Machine Learning I: Foundations

By: Prof. Marius Kloft

Contents

About the lecturer		5
Requirements		6
Objectives of the textbook		6
Credits		6
1	**Mathematical Notation & Basics**	**7**
1.1	Notation	7
1.2	Scalar Projection	8
1.3	Hyperplanes	9
1.4	Eigenvalues	10
1.5	Positive Definite Matrices	11
1.6	Gradient	12
1.7	Hessian Matrix	12
1.8	Exercises	13
2	**Linear Classifiers & SVM**	**15**
2.1	Formal Setting	15
2.2	Linear Classifier	16
2.3	Support Vector Machine (SVM)	17
2.4	Exercises	20
3	**Convex Optimization - The Gradient Descent Algorithm**	**23**
3.1	Convex Functions	23
3.2	Convex Optimization Problems	24
3.3	Making the SVM Convex	25
3.4	Solving Convex Optimization Problems	27
3.5	Applying Gradient Descent on the Convex SVM	29
3.6	Exercises	31
4	**Kernel SVM**	**33**
4.1	Kernel Methods	33
4.2	Applying the Kernel Trick to the SVM	36
4.3	Exercises	39
5	**Neural Networks**	**41**
5.1	The Brain Graph	41
5.2	Artifical Neural Networks (ANN)	42
5.3	Feed-Forward ANN	43
5.4	Training ANN	45
5.5	Convolutional Neural Networks (CNN)	47
5.6	CNN Architecture	48

5.7 Deep Learning 50
5.8 Exercises 51

6 Overfitting & Regularization 53
6.1 Overfitting & Underfitting 53
6.2 Unifying Loss View 54
6.3 Regularization 56
6.4 Regularization in Deep Learning 57
6.5 Exercises 60

7 Regression 61
7.1 Linear Regression 61
7.2 Leave-one-out cross-validation (LOOCV) 63
7.3 Non-Linear Regression 66
7.4 Unifying View 69
7.5 Exercises 69

8 Clustering 71
8.1 Linear Clustering 71
8.2 K-means 72
8.3 Non-Linear Clustering 75
8.4 Hierarchical Clustering 77
8.5 Exercises 79

9 Dimensionality Reduction 81
9.1 Linear Dimensionality Reduction 81
9.2 Kernel PCA 84
9.3 Autoencoders 85
9.4 Exercises 87

10 Decision Trees & Random Forests 89
10.1 Decision Trees 89
10.2 Random Forests 93

Bibliography 95
Index 96

About the Lecturer
Prof. Dr. Marius Kloft

Bio

Since 2017, Marius Kloft has been a professor of computer science at TU Kaiserslautern, Germany. Previously, he was an adjunct faculty member of the University of Southern California (09/2018-03/2019), an assistant professor at HU Berlin (2014-2017), and a joint postdoctoral fellow (2012-2014) at the Courant Institute of Mathematical Sciences and Memorial Sloan-Kettering Cancer Center, New York, working with Mehryar Mohri, Corinna Cortes, and Gunnar Rätsch.

From 2007-2011, he was a PhD student in the machine learning program of TU Berlin, headed by Klaus-Robert Müller. He was co-advised by Gilles Blanchard and Peter L. Bartlett, whose learning theory group at UC Berkeley he visited from 10/2009 to 10/2010. In 2006, he received a Master' degree in mathematics from the University of Marburg, Germany, with a thesis on algebraic geometry.

Research Interests

Marius Kloft is interested in theory and algorithms of statistical machine learning and its applications, especially in statistical genetics, mechanical engineering, and chemical process engineering. He has been working on, e.g., multiple kernel learning, transfer learning, anomaly detection, extreme classification, and adversarial learning. He co-organized workshops on these topics at NIPS 2010, 2013, 2014, 2017, ICML 2016, and Dagstuhl 2018.

His dissertation on Lp-norm multiple kernel learning was nominated by TU Berlin for the Doctoral Dissertation Award of the German Chapter of the ACM (GI). He has received the Google Most Influential Papers Award and the DFG Emmy-Noether Career Award.

Requirements

This course requires a solid foundation in linear algebra and calculus at university level. For guidance on how to brush up on your math for this course, we have set up a frequently updated page that links to various excellent resources[1]. The lecture will be accompanied by theoretical and programming exercises. The theory exercises can be solved using the concepts introduced in the lecture, but the programming exercises require basic familiarity with Python. If you are unsure about your Python knowledge, you may want to have a look at the Google Python tutorial[2], for example. In Section 1, we will briefly go through the basics of mathematics needed for machine learning.

Objectives of the textbook

After attending this lecture, you should be able to

- understand the aim of machine learning;
- know general machine learning algorithms;
- understand how to apply a specific model to a specific problem;
- understand the theoretical background needed for machine learning;
- implement solutions to ML problems both on a high level using popular ML frameworks and on a low level using only linear algebra subroutines.

Credits

Credits go out to the following people, that helped create the script:

- Prof. Marius Kloft for proofreading and supporting the slides and lecture videos that helped create the script.
- Geri Gokaj for converting the slides and lecture videos into this script.
- Billy Joe Franks for proofreading the script and writing the CNN part.
- Tobias Michels for proofreading the script and being a LaTeXwizard.
- The students of the 2021 ML1 course for finding typos and mistakes.

[1] `https://ml.cs.uni-kl.de/guide.html`
[2] `https://developers.google.com/edu/python`

1 Mathematical Notation & Basics

In this chapter, we will introduce some basic mathematical concepts and the notation we will use in this course. Machine learning requires solid mathematical knowledge in linear algebra and multivariate analysis.

1.1 Notation

In this section, we will fix the notation that we will use throughout the course.

