1 Linear algebra

Note that we consider only real values.

1.1 Gram-Schmidt orthogonalization

Using orthogonal projections $\frac{<v,u>}{<u,u>}u$ of vectors v onto vectors u and differences $v - \frac{<v,u>}{<u,u>}u$, the Gram-Schmidt orthogonalization constructs from a set of vectors a set of orthogonal ones in an iterative procedure. Note that the difference $v - \frac{<v,u>}{<u,u>}u$ is orthogonal to the projection (assuming u is a unit vector):

$$\begin{aligned} \langle < v,u > u, v - < v,u > u \rangle &= \langle < v,u > u, v \rangle - \langle < v,u > u, < v,u > u \rangle && (1.1)\\ &= < v,u >^2 - < v,u >^2 \underbrace{< u,u >}_{=1} && (1.2)\\ &= 0. && (1.3) \end{aligned}$$

Let $S_V = \{v_1, v_2, ..., v_n\}$ be a set of vectors and d be the dimension of the space V spanned by the vectors in S_V. Then the Gram-Schmidt orthogonalization (or here orthonormalization) generates an orthonormal basis set $S_U = \{u_1, u_2, ..., u_d\}$ in V by:

1. $u_1 = \frac{v_1}{\|v_1\|}$

2. For all $i > 1$: first
$$\tilde{u}_i = v_i - \sum_{k=1}^{i-1} < v_i, u_k > u_k, \qquad (1.4)$$
then $u_i = \frac{\tilde{u}_i}{\|\tilde{u}_i\|}$.

Zero vectors, which occur when there is a linear dependency in S_V, are not added to S_U. The sum in Eq. 1.4 gives the orthogonal projection of the vector v_i onto the hyperplane spanned by the $i-1$ vectors u_k. Note that in the code, we implement the orthogonalization also for functions (not only arrays), i.e. in "ortho_basis.py". If arrays (matrices) are used, the second step can be calculated via (in case of the standard scalar product)

$$\tilde{\boldsymbol{u}}_i = \boldsymbol{v}_i - \boldsymbol{U}_{i-1}\boldsymbol{U}_{i-1}^T\boldsymbol{v}_i, \qquad (1.5)$$

where $\boldsymbol{U}_{i-1}$ denotes the matrix with columns $\boldsymbol{u}_1, ..., \boldsymbol{u}_{i-1}$.
Better numerical stability can be obtained by the **modified Gram-Schmidt** orthogonalization which replaces the second step of the not-modified one by:

1. Let $k = 1$ and $\tilde{u}_i^{(1)} = v_i$.

2. Calculate, first
$$\tilde{u}_i^{(k+1)} = \tilde{u}_i^{(k)} - < \tilde{u}_i^{(k)}, u_k > u_k, \tag{1.6}$$
then $u_i^{(k+1)} = \frac{\tilde{u}_i^{(k+1)}}{\|\tilde{u}_i^{(k+1)}\|}$.

3. If $k = i - 1$ stop and $u_i = u_i^{(k+1)}$. Otherwise, let $k = k + 1$ and continue with 2. step (of this algorithm).

1.2 QR decomposition

The matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is decomposed into an orthogonal matrix $\boldsymbol{Q}$ and an upper triangular matrix $\boldsymbol{R}$:
$$\boldsymbol{A} = \boldsymbol{QR}. \tag{1.7}$$

In our implementation both $\boldsymbol{Q}$ and $\boldsymbol{R}$ do not need to be square matrices.

1.2.1 Using Gram-Schmidt orthogonalization

Using the Gram-Schmidt process an orthogonal basis set is found (for the column space of $\boldsymbol{A}$) by transforming the columns of $\boldsymbol{A}$. $\boldsymbol{Q}$ is given by the new orthogonal vectors. Therefore, we use the size convention $\boldsymbol{Q} \in \mathbb{R}^{m \times \text{rank}(A)}$ and $\boldsymbol{R} \in \mathbb{R}^{\text{rank}(A) \times n}$. One way to obtain $\boldsymbol{R}$ is by calculating $\boldsymbol{R} = \boldsymbol{Q}^T \boldsymbol{A}$. However, in practice the whole matrix multiplication does not need to be carried out because some entries are known to become zero.

1.2.2 Using Housholder reflections

The QR decompositions based on Housholder reflections is known to be more numerically stable than the Gram-Schmidt process (however, I could not verify this statement through the cases I have looked into with the here presented codes.) The Housholder reflection is a linear transformation that reflects a vector x about some (hyper-)plane by

$$x' = x - 2 < x, v > v, \tag{1.8}$$

where v is a unit vector orthogonal to the hyperplane. The reflected vector x' has the same length as x:

$$\begin{aligned}
\|x'\|^2 &= \langle x - 2 < x, v > v, x - 2 < x, v > v\rangle && (1.9)\\
&=< x, x > -2\langle x, 2 < x, v > v\rangle + 4 < x, v >^2 \underbrace{< v, v >}_{=1} && (1.10)\\
&=< x, x > -4 < x, v >^2 +4 < x, v >^2 && (1.11)\\
&= \|x\|^2. && (1.12)
\end{aligned}$$

Now, let us define:

$$\begin{aligned}
u &= x - \|x\|e, && (1.13)\\
v &= \frac{u}{\|u\|}, && (1.14)
\end{aligned}$$

where e a unit vector. Then, with $< e, e >= 1$,

$$\begin{aligned}
x' &= x - 2\frac{< x, x - \|x\|e >}{< x - \|x\|e, x - \|x\|e >}(x - \|x\|e) && (1.15)\\
&= x - 2\frac{< x, x > -\|x\| < x, e >}{< x, x > -2\|x\| < x, e > x + \|x\|^2 < e, e >}(x - \|x\|e) && (1.16)\\
&= x - 2\frac{\|x\|^2 - \|x\| < x, e >}{\|x\|^2 - 2\|x\| < x, e > x + \|x\|^2}(x - \|x\|e) && (1.17)\\
&= x - 2\frac{1}{2}(x - \|x\|e) && (1.18)\\
&= \|x\|e && (1.19)\\
& && (1.20)
\end{aligned}$$

So, interestingly, with the choice in Eq. 1.13 and 1.14 for v the Housholder reflection becomes a projection of x onto the unit vector e. Crucially, if we chose for e the vector $(1, 0, 0, ...)^T$ the Housholder reflection based on this v will transform x into a vector where only one element is non-zero. As a consequence, we can build a clever iterative procedure to apply this reflection (partially) to every column of a matrix $\boldsymbol{A}$ and transform $\boldsymbol{A}$ to the targeted upper triangular form $\boldsymbol{R}$ of the QR decompostion, as described in the following.

