \tanh(\gamma \mathbf{x}^\top \mathbf{y} - \delta) \qquad \text{sigmoid kernel} \tag{8.51}$$

With $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$. The entries of the Gram matrix are then defined using one of these kernel functions via

$$h_{ij} = y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j). \tag{8.52}$$

Some notes regarding the previously mentioned kernels:

- In the case of the polynomial kernel the dimension of the Hilbert space $\mathcal{H}$ is $\binom{p+d}{d}$ when the original signals are in $\mathbb{R}^p$.
- The radial basis function kernel is often also referred to as Gauss kernel and describes a nonlinear function ϕ that maps to an infinite dimensional Hilbert space $\mathcal{H}$.
- The sigmoid kernel only produces a s.p.d. Kernel matrix for certain choices of γ, δ and under specific conditions on the squared norm of the signals $\|\mathbf{x}\|^2$.

Solving the problem

$$\begin{aligned} \max_{\boldsymbol{\lambda}} \quad & \sum_i \lambda_i - \tfrac{1}{2}\boldsymbol{\lambda}^\top \mathbf{H}\boldsymbol{\lambda} \\ \text{s.t.} \quad & 0 \leq \lambda_i \leq c, \ \sum_i \lambda_i y_i = 0. \end{aligned} \tag{8.53}$$

with $h_{ij} = y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$ provides the Lagrange coefficients $\boldsymbol{\lambda}^\star$, but how can we classify a new point using this results? First, note that according to the classification rule from the linear case (i.e. a vector $\mathbf{z}$ is classified using the rule $\text{sign}(\mathbf{w}^\top \mathbf{z} - b)$), we can express the decision function by

$$f(\mathbf{z}) = \sum_{i \in Supp} \lambda_i y_i \kappa(\mathbf{x}_i, \mathbf{z}) - b, \tag{8.54}$$

where $Supp$ denotes the support (i.e., all i for which $0 < \lambda_i \leq c$). We assign the label according to the sign of $f(\mathbf{z})$.

The only remaining ingredient we need is the factor b. From the KKT conditions of the original problem, we know that the equation

$$y_i(\langle \phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}}) - b) - 1 = 0$$

has to hold for any $i \in Supp$. The vector $\mathbf{w}$ is an element of the high dimensional Hilbert space $\mathcal{H}$ and can be rewritten as a sum of the mapped support vectors $\sum_{j \in Supp} \lambda_i y_i \phi(\mathbf{x}_i)$. Plugging this into the previous equation yields

$$b = \sum_{j \in Supp} \left(\lambda_j y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}} \right) - y_i$$

for any i in the support. To make this result more robust, we average over all possible indices i and get

$$b = \tfrac{1}{N_{Supp}} \sum_{i \in Supp} \left(\sum_{j \in Supp} \left(\lambda_j y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}} \right) - y_i \right),$$

where N_{Supp} is the number of support vectors. Hence, by replacing the inner products with the kernel function, we can compute b as

$$b = \tfrac{1}{N_{Supp}} \sum_{i \in Supp} \left(\sum_{j \in Supp} \left(\lambda_j y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \right) - y_i \right). \tag{8.55}$$

Bibliography