- We denote vectors $\mathbf{v} \in \mathbb{R}^d$ by bold letters and scalars $s \in \mathbb{R}$ by regular letters:

$$\mathbf{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix}. \tag{1.1.1}$$

- We denote matrices $A \in \mathbb{R}^{m \times n}$ by capital letters:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & \ddots & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}. \tag{1.1.2}$$

- We define $\mathbf{0}$ (resp. $\mathbf{1}$) as the vector full of zeros (resp. full of ones) with appropriate dimension to the context:

$$\mathbf{0} := \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \mathbf{1} := \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}. \tag{1.1.3}$$

- We denote by I the identity matrix with appropriate dimension to the context:

$$I = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & & \ddots & \\ 0 & 0 & \dots & 1 \end{pmatrix}. \tag{1.1.4}$$

- If $\mathbf{v} \in \mathbb{R}^d$, then we define $\mathbf{v}^\top \in \mathbb{R}^{1 \times d}$ as its transpose:

$$\mathbf{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix}, \quad \mathbf{v}^\top := \begin{pmatrix} v_1, \dots, v_d \end{pmatrix}. \tag{1.1.5}$$

If $A \in \mathbb{R}^{m \times n}$, then we define $A^\top \in \mathbb{R}^{n \times m}$ as its transpose:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & \ddots & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}, \quad A^\top := \begin{pmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & & \ddots & \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{pmatrix}. \tag{1.1.6}$$

- Recall that the scalar product of two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^d$ is defined by

$$\langle \mathbf{v}, \mathbf{w} \rangle := \mathbf{v}^\top \mathbf{w} = \sum_{i=1}^{d} v_i w_i. \tag{1.1.7}$$

- The (2-)norm of a vector $\mathbf{v} \in \mathbb{R}^d$ is defined as

$$\|\mathbf{v}\| := \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle} = \sqrt{\mathbf{v}^\top \mathbf{v}} = \sqrt{\sum_{i=1}^{d} v_i^2}. \tag{1.1.8}$$

- Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^d$. Then we denote

$$\mathbf{v} \leq \mathbf{w} : \Longleftrightarrow \ \forall i = 1, \dots, d : v_i \leq w_i. \tag{1.1.9}$$

- For a set S, we denote the cardinality of S (the number of elements in S) by $|S|$.

- Let $\mathbf{v_1}, \mathbf{v_2} \dots \mathbf{v_n} \in \mathbb{R}^d$, then we denote the span of the vectors by

$$\operatorname{span}(\{\mathbf{v_1}, \mathbf{v_2} \dots \mathbf{v_n}\}) = \{\sum_{i=1}^{n} \lambda_i \mathbf{v_i} : \forall i = 1, \dots, n : \lambda_i \in \mathbb{R}\}. \tag{1.1.10}$$

- The sign function is defined as

$$\begin{aligned} \operatorname{sign} : \mathbb{R} &\longrightarrow \{-1, +1\} \\ x &\longmapsto \begin{cases} 1 & x \geq 0 \\ -1 & x < 0. \end{cases} \end{aligned} \tag{1.1.11}$$

- For a quadratic matrix $A \in \mathbb{R}^{m \times m}$ with entries a_{ij}, we denote the trace by

$$\operatorname{tr}(A) := \sum_{i=1}^{m} a_{ii} \tag{1.1.12}$$

1.2 Scalar Projection

scalar projection

Definition 1.2.1 (Scalar Projection)**.** Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^d$ with $\mathbf{w} \neq \mathbf{0}$. The *scalar projection* of $\mathbf{v}$ onto $\mathbf{w}$ is defined as $\Pi_{\mathbf{w}}(\mathbf{v}) := \dfrac{\mathbf{w}^\top \mathbf{v}}{\|\mathbf{w}\|}$.

We observe that the scalar projection is a scalar, not a vector. This quantity also has a nice geometric interpretation[3] as illustrated in Fig. 1.2.1.

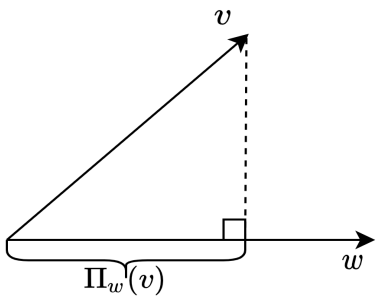

Figure 1.2.1: Geometrical illustration of the scalar projection: Given the vectors $\mathbf{v}$ and $\mathbf{w}$, here $\Pi_{\mathbf{w}}(\mathbf{v})$ is the scalar projection of $\mathbf{v}$ onto $\mathbf{w}$.

1.3 Hyperplanes

Definition 1.3.1 ((Affine-)linear Function)**.** An *(affine-)linear function* is a function $f : \mathbb{R}^d \to \mathbb{R}$ of the form $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$, where $\mathbf{w} \in \mathbb{R}^d \backslash \{0\}$ and $b \in \mathbb{R}$. For the sake of simplicity, we call (affine-)linear functions just *linear functions.* **(affine-)linear function**

▼

Example 1.1. The function

$$\begin{aligned} f &: \mathbb{R}^2 \to \mathbb{R} \\ &\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mapsto 3x_1 + 4x_2 - 7 \end{aligned}$$

is (affine-)linear as it is of the form $f(x) = \begin{pmatrix} 3 \\ 4 \end{pmatrix}^\top \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 7.$

▲

Definition 1.3.2 (Hyperplane)**.** A *hyperplane* is a subset $H \subset \mathbb{R}^d$ defined as $H := \left\{\mathbf{x} \in \mathbb{R}^d : f(\mathbf{x}) = 0\right\}$, where f is (affine-)linear. **hyperplane**