Consider matrix form. We can rewrite Eq. 1.8 to

$$\boldsymbol{x}' = \boldsymbol{Q}\boldsymbol{x} \tag{1.21}$$

with the square matrix

$$\boldsymbol{Q} = \boldsymbol{I} - 2\boldsymbol{v}\boldsymbol{v}^T. \tag{1.22}$$

In the first iteration step of the QR decomposition we define

$$\boldsymbol{u}_1 = \boldsymbol{A}_1 - \|\boldsymbol{A}_1\|\boldsymbol{e}_1, \tag{1.23}$$

$$\boldsymbol{v}_1 = \frac{\boldsymbol{u}_1}{\|\boldsymbol{u}_1\|}, \tag{1.24}$$

where $\boldsymbol{A}_1$ is the first column of $\boldsymbol{A}$ and $\boldsymbol{e}_1 = (1, 0, ..., 0)^T$ a unit vector. Then, with $\boldsymbol{Q}_1 = \boldsymbol{I} - 2\boldsymbol{v}_1\boldsymbol{v}_1^T$,

$$\boldsymbol{Q}_1\boldsymbol{A}_1 = \begin{pmatrix} \|\boldsymbol{A}_1\| \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{1.25}$$

and

$$\boldsymbol{Q}_1\boldsymbol{A} = \begin{pmatrix} \|\boldsymbol{A}_1\| & \star & \dots & \star \\ 0 & & & \\ \vdots & & \boldsymbol{A}' & \\ 0 & & & \end{pmatrix}. \tag{1.26}$$

In the next step this is repeated for the matrix $\boldsymbol{A}'$ ($\boldsymbol{A}$ with deleted first row and column), where now $\boldsymbol{v}_2$ is based on the first column $\boldsymbol{A}_1'$ of $\boldsymbol{A}'$, resulting in a matrix $\boldsymbol{Q}_2'$. To keep the original matrix sizes, $\boldsymbol{Q}_k'$ of any iteration step k is extended by:

$$\boldsymbol{Q}_k = \begin{pmatrix} \boldsymbol{I}_{k-1} & 0 \\ 0 & \boldsymbol{Q}_k' \end{pmatrix}. \tag{1.27}$$

After $t = \min(m-1, n)$ iterations, the transpose of the final orthogonal matrix $\boldsymbol{Q}$ and the triangular matrix $\boldsymbol{R}$ of the QR decomposition are given by

$$\boldsymbol{Q}^T = \boldsymbol{Q}_t \cdots \boldsymbol{Q}_2\boldsymbol{Q}_1 \tag{1.28}$$

and

$$\boldsymbol{R} = \boldsymbol{Q}^T\boldsymbol{A}. \tag{1.29}$$

1.3 Eigenvalues and -vectors

The eigenvalue equation is written

$$\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}, \tag{1.30}$$

where $\boldsymbol{A}$ is a square matrix, the non-zero vector $\boldsymbol{v}$ is the eigenvector, and the scalar λ is the eigenvalue. The eigenvectors yield the set of vectors which are not rotated in a linear transformation based on $\boldsymbol{A}$ but only scaled or inverted in direction.

1.3.1 QR algorithm

The (practical) QR algorithm (without shifts) calculates the eigenvalues of $\boldsymbol{A}$ using QR decompositions iteratively. We initialize $\boldsymbol{A}_0 = \boldsymbol{A}$ and calculate at each iteration

$$\boldsymbol{A}_{k+1} = \boldsymbol{R}_k \boldsymbol{Q}_k, \tag{1.31}$$

where $\boldsymbol{R}_k$ and $\boldsymbol{Q}_k$ are obtained from the QR decomposition: $\boldsymbol{A}_k = \boldsymbol{Q}_k \boldsymbol{R}_k$. Given that all $\boldsymbol{A}_k$ are similar,

$$\boldsymbol{A}_{k+1} = \boldsymbol{R}_k \boldsymbol{Q}_k = \boldsymbol{Q}_k^{-1} \boldsymbol{Q}_k \boldsymbol{R}_k \boldsymbol{Q}_k = \boldsymbol{Q}_k^{-1} \boldsymbol{A}_k \boldsymbol{Q}_k, \tag{1.32}$$

they all have the same eigenvalues, as:

$$\boldsymbol{A}_{k+1} \boldsymbol{v} = \boldsymbol{Q}_k^{-1} \boldsymbol{A}_k \boldsymbol{Q}_k \boldsymbol{v} \tag{1.33}$$

$$\lambda \boldsymbol{v} = \boldsymbol{Q}_k^{-1} \boldsymbol{A}_k \boldsymbol{Q}_k \boldsymbol{v} \tag{1.34}$$

$$\lambda \boldsymbol{Q}_k \boldsymbol{v} = \boldsymbol{A}_k \boldsymbol{Q}_k \boldsymbol{v}. \tag{1.35}$$

So every eigenvalue λ of $\boldsymbol{A}_{k+1}$ is also an eigenvalue of $\boldsymbol{A}_k$, where the corresponding eigenvector of $\boldsymbol{A}_k$ is $\boldsymbol{Q}_k \boldsymbol{v}$. And one can similarly show that every eigenvalue of $\boldsymbol{A}_k$ is an eigenvalue of $\boldsymbol{A}_{k+1}$.
Under some conditions, the matrices $\boldsymbol{A}_k$ will converge against a triangular matrix in the iterative procedure. The eigenvalues of a triangular matrix are given by its diagonal elements. If $\boldsymbol{A}$ is symmetric, then the product of the matrices $\cdots \boldsymbol{Q}_2 \boldsymbol{Q}_1 \boldsymbol{Q}_0$ yield a matrix whose columns are the eigenvectors of $\boldsymbol{A}$.

1.4 Singular value decomposition

The matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is decomposed into:

$$\boldsymbol{A} = \boldsymbol{U} \boldsymbol{S} \boldsymbol{V}^T, \tag{1.36}$$

where $\boldsymbol{U} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices and $\boldsymbol{S} \in \mathbb{R}^{m \times n}$ is a diagonal rectangular matrix. The diagonal entries of $\boldsymbol{S}$ are given by the singular values of $\boldsymbol{A}$, i.e. the square roots of non-negative eigenvalues of $\boldsymbol{A}^T \boldsymbol{A}$. However, in this code, $\boldsymbol{U}$ and $\boldsymbol{V}$ are not necessarily square matrices but shaped by rank($\boldsymbol{A}$) while $\boldsymbol{S}$ is square. We compute the decomposition by:

1. Calculate the eigenvalues and -vectors of $\boldsymbol{A}^T \boldsymbol{A}$.

2. Build the matrix $\boldsymbol{V}$ of the eigenvectors as columns and the rectangular diagonal matrix $\boldsymbol{S}$ with square roots of the non-zero eigenvalues.

3. Calculate $\boldsymbol{U} = \boldsymbol{A}\boldsymbol{V}\boldsymbol{S}^{-1}$.

Interestingly, when defining a linear transformation T with $\boldsymbol{A}$ from K^n to K^m, then

$$T(\boldsymbol{V}_i) = \sigma_i \boldsymbol{U}_i \quad \text{for} \quad i = 1, ..., \operatorname{rank}(\boldsymbol{A}) \tag{1.37}$$

$$T(\boldsymbol{V}_i) = 0 \quad \text{for} \quad i > \operatorname{rank}(\boldsymbol{A}). \tag{1.38}$$

This means that one can find orthonormal bases of K^n and K^m such that T maps the i-th basis vector of K^n to a non-negative multiple of the i-th basis vector of K^m, and yields zero for the remaining basis vectors.

1.5 Pseudoinvers

In order to calculate the generalized inverse $\boldsymbol{A}^+ \in \mathbb{R}^{n\times m}$ of a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ that is non-square or not invertible, we use the singular value decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$:

$$\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{U}^T. \tag{1.39}$$

This definition fulfills the Moore-Penrose conditions that define a pseudoinverse:

1. $\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{A}^+$

2. $\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}$

3. $(\boldsymbol{A}\boldsymbol{A}^+)^T = \boldsymbol{A}\boldsymbol{A}^+$

4. $(\boldsymbol{A}^+\boldsymbol{A})^T = \boldsymbol{A}^+\boldsymbol{A}$.

1.6 Cholesky decomposition

The Cholesky decomposition of a positive-definite matrix $A \in \ltimes \times \ltimes$ is given by

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T \tag{1.40}$$

where $\boldsymbol{L} \in \mathbb{R}^{n\times n}$ is a lower triangular matrix with positive diagonal elements. The decomposition can be shown using the QR decomposition. A positive-definitem matrix $\boldsymbol{A}$ can be written as a product of its square root matrix, $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{B}^T$. With $\boldsymbol{B}^T = \boldsymbol{Q}\boldsymbol{R}$ and $\boldsymbol{R} = \boldsymbol{L}^T$, we have:

$$\boldsymbol{A} = \boldsymbol{B}\boldsymbol{B}^T = \boldsymbol{R}^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{R} = \boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{L}\boldsymbol{L}^T. \tag{1.41}$$

The elements of $\boldsymbol{L}$ are determined by:

$$L_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2}, \tag{1.42}$$

$$L_{i,j} = \frac{1}{L_{j,j}} \left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right) \quad \text{for} \quad i > j. \tag{1.43}$$

The Cholesky decomposition is useful for solving $\boldsymbol{Ax} = \boldsymbol{b}$. The equation is rewritten: $\boldsymbol{LL}^T\boldsymbol{x} = \boldsymbol{b}$. Then one can solve first $\boldsymbol{Ly} = \boldsymbol{b}$ using forward substitution and next $\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{y}$ via back substitution.

2 Quadratic programming

Consider the optimization problem:

$$\begin{aligned}
\min_{\boldsymbol{x}} \quad & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{p}^T\boldsymbol{x} && (2.1)\\
\text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && (2.2)\\
& \boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h} && (2.3)\\
& \boldsymbol{x} \geq \boldsymbol{0}. && (2.4)
\end{aligned}$$

This problem is solved by quadratic programming. Before that, we transform the inequality constraint $\boldsymbol{G}\boldsymbol{x} \leq \boldsymbol{h}$ to an equality constraint, by introducing slack variables $\boldsymbol{\xi} > \boldsymbol{0}$ such that $\boldsymbol{G}\boldsymbol{x} + \boldsymbol{\xi} \leq \boldsymbol{h}$. This transforms the constraints to:

$$\begin{pmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{G} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{h} \end{pmatrix}, \tag{2.5}$$

$$\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{pmatrix} \geq \boldsymbol{0}. \tag{2.6}$$

From now we will assume that this is already taken care of (also the right vectorizations in the implementation) and that it is enough to consider a problem of the form:

$$\begin{aligned}
\min_{\boldsymbol{x}} \quad & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{p}^T\boldsymbol{x} && (2.7)\\
\text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} && (2.8)\\
& \boldsymbol{x} \geq \boldsymbol{0}. && (2.9)
\end{aligned}$$

We will solve this problem by the primal-dual interior-point method.
First we will reformulate the problem based on minimizing a function with constraints to minimizing a function with further penalty terms, using the method of Lagrange multipliers and introducing a logarithmic barrier function for the non-negative variables to force almost all iterates to stay in the feasible set (space of $\boldsymbol{x}$ fulfilling the constraints). Define the Lagrange function:

$$\mathcal{L} = \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{p}^T\boldsymbol{x} - \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) - \mu \sum_i \log x_i, \tag{2.10}$$

where $\boldsymbol{\lambda} > \mathbf{0}$ and $\mu > 0$. For a given μ, the minimum point satisfies the Karush-Kuhn-Tucker conditions:

$$\nabla_{\boldsymbol{x}}\mathcal{L} = \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{p} - \boldsymbol{A}\boldsymbol{\lambda}^T - \mu\boldsymbol{\chi} = 0, \tag{2.11}$$

$$\nabla_{\boldsymbol{\lambda}}\mathcal{L} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = 0, \tag{2.12}$$

where $\chi_i = 1/x_i$. By defining $\boldsymbol{s} = \mu\boldsymbol{\chi}$ the equation to be solved becomes:

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{s}) = \mathbf{0} \tag{2.13}$$

with

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{s}) = \begin{pmatrix} \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{p} - \boldsymbol{A}\boldsymbol{\lambda}^T - \boldsymbol{s} \\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \\ \boldsymbol{X}\boldsymbol{s} - \boldsymbol{\mu} \end{pmatrix}, \tag{2.14}$$

where $\boldsymbol{X} = \operatorname{diag}(\boldsymbol{x})$ is a diagonal with elements of $\boldsymbol{x}$ and $\boldsymbol{\mu}$ is a vector full of μ. We will solve this problem using Newtons method, i.e. an iterative procedure to find the root, where at each step, x is updated by Δx, with $f(x)'\Delta x = f(x)$. The Jacobian matrix (gradients of $\boldsymbol{f}$) is denoted by:

$$\boldsymbol{J}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{s}) = \begin{pmatrix} \boldsymbol{Q} & -\boldsymbol{A}^T & -\boldsymbol{I} \\ \boldsymbol{A} & \mathbf{0} & \mathbf{0} \\ \boldsymbol{S} & \mathbf{0} & \boldsymbol{X} \end{pmatrix}, \tag{2.15}$$

where $\boldsymbol{S} = \operatorname{diag}(\boldsymbol{s})$. Then, at each iteration k, we solve the linear system

$$\boldsymbol{J}(\boldsymbol{t}^k)\boldsymbol{d} = \boldsymbol{f}(\boldsymbol{t}^k), \tag{2.16}$$

where $\boldsymbol{t}^k = (\boldsymbol{x}^k, \boldsymbol{\lambda}^k, \boldsymbol{s}^k)$, and then update $\boldsymbol{t}^{k+1} = \boldsymbol{t}^k + \alpha\boldsymbol{t}^k$. Note that this gives a minimum point $\boldsymbol{x}_\mu$ only for a given μ. However, by decreasing μ, x_μ will converge towards the solution of the problem. In practice, μ (and also α) could be updated at each iteration k of the Newton(-like) method.

3 Machine learning

We consider a data set of n input-output pairs $\{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)\}$. y_i represents the value of a targeted property of the data point i to be predicted by a vectorial input (representation of the data point) $\boldsymbol{x}_i \in \mathbb{R}^m$ of the data point. Often, the goal is to identify a function f out of a function space $\mathcal{F}$ in order to describe the input-output relationship. A general formulation of the ML optimization problem is given by

$$\underset{f \in \mathcal{F}}{\arg\min} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i), y_i) + \lambda r(f), \tag{3.1}$$

where L is a loss function and $r(f)$ is a measurement of the *complexity* of f. The parameter $\lambda \geq 0$ regulates the compromise between the accuracy and a low complexity of the model. One reason for regularization is to avoid *overfitting* (fitting *unimportant fluctuations*) in order to increase the prediction accuracy on data points out of the training set. The choice of $\mathcal{F}$, L and k determines the ML method. All supervised-learning based methods used in the code are based on the squared error loss $L(f(\boldsymbol{x}_i), y_i) = (f(\boldsymbol{x}_i) - y_i)^2$.

In the following sections we write many equations in matrix form and yield the n data points as a target vector $\boldsymbol{y} \in \mathbb{R}^n$ and an input matrix $\boldsymbol{X} \in \mathbb{R}^{n \times m}$. We will assume that $\boldsymbol{X}$ is standardized to have zero mean and variance one, for numerical or conceptional reasons.

3.1 Linear regression

The most basic and widely used machine-learning method is linear regression. One reason is that many linear-regression problems can be solved by linear algebra and convex optimization. The least-squares problem

$$\underset{\boldsymbol{c} \in \mathbb{R}^m}{\arg\min} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{c}\|^2 \tag{3.2}$$

formulates the fundament of linear regression analysis determining the regression coefficients $\boldsymbol{c}$ of a linear model $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{c}$. Its solution is given by the closed-form expression $\boldsymbol{c}^* = (\boldsymbol{X}^\mathrm{T}\boldsymbol{X})^{-1}\boldsymbol{X}^\mathrm{T}\boldsymbol{y}$, which gives the orthogonal projection $\mathrm{Proj}_{\mathcal{C}(\boldsymbol{X})}(\boldsymbol{y}) = \boldsymbol{X}\boldsymbol{c}^*$ of $\boldsymbol{y}$ onto the column space $\mathcal{C}(\boldsymbol{X}) = \{\boldsymbol{X}\boldsymbol{c} \in \mathbb{R}^n : \boldsymbol{c} \in \mathbb{R}^m\}$ of $\boldsymbol{X}$.

The fact that the orthogonal projection minimizes the Euclidean distance in Eq. 3.2 is clear because any other vector $\boldsymbol{X}(\boldsymbol{c}^* - \boldsymbol{k})$ with nonzero $\boldsymbol{k}$ will have a higher distance to $\boldsymbol{y}$ due to the orthogonality $< \mathrm{Proj}_{\mathcal{C}(\boldsymbol{X})}(\boldsymbol{y}), \boldsymbol{X}\boldsymbol{k} >= 0$ (using Pytagoras theorem):

$$\begin{aligned} \|\boldsymbol{y} - \mathrm{Proj}_{\mathcal{C}(\boldsymbol{X})}(\boldsymbol{y}) + \boldsymbol{X}\boldsymbol{k}\|^2 &= \|\boldsymbol{y} - \mathrm{Proj}_{\mathcal{C}(\boldsymbol{X})}(\boldsymbol{y})\|^2 + \|\boldsymbol{X}\boldsymbol{k}\|^2 && (3.3)\\ &> \|\boldsymbol{y} - \mathrm{Proj}_{\mathcal{C}(\boldsymbol{X})}(\boldsymbol{y})\|^2. && (3.4) \end{aligned}$$

We can calculate the least squares solution using the singular value decomposition by (note that we use square $\boldsymbol{S}$):

$$\begin{aligned} \boldsymbol{c}^* &= (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.5)\\ &= (\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.6)\\ &= (\boldsymbol{V}\boldsymbol{S}\boldsymbol{S}\boldsymbol{V}^T)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.7)\\ &= (\boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{S}^{-1}\boldsymbol{V}^T)\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.8)\\ &= \boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{S}^{-1}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T\boldsymbol{y} && (3.9)\\ &= \boldsymbol{V}\boldsymbol{S}^{-1}\boldsymbol{U}^T\boldsymbol{y}. && (3.10) \end{aligned}$$

In order to avoid overfitting (and also numerical instability), the problem 3.2 is often extended by an ℓ_2 penalty $\lambda\|c\|_2^2 = \lambda\|c\|^2$ (linear ridge regression):

$$\underset{\boldsymbol{c}\in\mathbb{R}^m}{\arg\min}\, \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{c}\|^2 + \lambda\|\boldsymbol{c}\|^2. \tag{3.11}$$