Proposition 1.3.3. *Let H be a hyperplane defined by the affine-linear function $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$. The vector $\mathbf{w}$ is orthogonal to H, meaning that: $\forall \mathbf{x}_1, \mathbf{x}_2 \in H$ $\mathbf{w}^\top (\mathbf{x}_1 - \mathbf{x}_2) = 0$ holds.*

Proof. Let $\mathbf{x}_1, \mathbf{x}_2 \in H = \left\{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top \mathbf{x} + b = 0\right\}$. Then $\mathbf{w}^T \mathbf{x}_1 + b = 0$ and $\mathbf{w}^T \mathbf{x}_2 + b = 0$. Thus

$$\mathbf{w}^T \mathbf{x}_1 + b = \mathbf{w}^T \mathbf{x}_2 + b \iff \mathbf{w}^T (\mathbf{x}_1 - \mathbf{x}_2) = 0.$$

[3] If you are looking for a proof, see: `https://www.youtube.com/watch?v=LyGKycYT2v0`.

□

signed distance **Definition 1.3.4** (Signed Distance)**.** The *signed distance* of a point x to H is given by

$$d(\mathbf{x}, H) := \operatorname{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \min_{\tilde{x} \in H} \|x - \tilde{x}\| . \tag{1.3.1}$$

Observe how (1.3.1) implies that

$$|d(\mathbf{x}, H)| = \min_{\tilde{x} \in H} \|x - \tilde{x}\| .$$

As the name suggests, the signed distance gives us the distance from a point to the hyperplane. It is positive if it is towards the normal and negative elsewise.

Proposition 1.3.5. *For the signed distance, we have that:*

$$d(\mathbf{x}, H) = \frac{1}{\|\mathbf{w}\|} \left(\mathbf{w}^\top \mathbf{x} + b\right) . \tag{1.3.2}$$

Proof. Let $\tilde{\mathbf{x}}$ be an arbitrary element of H. By our previous proof, we know that $\mathbf{w}$ is orthogonal to our hyperplane. First we notice from Fig. 1.3.1 that

$$\forall \tilde{\mathbf{x}} \in H : \Pi_{\mathbf{w}}(\mathbf{x} - \tilde{\mathbf{x}}) = \operatorname{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) \min_{\tilde{\mathbf{x}} \in H} \|\mathbf{x} - \tilde{\mathbf{x}}\|,$$

where $\operatorname{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right)$ indicates on which side of the hyperplane $\mathbf{x}$ lies. So

$$d(\mathbf{x}, H) = \Pi_w(\mathbf{x} - \tilde{\mathbf{x}}) \overset{\text{def.}}{=} \frac{\mathbf{w}^\top(\mathbf{x} - \tilde{\mathbf{x}})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^\top \mathbf{x} - \mathbf{w}^\top \tilde{\mathbf{x}}}{\|\mathbf{w}\|} \overset{(\star)}{=} \frac{\mathbf{w}^\top \mathbf{x} + b}{\|\mathbf{w}\|},$$

where in $(\star)$ we use: $\tilde{\mathbf{x}} \in H \Rightarrow \mathbf{w}^\top \tilde{\mathbf{x}} + b = 0 \Longleftrightarrow -\mathbf{w}^\top \tilde{\mathbf{x}} = +b$. □

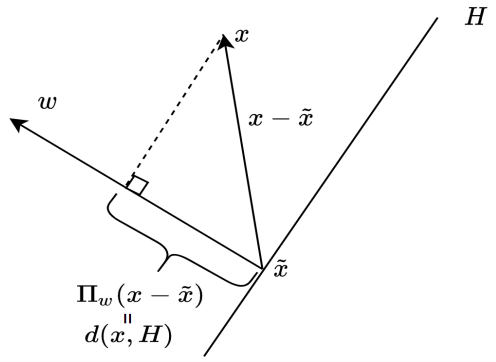

Figure 1.3.1: The hyperplane and the vectors are represented geometrically. Crucial for the proof of Proposition 1.3.5 is the observation that $d(\mathbf{x}, H) = \Pi_w(\mathbf{x} - \tilde{\mathbf{x}})$.

1.4 Eigenvalues

eigenvalue **Definition 1.4.1** (Eigenvalue)**.** Let $A \in \mathbb{R}^{d \times d}$. $\lambda \in \mathbb{R}$ is called an *eigenvalue*

of A if there is a vector $\mathbf{x} \in \mathbb{R}^d \setminus \{0\}$ such that $A\mathbf{x} = \lambda\mathbf{x}$. In that case, $\mathbf{x}$ is an *eigenvector* corresponding to the eigenvalue λ. The set

$$\mathrm{Eig}(A, \lambda) := \left\{\mathbf{x} \in \mathbb{R}^d : A\mathbf{x} = \lambda\mathbf{x}\right\} \tag{1.4.1}$$

is called the *eigenspace* corresponding to the eigenvalue λ.

Intuitively, an eigenvector preserves its direction under the linear transformation A but not necessarily its magnitude.