The solution to this problem is given by $\boldsymbol{c} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$, where $\boldsymbol{I} \in \mathbb{R}^{m\times m}$ is the identity matrix. In terms of singular value decomposition, we can write the solution:

$$\begin{aligned} \boldsymbol{c}^* &= (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.12)\\ &= (\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.13)\\ &= (\boldsymbol{V}\boldsymbol{S}\boldsymbol{S}\boldsymbol{V}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.14)\\ &= (\boldsymbol{V}\boldsymbol{S}\boldsymbol{S}\boldsymbol{V}^T + \lambda\boldsymbol{I}\boldsymbol{V}\boldsymbol{V}^T)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.15)\\ &= (\boldsymbol{V}[\boldsymbol{S}\boldsymbol{S} + \lambda\boldsymbol{I}]\boldsymbol{V}^T)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.16)\\ &= \boldsymbol{V}(\boldsymbol{S}\boldsymbol{S} + \lambda\boldsymbol{I})^{-1}\boldsymbol{V}^T\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} && (3.17)\\ &= \boldsymbol{V}(\boldsymbol{S}\boldsymbol{S} + \lambda\boldsymbol{I})^{-1}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T\boldsymbol{y} && (3.18)\\ &= \boldsymbol{V}(\boldsymbol{S}\boldsymbol{S} + \lambda\boldsymbol{I})^{-1}\boldsymbol{S}\boldsymbol{U}^T\boldsymbol{y}. && (3.19) \end{aligned}$$

Both the least-squares and ridge solution are written in the form of $\boldsymbol{V}\boldsymbol{D}\boldsymbol{U}^T\boldsymbol{y}$ and distinguished only by the diagonal matrix $\boldsymbol{D}$. The elements of $\boldsymbol{D}$ are given by

$$d_{ii} = \frac{1}{s_{ii}} \tag{3.20}$$

and

$$d_{ii} = \frac{s_{ii}}{s_{ii}^2 + \lambda}, \tag{3.21}$$

respectively. When the s_{ii} come close to zero, the inverse singular values explode. If $\lambda > 0$, the solution of the ridge regression becomes not only more stable, but also the coefficients of correlated columns (features) of $\boldsymbol{X}$ become more strongly shrinkaged. In particular, $|s_{ii}|$ is small when features are correlated, because when being close to collinearity, $\|\boldsymbol{X}\boldsymbol{v}_i\|$ is small:

$$\begin{align}
\|\boldsymbol{X}\boldsymbol{v}_i\|^2 &= \boldsymbol{v}_i^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{v}_i \tag{3.22}\\
&= \boldsymbol{v}_i^T\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^T\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T\boldsymbol{v}_i \tag{3.23}\\
&= \boldsymbol{v}_i^T\boldsymbol{V}\boldsymbol{S}\boldsymbol{S}\boldsymbol{V}^T\boldsymbol{v}_i \tag{3.24}\\
&= s_{ii}^2. \tag{3.25}
\end{align}$$

3.2 Non-negative linear regression

Assume we want solve the linear regression (ridge regression) problem of Eq. 3.11, however, with the constraint that the coefficients $\boldsymbol{c}$ should be non-negative:

$$\operatorname*{arg\,min}_{\boldsymbol{c}\in\mathbb{R}^m} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{c}\|^2 + \lambda\|\boldsymbol{c}\|^2 \quad \text{subject to} \quad \boldsymbol{c} \geq \boldsymbol{0}. \tag{3.26}$$

Rewriting the cost function in Eq. 3.32 yields:

$$\begin{align}
\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{c}\|^2 + \lambda\|\boldsymbol{c}\|^2 &= (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{c})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{c}) + \lambda\boldsymbol{c}^T\boldsymbol{c} \tag{3.27}\\
&= \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T\boldsymbol{X}\boldsymbol{c} - (\boldsymbol{X}\boldsymbol{c})^T\boldsymbol{y} + (\boldsymbol{X}\boldsymbol{c})^T\boldsymbol{X}\boldsymbol{c} + \lambda\boldsymbol{c}^T\boldsymbol{c} \tag{3.28}\\
&= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T\boldsymbol{X}\boldsymbol{c} + \boldsymbol{c}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{c} + \lambda\boldsymbol{c}^T\boldsymbol{c} \tag{3.29}\\
&= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T\boldsymbol{X}\boldsymbol{c} + \boldsymbol{c}^T(\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})\boldsymbol{c} \tag{3.30}\\
&= \boldsymbol{y}^T\boldsymbol{y} + 2\boldsymbol{p}^T\boldsymbol{c} + \boldsymbol{c}^T\boldsymbol{Q}\boldsymbol{c}, \tag{3.31}
\end{align}$$

where $\boldsymbol{Q} = \boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}$ and $\boldsymbol{p} = -\boldsymbol{X}^T\boldsymbol{y}$. Substituting, Eq. 3.31 into Eq. 3.32 gives (in equivalent form):

$$\operatorname*{arg\,min}_{\boldsymbol{c}\in\mathbb{R}^m} \frac{1}{2}\boldsymbol{c}^T\boldsymbol{Q}\boldsymbol{c} + \boldsymbol{p}^T\boldsymbol{c} \quad \text{subject to} \quad \boldsymbol{c} \geq \boldsymbol{0}. \tag{3.32}$$

This problem can be solved by quadratic programming described in Chapter 2.

3.3 Orthogonal matching pursuit

In some applications it is desirable to find linear models $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{c}$ where some, many, or most coefficients (elements of $\boldsymbol{c}$) are zero. Apart from the least absolute shrinkage and selection operator (LASSO, ℓ_1 regularization) the orthogonal-matching-pursuit (OMP) algorithm provides a way to find such a model. It is a greedy algorithm which adds at each iteration k a column of $\boldsymbol{X}$ with non-zero coefficient to the model (where all columns with zero c_i are removed in some sense) that is based on all added columns during the iterative procedure. The non-zero coefficients are determined via least-squares regression. The added column $\boldsymbol{x}^r$ provides that column among all columns $\boldsymbol{x}^i$ of $\boldsymbol{X}$ which is the closest to the current residual vector $\boldsymbol{R} = \boldsymbol{y} - f_k$

$$\|\boldsymbol{R} - \boldsymbol{x}^r c_r\|^2 \leq \|\boldsymbol{R} - \boldsymbol{x}^i c_i\|^2 \quad \forall i \tag{3.33}$$

where c_i denotes the corresponding least squares solutions in fitting $\boldsymbol{R}$. First, let us add $\|\boldsymbol{x}^r c_r\|_2^2 + \|\boldsymbol{x}^i c_i\|_2^2$ to both sides of Eq. 3.33:

$$\|\boldsymbol{R} - \boldsymbol{x}^r c_r\|^2 + \|\boldsymbol{x}^r c_r\|^2 + \|\boldsymbol{x}^i c_i\|^2 \leq \|\boldsymbol{R} - \boldsymbol{x}^i c_i\|^2 + \|\boldsymbol{x}^r c_r\|^2 + \|\boldsymbol{x}^i c_i\|^2 \tag{3.34}$$

Recall that the least squares solution c_i provides the orthogonal projection $\boldsymbol{x}^i c_i$ of $\boldsymbol{R}$ onto $\boldsymbol{x}_i$. Thus, $\boldsymbol{x}^i c_i$ and $\boldsymbol{R} - \boldsymbol{x}^i c_i$ are orthogonal, i.e. $\langle \boldsymbol{R} - \boldsymbol{x}^i c_i, \boldsymbol{x}^i c_i\rangle = 0$, see Eq. 1.3. This allows us to use the Pythagorean theorem:

$$\begin{align}
\|\boldsymbol{R} - \boldsymbol{x}^r c_r + \boldsymbol{x}^r c_r\|^2 + \|\boldsymbol{x}^i c_i\|^2 \leq \|\boldsymbol{R} - \boldsymbol{x}^i c_i \ + \boldsymbol{x}^i c_i\|^2 + \|\boldsymbol{x}^r c_r\|^2 \tag{3.35}\\
\Leftrightarrow \|\boldsymbol{R}\|^2 + \|\boldsymbol{x}^i c_i\|^2 \leq \|\boldsymbol{R}\|^2 + \|\boldsymbol{x}^r c_r\|^2 \tag{3.36}\\
\Leftrightarrow \|\boldsymbol{x}^i c_i\|^2 \leq \|\boldsymbol{x}^r c_r\|^2 \tag{3.37}\\
\Leftrightarrow \|\boldsymbol{x}^i\|^2 |c_i|^2 \leq \|\boldsymbol{x}^r\|^2 |c_r|^2 \tag{3.38}\\
\tag{3.39}
\end{align}$$

Recall that the orthogonal projection coefficient is given by

$$c_i = \frac{\langle \boldsymbol{R}, \boldsymbol{x}^i\rangle}{\|\boldsymbol{x}^i\|^2}. \tag{3.40}$$

Using that and assuming furthermore that all columns have same lengths $\|\boldsymbol{x}^i\|$ (that is the case if they are for example standardized), we obtain:

$$\begin{align}
\|\boldsymbol{x}^i\|^2 \frac{|\langle \boldsymbol{R}, \boldsymbol{x}^i\rangle|^2}{\|\boldsymbol{x}^i\|^4} \leq \|\boldsymbol{x}^r\|^2 \frac{|\langle \boldsymbol{R}, \boldsymbol{x}^r\rangle|^2}{\|\boldsymbol{x}^r\|^4} \tag{3.41}\\
\Leftrightarrow \frac{|\langle \boldsymbol{R}, \boldsymbol{x}^i\rangle|^2}{\|\boldsymbol{x}^i\|^2} \leq \frac{|\langle \boldsymbol{R}, \boldsymbol{x}^r\rangle|^2}{\|\boldsymbol{x}^r\|^2} \tag{3.42}\\
\Leftrightarrow |\langle \boldsymbol{R}, \boldsymbol{x}^i\rangle|^2 \leq |\langle \boldsymbol{R}, \boldsymbol{x}^r\rangle|^2 \tag{3.43}\\
\Leftrightarrow |\langle \boldsymbol{R}, \boldsymbol{x}^i\rangle| \leq |\langle \boldsymbol{R}, \boldsymbol{x}^r\rangle|. \tag{3.44}
\end{align}$$

This means that in each iteration, the closest column $\boldsymbol{x}^i$ to the residual $\boldsymbol{R}$ is given by the highest absolute value of the projection score $|\langle \boldsymbol{R}, \boldsymbol{x}^i \rangle|$, i.e. the calculation expense at each iteration is only given by calculating $\boldsymbol{X}^T\boldsymbol{R}$ and determining the index with the highest absolute value.
The OMP algorithm is given by:

1. Initialize $\boldsymbol{R}_1 = \boldsymbol{y}$ and $\mathcal{X} = \emptyset$ the set of saved columns. Chose a target number q of selected columns (or nonzero coefficients) for the linear model and let the iteration counter $k = 1$.

2. Find the closest $\boldsymbol{x}^i$ to $\boldsymbol{R}_k$ and add it to $\mathcal{X}$.

3. Build the matrix $\tilde{\boldsymbol{X}}$ with all $\boldsymbol{x}^i \in \mathcal{X}$. Calculate $\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}\in\mathbb{R}^k} \|\boldsymbol{y} - \tilde{\boldsymbol{X}}\boldsymbol{c}\|^2$. Let $\boldsymbol{R}_{k+1} = \boldsymbol{y} - \tilde{\boldsymbol{X}}\boldsymbol{c}^*$.

4. If $k = q$, stop. Otherwise set $k = k + 1$ and return to the 2. step.

3.4 Kernel ridge regression

The kernel ridge regression is a generalization of the linear ridge regression 3.11 towards considering nonlinar models f. Consider again the general optimization problem in Eq. 3.1 with the squared error loss $L(f(\boldsymbol{x}_i), y_i) = (f(\boldsymbol{x}_i) - y_i)^2$ and, moreover, $r(f) = \|f\|^2_{\mathcal{F}}$ for a reproducing kernel Hilbert space $\mathcal{F}$[1]. Then, according to the representer theorem, the solution to the optimization problem has the form

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, \boldsymbol{x}_i) \tag{3.45}$$

where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is the associated positive-definite real-valued kernel on our input space $\mathcal{X}$. It can be shown that the coefficients $\{\alpha_i\}$ are a closed-form solution $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$ to the optimization problem (kernel ridge regression)

$$\arg\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|^2 + \lambda\boldsymbol{\alpha}^T\boldsymbol{K}\boldsymbol{\alpha}, \tag{3.46}$$

where $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ represents the matrix with elements $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The fact that the non-linear function 3.45 is learnt by a linear learning algorithm, i.e. the closed-form expression, is considered as the *kernel trick*. For a linear kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$ the model 3.45 becomes a linear function and equals the linear model that results from the optimization problem 3.11 of the linear ridge regression. For instance, the kernel ridge regression can alternatively be derived by kernelizing the linear ridge regression and, moreover, introducing a potentially nonlinear feature map ϕ such that $k(\boldsymbol{x}_i, \boldsymbol{x}_j) =< \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) >$. However, ϕ does not need to be known and rather the kernel is specified directly. A typical (nonlinear)

[1]A Hilbert space is a reproducing kernel Hilbert space if it has a bounded linear evaluation function. For more details we refer to the corresponding literature of functional analysis.

kernel is given by the Gaussian kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2})$. Note that the kernel function is often viewed as a similarity measurement between two data points.
In practice, we solve the linear system $\boldsymbol{K\alpha} = \boldsymbol{y}$ using the Cholesky decomposition and forward and back substitution as described in section 1.6.