▼———————————————

Example 1.2. [Eigenvalue Calculation] Let

$$B = \begin{pmatrix} -6 & 3 \\ 4 & 5 \end{pmatrix}.$$

Let us try to find the eigenvalues of the matrix B. Let I denote the identity matrix. For $\lambda \in \mathbb{R}$ and $A \in \mathbb{R}^{d\times d}$, it holds:

$$\begin{aligned}
\lambda \text{ eigenvalue of } A &\iff \exists\, \mathbf{x} \neq \mathbf{0} \in \mathbb{R}^d \text{ with } A\mathbf{x} = \lambda\mathbf{x} \\
&\iff \exists\, \mathbf{x} \neq \mathbf{0} \in \mathbb{R}^d \text{ with } A\mathbf{x} = \lambda I\mathbf{x} \\
&\iff \exists\, \mathbf{x} \neq \mathbf{0} \in \mathbb{R}^d \text{ with } \lambda I\mathbf{x} - A\mathbf{x} = 0 \\
&\iff \dim \mathrm{Ker}(\lambda I - A) > \mathbf{0} \\
&\iff \dim \mathrm{Im}(\lambda I - A) < d \\
&\iff \lambda I - A \text{ not invertible} \\
&\iff \det(\lambda I - A) = \mathbf{0}.
\end{aligned}$$

Let us use this insight to calculate the eigenvalues of B:

$$\begin{aligned}
\det \begin{pmatrix} \lambda + 6 & 3 \\ 4 & \lambda - 5 \end{pmatrix} = 0 &\iff (\lambda + 6)(\lambda - 5) - 12 = 0 \\
&\iff \lambda^2 + \lambda - 42 = 0 \\
&\iff (\lambda + 7)(\lambda - 6) = 0.
\end{aligned}$$

Apparently, the eigenvalues of A are -7 and 6. We can also find the corresponding eigenvectors by resubstituting these values back into $A\mathbf{x} = \lambda\mathbf{x}$.

▲———————————————

1.5 Positive Definite Matrices

Definition 1.5.1 (Positive Definiteness)**.** Let $A \in \mathbb{R}^{d\times d}$ be a symmetric matrix ($A^\top = A$). We call

positive definiteness

$$A \textit{ positive definite} : \iff \forall \mathbf{x} \neq \mathbf{0} \in \mathbb{R}^d : \ \mathbf{x}^\top A\mathbf{x} > 0 \tag{1.5.1}$$

$$A \textit{ negative definite} : \iff \forall \mathbf{x} \neq \mathbf{0} \in \mathbb{R}^d : \ \mathbf{x}^\top A\mathbf{x} < 0 \tag{1.5.2}$$

$$A \textit{ positive semi-definite} : \iff \forall \mathbf{x} \neq \mathbf{0} \in \mathbb{R}^d : \ \mathbf{x}^\top A\mathbf{x} \geq 0 \tag{1.5.3}$$

$$A \textit{ negative semi-definite} : \iff \forall \mathbf{x} \neq \mathbf{0} \in \mathbb{R}^d : \ \mathbf{x}^\top A\mathbf{x} \leq 0. \tag{1.5.4}$$

1.6 Gradient

gradient

Definition 1.6.1 (Gradient)**.** Let $f : \mathbb{R}^d \to \mathbb{R}$ be differentiable. We define the *gradient* by:

$$\nabla f = \operatorname{grad} f := \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)^\top. \tag{1.6.1}$$

We observe that ∇f is again a function. Each component of the gradient tells us how fast our function changes in each direction. To see how fast the change is at a point $\mathbf{p}$ into direction $\mathbf{v}$, we multiply $\nabla f(\mathbf{p}) \cdot \mathbf{v}$. Observe that this scalar product is maximized if $\mathbf{v}$ is parallel to $\nabla f(\mathbf{p})$, which shows that ∇f points towards the direction of the steepest ascent.

1.7 Hessian Matrix

hessian matrix

Definition 1.7.1 (Hessian Matrix)**.** Let $f : \mathbb{R}^d \to \mathbb{R}$. If all second partial derivatives of f exist and are continuous over the domain of the function, then the *Hessian matrix* (also simply called the *Hessian*) is defined by:

$$\mathrm{H}_f(\mathbf{x}) := \left(\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x})\right)_{i,j=1,\ldots,d} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\mathbf{x}) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_2 \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_n \partial x_2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n}(\mathbf{x}) \end{pmatrix}. \tag{1.7.1}$$

▼———————————————

Example 1.3. [Hessian Matrix] Let $f : \mathbb{R}^2 \to \mathbb{R}, f(x, y) = x^3 + y^3 - 3xy$. Let us try to calculate the Hessian matrix of f in a point (x, y). First we need to find the partial derivatives. We have:

$$\frac{\partial f}{\partial x}(x, y) = 3x^2 - 3y$$
$$\frac{\partial f}{\partial y}(x, y) = 3y^2 - 3x.$$

We know that

$$\frac{\partial^2 f}{\partial x \partial x} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right) = 6x$$
$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right) = -3$$
$$\frac{\partial^2 f}{\partial y \partial x} = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial x}\right) = -3$$
$$\frac{\partial^2 f}{\partial y \partial y} = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial y}\right) = 6y.$$

With this information, we can now state the Hessian matrix:

$$H = \begin{pmatrix} 6x & -3 \\ -3 & 6y \end{pmatrix}.$$

▲———————————————

Theorem 1.7.2. *Let $H_f(\mathbf{x})$ be the Hessian (matrix) of a function f in a point $\mathbf{x}$, as defined in Definition 1.7.1. Then the following statements hold:*

- *For any $\mathbf{x}$, $H_f(\mathbf{x})$ is symmetric.*
- *If f is a convex function, then $H_f(\mathbf{x})$ is positive semi-definite for any $\mathbf{x}$.*
- *If $H_f(\mathbf{x})$ is positive definite at a critical point $\mathbf{x}$ (meaning $\nabla f(\mathbf{x}) = \mathbf{0}$), then f attains an isolated local minimum at $\mathbf{x}$.*
- *If $H_f(\mathbf{x})$ is negative definite at a critical point $\mathbf{x}$, then f attains an isolated local maximum at $\mathbf{x}$.*

We finish the chapter with useful derivatives we will continually use during the course.

Theorem 1.7.3 (Calculation rules for derivatives)**.** *The following statements hold:*

- $\frac{\partial}{\partial \mathbf{x}} \mathbf{c}^\top \mathbf{x} = \mathbf{c}$
- $\frac{\partial}{\partial \mathbf{x}} A\mathbf{x} = A$
- $\frac{\partial}{\partial \mathbf{x}} \mathbf{x}^\top A\mathbf{x} = \left(A + A^\top\right)\mathbf{x}$
- $\frac{\partial}{\partial \mathbf{x}} ||\mathbf{x}||^2 = \frac{\partial}{\partial \mathbf{x}} \mathbf{x}^\top \mathbf{x} = 2\mathbf{x}$.
- *See [11] for further useful properties.*

1.8 Exercises

1. The scalar projection, $\Pi_{\mathbf{w}}(\mathbf{x})$, of a vector $\mathbf{x}$ onto another vector $\mathbf{w}$ is defined as the coordinate of the orthonormal projection of $\mathbf{x}$ onto a line parallel to $\mathbf{w}$. Assume $\mathbf{b} := \mathbf{x} - a_1 \frac{\mathbf{w}}{\|\mathbf{w}\|}$ and $\mathbf{b}^\top \mathbf{w} = 0$, then $\Pi_{\mathbf{w}}(\mathbf{x}) := a_1$. Show that the scalar projection of vector $\mathbf{x}$ onto $\mathbf{w}$ is equal to:

$$\Pi_{\mathbf{w}}(\mathbf{x}) := \frac{1}{\|\mathbf{w}\|} \mathbf{w}^\top \mathbf{x}.$$

2. Let $f(\mathbf{x}) := \mathbf{w}^\top \mathbf{x} + b$, with $\mathbf{w}, \mathbf{x} \in \mathbb{R}^d$, $b \in \mathbb{R}$. Let H be a hyperplane defined as $H := \{\mathbf{x} \in \mathbb{R}^d | f(\mathbf{x}) = 0\}$ and let $\tilde{\mathbf{x}} \in H$. The signed distance of $\mathbf{x} \in \mathbb{R}^d$ to H is $d(\mathbf{x}, H) := \Pi_{\mathbf{w}}(\mathbf{x} - \tilde{\mathbf{x}})$. Show that the signed distance can be equivalently defined as:

$$d(\mathbf{x}, H) := \frac{\mathbf{w}^\top \mathbf{x} + b}{\|\mathbf{w}\|}.$$

3. In this question, we will be exploring eigenvectors and eigenvalues. Let $A \in \mathbb{R}^{d \times d}$. Recall the following definition from linear algebra: A vector $\mathbf{v} \in \mathbb{R}^d$ is an *eigenvector* of A if and only if there exists $\lambda \in \mathbb{R}$ such that $A\mathbf{v} = \lambda\mathbf{v}$. We then call λ the *eigenvalue* of A associated with the vector $\mathbf{v}$. Note that, if $\mathbf{v}$ is an eigenvector of A, then, for any $a \in \mathbb{R}$, $a\mathbf{v}$ is also an eigenvector of A. Therefore, $\mathbf{v}$ and $a\mathbf{v}$ are not considered 'distinct' eigenvectors. Prove the following:

   Proposition 1. *If A has a finite number of distinct eigenvectors, then each eigenvector must have a unique eigenvalue.*

4. Solve the programming assignment Sheet 0.ipynb. You will need to install the Jupyter notebook software to open this file and solve the task. Please visit the Jupyter website[4] for instructions on how to do that. We recommend `conda`[5] for installing and managing Python environments.

[4] `https://jupyter.org/install#getting-started-with-the-classic-jupyter-notebook`
[5] `https://docs.conda.io/en/latest/miniconda.html`

2 Linear Classifiers & SVM

Classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data elements whose category membership is known. In this chapter, we consider a simplification of classification, namely binary classification, as in Fig. 2.3.1. In binary classification, the set of categories has cardinality 2. In the context of binary classification, we consider linear classifiers, which are fairly simple but work well surprisingly often.

2.1 Formal Setting

Let $\mathcal{X}$ (input space) and $\mathcal{Y}$ (label space) be some sets. A given set of the form

$$D := \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \mathcal{X} \times \mathcal{Y}$$

is called training data, where

$$\mathbf{x_1}, \ldots, \mathbf{x_n}$$

are our inputs and $y_1, \ldots, y_n$ are the corresponding labels. We call

$$X := (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$$

the data matrix, whose columns consist of the input data points. Our goal is to write a function $f : \mathcal{X} \to \mathcal{Y}$ (prediction function), which correctly predicts the label of yet unseen data with the help of our training data. We call the elements of $\mathcal{Y}$ classes. If there are finitely many classes, we also call f a classifier. The algorithm used to find such an f with the help of the training data is called *learning machine* or *learning algorithm*. This process is also called *training*. Unless stated otherwise, our setting will be $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1, 1\}$ and $D := \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \mathcal{X} \times \mathcal{Y}$ as our training data. This is the *binary classification* setting.

▼———————————————

Example 2.1. To facilitate understanding, let us go through the aforementioned terms in an example. Consider the following dataset:

Table 2.1: Assume n persons were asked whether they are currently happy with their life in the following imaginary dataset.