3.5 Logistic regression

Consider the binary classification problem with $y_i \in \{0, 1\}$. The logistic regression model

$$f(\boldsymbol{x}, y) = P(y|\boldsymbol{x}; \boldsymbol{c}) = P(Y = 1|\boldsymbol{x}; \boldsymbol{c})^y P(Y = 0|\boldsymbol{x}; \boldsymbol{c})^{(1-y)} \tag{3.47}$$

calculates the probability of a data point $\boldsymbol{x}$ for having the label y, where

$$P(Y = 1|\boldsymbol{x}; \boldsymbol{c}) = \frac{1}{1 + e^{-\boldsymbol{x}^T\boldsymbol{c}}} \tag{3.48}$$

$$P(Y = 0|\boldsymbol{x}; \boldsymbol{c}) = 1 - P(Y = 1|\boldsymbol{x}; \boldsymbol{c}) = 1 - \frac{1}{1 + e^{-\boldsymbol{x}^T\boldsymbol{c}}}. \tag{3.49}$$

Assuming that the $\boldsymbol{x}_i$ are independent and identically distributed random variables, their joint probability distribution is given by

$$P(y_1, y_2, ..., y_n|\boldsymbol{x_1}, \boldsymbol{x}_2, ..., \boldsymbol{x}_n|\boldsymbol{c}) = \prod_{i=1}^{n} P(y_i|\boldsymbol{x_i}; \boldsymbol{c}). \tag{3.50}$$

The right side is equal to the likelihood function $L(\boldsymbol{c}|y_1, y_2, ..., y_n|\boldsymbol{x_1}, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ which, however, treats $\boldsymbol{c}$ as a parameter and the input-output pairs as given. In order to determine $\boldsymbol{c}$ such that the observed data is most probable, L is maximized or, in practice, $\log L$:

$$\log L(\boldsymbol{c}|y_1, y_2, ..., y_n|\boldsymbol{x_1}, \boldsymbol{x}_2, ..., \boldsymbol{x}_n) = \log \prod_{i=1}^{n} P(y_i|\boldsymbol{x_i}; \boldsymbol{c}) \tag{3.51}$$

$$= \sum_{i=1}^{n} \log P(y_i|\boldsymbol{x_i}; \boldsymbol{c}). \tag{3.52}$$

We solve the optimization problem

$$\max_{\boldsymbol{c} \in \mathbb{R}^m} \sum_{i=1}^{n} \log P(y_i|\boldsymbol{x_i}; \boldsymbol{c}) \tag{3.53}$$

via gradient descent.

3.6 Support vector machines

Consider the binary classification problem with $y_i \in \{1, -1\}$. Let us target the goal to find some hyperplane defined by

$$\boldsymbol{x}\boldsymbol{w} - b = 0 \tag{3.54}$$

that separates the two classes in the input space of the data represented by $\boldsymbol{X}$. In Eq. 3.54, $\boldsymbol{w}$ denotes the normal vector to the hyperplane and $\frac{b}{\|\boldsymbol{w}\|}$ the offset of the hyperplane from the origin along $\boldsymbol{w}$. Assume for the moment that the data is linearly separable. The SVM determines $\boldsymbol{w}$ and b (there is at least one with $y = 1$ and one with $y = -1$) such that the closest data points (there is at least one with $y = 1$ and one with $y = -1$) to the hyperplane have a distance to it of $\frac{1}{\|\boldsymbol{w}\|}$ along $\boldsymbol{w}$ if $y = 1$ and along $-\boldsymbol{w}$ if $y = -1$:

$$\boldsymbol{x}_i\boldsymbol{w} - b \geq 1 \quad \text{if} \quad y_i = 1, \tag{3.55}$$

$$\boldsymbol{x}_i\boldsymbol{w} - b \leq -1 \quad \text{if} \quad y_i = -1 \tag{3.56}$$

for all data points i. This can be rewritten:

$$y_i(\boldsymbol{x}_i\boldsymbol{w} - b) \geq 1. \tag{3.57}$$

That means the closest data points $\boldsymbol{x}_i$ (support vectors) lie on two hyperplanes with the same normal vector $\boldsymbol{w}$ which define a margin of $\frac{2}{\|\boldsymbol{w}\|}$ between the two classes.
In case the data is not linearly separable, we need to introduce a set of slack variables $\xi_i \geq 0$ which allow for misclassified data points:

$$\boldsymbol{x}_i\boldsymbol{w} - b \geq 1 - \xi_i \quad \text{if} \quad y_i = 1, \tag{3.58}$$

$$\boldsymbol{x}_i\boldsymbol{w} - b \leq -1 + \xi_i \quad \text{if} \quad y_i = -1 \tag{3.59}$$

for all data points i. This can be rewritten:

$$y_i(\boldsymbol{x}_i\boldsymbol{w} - b) \geq 1 - \xi_i. \tag{3.60}$$

Together with the further criterion that the margin between the two classes should be maximal, the SVM-optimization problem becomes:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to} \quad y_i(\boldsymbol{x}_i\boldsymbol{w} - b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0, \quad \forall i. \tag{3.61}$$

Note that the optimization problem is fully described by those data points that do not fulfill $y_i(\boldsymbol{x}_i\boldsymbol{w} - b) > 1$. A consequence of minimizing the slack variables ξ_i is that they will be zero for data points that are on the correct side of the hyperplane margin, i.e. $y_i(\boldsymbol{x}_i\boldsymbol{w} - b) \geq 1$. In case of misclassifed data points and also ones that are correctly classified but inside the margin, the consequence of minimizing the ξ_i is the that their sizes are not larger than the respective distances of the positions of the data point along

$\boldsymbol{w}$ to the correct side of the hyperplane margin. However, given the side constraint that ξ_i should be at least the distance to the correct side of the hyperplane margin, we have $\xi_i = 1 - y_i(\boldsymbol{x}_i\boldsymbol{w} - b)$ for misclassified data points. With a large C, we will put more weight on minimizing the slack variables ξ_i. As a result, increasing C will lead to decreasing the margin of the hyperplane, because the distance of the data points which are not on the correct side of the margin needs to become small (or just because less weight is put on minimizing $\|\boldsymbol{w}\|$, i.e. the inverse of the margin). If the model complexity allows such as with a flexible model based on some (nonlinear) kernel (this will be introduced a few paragraphs later), instead of a linear model as discussed till now, increasing C will lead to a more *wiggly* that decreases the number of mislassified data points, deacreasing the number of $\xi_i > 0$. Decreasing C will lead to a *coarser* model that allows for misclassified data points (taken care of by sufficiently large ξ_i), however, might be more robust when predicting the labels of new data points.
We will solve this problem using the method of Lagrange multipliers, where the multipliers $\alpha_i \geq 0$ and $\mu_i \geq 0$, take care of the side constraints. The respective Lagrangian is given by:

$$\begin{aligned}
\mathcal{L} &= \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y_i(\boldsymbol{x}_i\boldsymbol{w} - b) - 1 + \xi_i] - \sum_{i=1}^{n}\mu_i\xi_i &(3.62)\\
&= \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \boldsymbol{w}\sum_{i=1}^{n}\alpha_i y_i\boldsymbol{x}_i + b\sum_{i=1}^{n}\alpha_i y_i + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\alpha_i\xi_i - \sum_{i=1}^{n}\mu_i\xi_i &(3.63)\\
&= \frac{1}{2}\|\boldsymbol{w}\|^2 - \boldsymbol{w}\sum_{i=1}^{n}\alpha_i y_i\boldsymbol{x}_i + b\sum_{i=1}^{n}\alpha_i y_i + \sum_{i=1}^{n}\alpha_i + \sum_{i=1}^{n}(C - \alpha_i - \mu_i)\xi_i &(3.64)
\end{aligned}$$