Person	**Marital Status**	**Bank Account**	**Happy ?**
John	Married	35,000 €	Yes
Maria	Single	100,000 €	No
⋮	⋮	⋮	⋮
Donald	Single	7,500€	Yes
Peter	Divorced	-2,000€	No

Now, assuming a new person shows up and we know their marital status and bank account status, we would like to predict whether they are currently happy. Let us first encode our dataset. We consider the marital status married (1), single (2), and divorced (3) and finally encode the classifications happy (1) and unhappy (-1). As the person's name is somewhat irrelevant, we discard it altogether. Thus, our input space $\mathcal{X}$ and label space $\mathcal{Y}$ are of the form

$$\mathcal{X} = \{1, 2, 3\} \times \mathbb{R}, \quad \mathcal{Y} = \{-1, +1\}.$$

Our second datapoint (Maria) can now be seen as

$$(\mathbf{x_2}, y_2) = \left(\begin{pmatrix} 2 \\ 100000 \end{pmatrix}, -1 \right).$$

The whole data matrix would be

$$X = \begin{pmatrix} 1 & 2 & \dots & 2 & 3 \\ 35000 & 100000 & \dots & 7500 & -2000 \end{pmatrix}.$$

Given the dataset, one could construct a function $f : \mathcal{X} \to \mathcal{Y}$, which we could use to predict whether a new person is happy. Notice that f would even be a binary classifier in our case, as $|\mathcal{Y}| = 2$. Whether one can actually use marital status and bank account status to predict happiness is irrelevant in our course, but it is always good to have this in the back of our mind. In the following chapters, we will see how we can construct f given $D = \{(\mathbf{x_1}, y_1), \dots, (\mathbf{x_n}, y_n)\}$.

▲———————————————

2.2 Linear Classifier

linear classifier

Definition 2.2.1 (Linear Classifier)**.** A classifier of the form $f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$ is called a *linear classifier*.

▼———————————————

Example 2.2. [Nearest Centroid Classifier] The idea of NCC is the following: Given our training data, look at the two clusters (groups of data points):

$$A := \{\mathbf{x}_i \, : (\mathbf{x}_i, y_i) \in D, \, y_i = -1\}, \quad B := \{\mathbf{x}_i \, : (\mathbf{x}_i, y_i) \in D, \, y_i = +1\}.$$

These clusters partition our training data into two groups. To find the label of a new data point, we check whether the new data point is closer to the centroid of cluster A or the centroid of cluster B and give it the label -1 or $+1$, respectively. See also Fig. 2.2.1.

▲———————————————

Algorithm:

- **Training:** Compute the centroids $\mathbf{c}_{-1} = \frac{1}{|A|} \sum_{x \in A} \mathbf{x}$, $\mathbf{c}_{+1} = \frac{1}{|B|} \sum_{x \in B} \mathbf{x}$.

- **Prediction:** Given a new data point $\mathbf{x_k}$, set $y_k := \arg\min_{l\in\{-1,+1\}} \|\mathbf{c}_l - \mathbf{x_k}\|$.

Theorem 2.2.2. *NCC is a linear classifier.*

Proof. We want to write the condition whether a new data point is closer to $\mathbf{c_{-1}}$ or to $\mathbf{c_{+1}}$ in the form $\text{sign}(\mathbf{w}^\top\mathbf{x}+b=0)$. As NCC classifies points based on which centroid is closer, points that are equidistant to both centroids should be on the sought hyperplane.

So we want that $\mathbf{w}^\top\mathbf{x}+b=0 \iff \|\mathbf{x}-\mathbf{c}_{-1}\| = \|\mathbf{x}-\mathbf{c}_{+1}\|$. By expanding the right side, we get:

$$\begin{aligned}
\|\mathbf{x}-\mathbf{c}_{-1}\| = \|\mathbf{x}-\mathbf{c}_{+1}\| &\iff \|\mathbf{x}-\mathbf{c}_{-1}\|^2 = \|\mathbf{x}-\mathbf{c}_{+1}\|^2 \\
&\iff \|\mathbf{x}\|^2 - 2\mathbf{x}^\top\mathbf{c}_{-1} + \|\mathbf{c}_{-1}\|^2 = \|\mathbf{x}\|^2 - 2\mathbf{x}^\top\mathbf{c}_{+1} + \|\mathbf{c}_{+1}\|^2 \\
&\iff -2\mathbf{x}^\top\mathbf{c}_{-1} + 2\mathbf{x}^\top\mathbf{c}_{+1} + \|\mathbf{c}_{-1}\|^2 - \|\mathbf{c}_{+1}\|^2 = 0 \\
&\iff \underbrace{2\left(\mathbf{c}_{+1}-\mathbf{c}_{-1}\right)^\top}_{=:\mathbf{w}^\top}\mathbf{x} + \underbrace{\|\mathbf{c}_{-1}\|^2 - \|\mathbf{c}_{+1}\|^2}_{=:b} = 0.
\end{aligned}$$

□

This shows that NCC is a linear classifier. Notice that $\mathbf{w} := 2\left(\mathbf{c}_{-1}-\mathbf{c}_{+1}\right)$ and $b := \|\mathbf{c}_{+1}\|^2 - \|\mathbf{c}_{-1}\|^2$ would also yield a linear classifier, but the two hyperplanes would have swapped signed distance. To see which formulation matches with our setting, it suffices to plug in the centroids and see whether the labels match. The NCC is a first example of linear classifiers. There are more to come in the following.

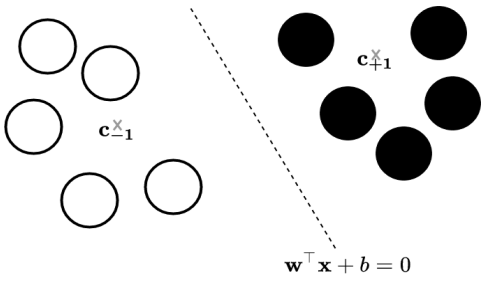

Figure 2.2.1: Visualization of the NCC. The white points represent class -1 with centroid $\mathbf{c_{-1}}$, whereas the black points represent class $+1$ with centroid $\mathbf{c_{+1}}$. The separating hyperplane will be linear and defined as in the proof of Theorem 2.2.2.

2.3 Support Vector Machine (SVM)

We seek the best hyperplane separating our data. To this end, consider Fig. 2.3.1 (left). Intuitively, the best (fairest) hyperplane would be the one that separates the data with the largest margin, visualized in Fig. 2.3.1 (right). We also want our hyperplane to separate our data correctly; namely, we require all points with the label $+1$ to lie "above" the hyperplane and all points with

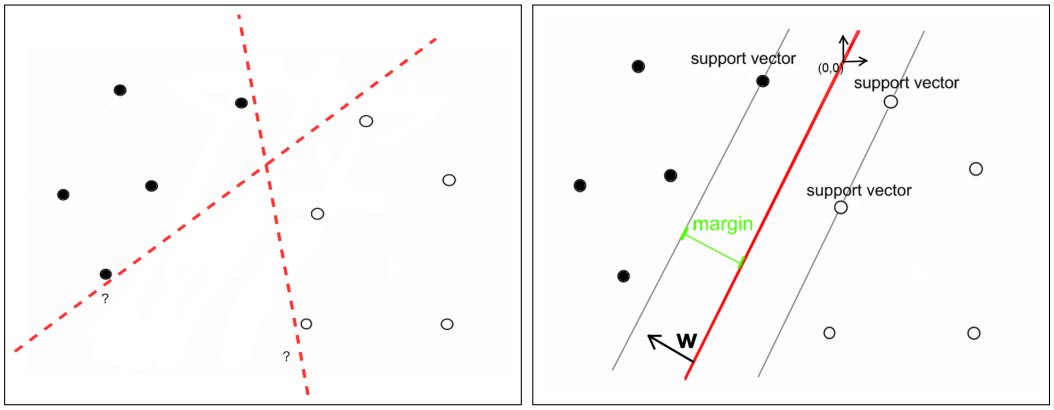

Figure 2.3.1: Visualization of linear classifiers. Data points with the label -1 are black and those with the label $+1$ are white. Left: Two potential classifiers. Right: The hard-margin SVM H and its margin hyperplanes H_+ and H_-.

the label -1 to be "below" this hyperplane. We can formalize this thought mathematically: Assume there exists a line that can separate our data. Let

$$H = \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top \mathbf{x} + b = 0\},$$

with $\mathbf{w} \neq 0$, be a hyperplane. The margin γ is the size of the open space around the hyperplane that no points occupy. Its boundaries on either side can be described by

$$H_+ = \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top \mathbf{x} + b = \gamma\}$$

and

$$H_- = \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top \mathbf{x} + b = -\gamma\}.$$

We additionally require all points with the label $+1$ to be "above" H_+ and all points with the label -1 to be "below" H_-. Our goal is to maximize γ by choosing appropriate $\mathbf{w}$ and b. Recall the signed distance from Proposition 1.3.5:

$$d(\mathbf{x}, H) = \frac{1}{\|\mathbf{w}\|} \left(\mathbf{w}^\top \mathbf{x} + b\right).$$

We wish to not just choose *any* hyperplane; rather, we wish to choose our hyperplane such that the classification is done correctly. Since we will use the signed distance for classification, we need

$$d(\mathbf{x_i}, H) \geq 0 \text{ if } y_i = +1$$

$$d(\mathbf{x_i}, H) \leq 0 \text{ if } y_i = -1.$$

We simplify these two constraints by the new constraint:

$$y_i \cdot d(\mathbf{x_i}, H) \geq 0. \tag{2.3.1}$$

Finally, the distance between any point and the hyperplane should be at least γ, giving the constraint:

$$y_i \cdot d(\mathbf{x_i}, H) \geq \gamma \iff y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq \|\mathbf{w}\|\gamma. \tag{2.3.2}$$

Last, we make the observation that if (2.3.2) is satisfied, (2.3.1) is trivially satisfied because $\gamma \geq 0$ ($\gamma \geq 0$ holds because γ=0 is a trivial lower bound of our maximization problem). This gives us our maximization problem:

$$\begin{aligned} &\max_{\gamma, b \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d \setminus \{0\}} \gamma \\ &\text{s.t. } y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right) \geq \|\mathbf{w}\| \gamma, \quad \forall i = 1, \ldots, n. \end{aligned} \tag{2.3.3}$$

Definition 2.3.1 (Hard-margin SVM)**.** The mathematical program (2.3.3) is called the *hard-margin SVM*.

hard-margin SVM

Let us intuitively recap(2.3.3). Our objective function maximizes the margin, and each of the n constraints makes sure that we classify datapoint $\mathbf{x_i}$ correctly. We managed to achieve the formalization that we wanted. We now state a few problems with the hard-margin SVM. First, data points are not always linearly separable (see Fig. 2.3.2). Also, outlier points can potentially corrupt the SVM

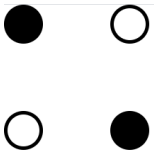

Figure 2.3.2: A set of data points that is not linearly separable

(see Fig. 2.3.3). To fix this issue, we take a closer look at (2.3.3). A first idea

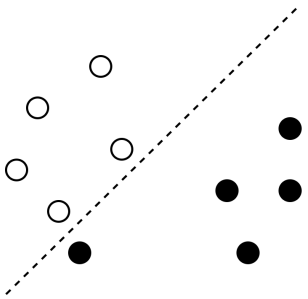

Figure 2.3.3: An outlier point forces us to choose a suboptimal hyperplane.