Minimizing the Lagrangian with respect to $\boldsymbol{w}$ and b and ξ_i yields:

$$\frac{\partial\mathcal{L}_\mathrm{P}}{\partial\boldsymbol{w}} = 0 \quad \Rightarrow \boldsymbol{w} = \sum_{i=1}^{n}\alpha_i y_i\boldsymbol{x}_i, \tag{3.65}$$

$$\frac{\partial\mathcal{L}_\mathrm{P}}{\partial b} = 0 \quad \Rightarrow \sum_{i=1}^{n}\alpha_i y_i = 0. \tag{3.66}$$

$$\frac{\partial\mathcal{L}_\mathrm{P}}{\partial\xi_i} = 0 \quad \Rightarrow C - \alpha_i - \mu_i = 0. \tag{3.67}$$

By substituting Eq. 3.65, 3.66, and 3.67 into 3.64, we obtain a new Lagrangian:

$$\mathcal{L}_\mathrm{D} = \frac{1}{2}\|\boldsymbol{w}\|^2 - \boldsymbol{w}\sum_{i=1}^{n}\alpha_i y_i \boldsymbol{x}_i + b\underbrace{\sum_{i=1}^{n}\alpha_i y_i}_{=0} + \sum_{i=1}^{n}\alpha_i + \sum_{i=1}^{n}\underbrace{(C - \alpha_i - \mu_i)}_{=0}\xi_i \tag{3.68}$$

$$= \frac{1}{2}\|\boldsymbol{w}\|^2 - \boldsymbol{w}\sum_{i=1}^{n}\alpha_i y_i \boldsymbol{x}_i + \sum_{i=1}^{n}\alpha_i \tag{3.69}$$

$$= \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i\boldsymbol{x}_j - \sum_{i,j}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i\boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i \tag{3.70}$$

$$= -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \boldsymbol{x}_i\boldsymbol{x}_j + \sum_{i=1}^{n}\alpha_i \tag{3.71}$$

$$= -\frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{H}\boldsymbol{\alpha} + \sum_{i=1}^{n}\alpha_i, \tag{3.72}$$

where $H_{ij} = y_i y_j \boldsymbol{x}_i\boldsymbol{x}_j$. With that and the fact that Eq. 3.67 and $\mu_i > 0$ lead to $0 \leq \alpha_i \leq C$, the optimization problem becomes the dual problem

$$\max_{\boldsymbol{\alpha}}\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{H}\boldsymbol{\alpha} \quad \text{subject to} \quad 0 \leq \alpha_i \leq C \;\; \forall i \quad \text{and} \quad \sum_{i=1}^{n}\alpha_i y_i = 0, \tag{3.73}$$

which can be solved via quadratic programming, see Chapter 2. Recall the Lagrange duality theory: the primal problem is given by

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}}\left(\max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\mu}\geq 0}\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha},\boldsymbol{\mu},\boldsymbol{\xi})\right) = \min_{\boldsymbol{w},b,\boldsymbol{\xi}}\mathcal{L}_\mathrm{P} \tag{3.74}$$

and the dual one by

$$\max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\mu}\geq 0}\left(\min_{\boldsymbol{w},b,\boldsymbol{\xi}}\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha},\boldsymbol{\mu},\boldsymbol{\xi})\right) = \max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\mu}\geq 0}\mathcal{L}_\mathrm{D}. \tag{3.75}$$

Note that when considering the side constraints, the maximum of the $\mathcal{L}_\mathrm{D}$ with respect to $\boldsymbol{\alpha}$ (and $\boldsymbol{\mu}$) is $\frac{1}{2}\|\boldsymbol{w}\|^2$. Assume for a given $\boldsymbol{w}$ and b, some data points fulfill $y_i(\boldsymbol{x}_i\boldsymbol{w} - b) > 1 - \xi_i$ (then also $y_i(\boldsymbol{x}_i\boldsymbol{w} - b) > 1$ because ξ_i is minimized). Then for the α_i-based term in Eq. 3.62, we have:

$$-\sum_{i=1}^{n}\alpha_i[y_i(\boldsymbol{x}_i\boldsymbol{w} - b) - 1 + \xi_i] < 0. \tag{3.76}$$

Maximizing with respect to α_i yields $\alpha_i = 0$ for these (non-support-vector) data points. We define the set S of support vectors $\boldsymbol{x_s}$ which are characterized by $\alpha_s > 0$. We rewrite the weight vector as a linear combination of the support vectors, i.e. we rewrite Eq. 3.65 to

$$\boldsymbol{w} = \sum_{s\in S}^{n}\alpha_s y_s \boldsymbol{x}_s. \tag{3.77}$$

Given furthermore that the support vectors fulfill $y_i(\boldsymbol{x}_i\boldsymbol{w}-b) = 1$, we obtain from

$$y_i = \underbrace{y_i^2}_{=1}(\boldsymbol{x}_i\boldsymbol{w} - b) \tag{3.78}$$

$$= \boldsymbol{x}_i \sum_{s\in S}^{n} \alpha_s y_s \boldsymbol{x}_s - b \tag{3.79}$$

a way to calculate b:

$$b = \sum_{s\in S}^{n} \alpha_s y_s \boldsymbol{x}_i \boldsymbol{x}_s - y_i. \tag{3.80}$$

The model can be extended to a nonlinear decision boundary by kernelizing the linear problem (see kernel ridge regression section) by introducing a potentially nonlinear feature map $\phi(\boldsymbol{x}_i)$ and a kernel given by an inner product $k(\boldsymbol{x}_i, \boldsymbol{x}_j) =< \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) >$. Then $H_{ij} = y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and the original problem can be reproduced by considering a linear kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \boldsymbol{x}_j$.