could be to allow some more freedom in the constraints. Not every data point needs to have a distance of γ to the hyperplane. If we allow for every datapoint a variable

$$\xi_i \geq 0 \quad \forall i = 1, \ldots, n,$$

then we can formalize this by:

$$y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right) \geq \|\mathbf{w}\| \gamma - \xi_i, \quad \forall i = 1, \ldots n.$$

By choosing ξ_i large enough, our problem becomes unbounded. To not let this happen, we need to somehow penalize our maximization function every time we use ξ_i. One way to do so could be:

$$\max_{\gamma, b \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d \setminus \{0\}, \xi_1, \ldots, \xi_n \geq 0} \quad \gamma - C \sum_{i=1}^{n} \xi_i,$$

where C is a constant telling us how harshly to penalize. The higher C, the more we penalize violations. Finally, we get:

$$\begin{aligned} \max_{\gamma,b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d\setminus\{0\},\xi_1,\ldots,\xi_n\geq 0} \quad & \gamma - C\sum_{i=1}^{n}\xi_i \\ \text{s.t. } & y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right) \geq \|\mathbf{w}\|\gamma - \xi_i \quad \forall i = 1,\ldots n \\ & \xi_i \geq 0 \qquad\qquad\qquad\qquad \forall i = 1,\ldots n. \end{aligned} \tag{2.3.4}$$

soft-margin SVM

Definition 2.3.2 (Soft-Margin SVM)**.** The mathematical program (2.3.4) is called the *soft-margin SVM*.

Let us intuitively recap (2.3.4). The objective function wants to maximize the margin while also violating penalizations. The constraints lead to a weakened version of correct classification, allowing a mistake of the size ξ_i. Furthermore, the remaining constraints ensure that the ξ_i are positive. Denote by γ^*, $\mathbf{w}^*$, and $b*$ the optimal arguments of (2.3.4).

support vector

Definition 2.3.3 (Support Vector)**.** All vectors $\mathbf{x}_i$ with $y_i \cdot d\left(\mathbf{x}_i, H\left(\mathbf{w}^*, b^*\right)\right) \leq \gamma^*$ are called *support vectors*.

Observation 2.3.4. We notice that (2.3.4) allows the existence of support vectors. Also notice that the value of (2.3.4) only depends on these support vectors. Removing all other data points would not have an effect on the outcome of (2.3.4) because, if $\mathbf{x_i}$ is not a support vector, then $\xi_i = 0$. Now that we have the optimization problem (2.3.4), let us see in the following chapters how to solve it.

2.4 Exercises

1. Interestingly, the linear hard-margin SVM given by (2.3.3) requires only two (non-equal) training points (with opposite labels) to find a separating hyperplane. Let $X := \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $Y := \{y_1, \ldots, y_n\}$, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$ be a dataset. Let γ^*, $\mathbf{w}^*$, and b^* be the optimal solution to (2.3.3) on X, Y. You may assume $\mathbf{w}^* \neq 0$.

   a) Find a minimal dataset (X', Y') with $|X'| = |Y'| = 2$ (consisting of only two data points) with the same hard-margin SVM solution (2.3.3) as for the dataset (X, Y), that is, γ^*, $\mathbf{w}^*$, and b^*.

   b) Prove that, for your choice of X' and Y' in a), γ, $\mathbf{w}^*$, and b^* are optimal solutions of (2.3.3).

   c) How is this choice of X' and Y' related to the nearest centroid classifier (NCC)? (Answer this question with at most 5 sentences.)

2. Consider the soft-margin SVM as in the lecture. Now assume b=0, i.e, we do not optimize over b. Construct a dataset for which any classifier learned (with $b = 0$) performs poorly. Does any classifier with b fixed to a different constant (like $b = 1$) still perform poorly?

3. Construct a worst-case dataset for the nearest centroid classifier (NCC). This dataset should be easily (not necessarily linearly) separable and the NCC should behave as poorly as possible on this training dataset. **Hint:** To this end, you will have to figure out for yourself how poorly the NCC can perform. Is it possible for the NCC to have 0% accuracy?

4. Solve the programming assignment Sheet 1.ipynb.

3 Convex Optimization - The Gradient Descent Algorithm

In the last chapter, we had a look at optimization problems as formalizations of the classification problem. In this chapter, we will develop the theory and the tools to solve them. The main algorithm in this chapter will be the Gradient Descent Algorithm.

3.1 Convex Functions

Definition 3.1.1 (Convex Set)**.** A set $\mathcal{X} \subset \mathbb{R}^d$ is called *convex* if and only if the line segment connecting any two points in $\mathcal{X}$ entirely lies within $\mathcal{X}$, that is,

convex set

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad \forall \theta \in [0,1] : (1-\theta)\mathbf{x} + \theta \mathbf{x}' \in \mathcal{X}.$$

▼———————————————

Example 3.1. [Hyperplanes are convex sets] Let $H = \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top \mathbf{x} + b = 0\}$ be a hyperplane. We will now show that hyperplanes are convex. Let $\mathbf{x_1}, \mathbf{x_2} \in H$ and $\theta \in [0,1]$. Then $\mathbf{w}^\top \mathbf{x_1} = -b$, $\mathbf{w}^\top \mathbf{x_2} = -b$, and

$$\mathbf{w}^\top((1-\theta)\mathbf{x}_1 + \theta \mathbf{x}_2) + b = (1-\theta)\mathbf{w}^\top \mathbf{x_1} + \theta \mathbf{w}^\top \mathbf{x_2} + b = (1-\theta)(-b) - \theta b + b = 0.$$

▲———————————————

Definition 3.1.2 (Convex Function)**.** A function $f : \mathbb{R}^d \to \mathbb{R}$ is *convex* if and only if the set above the graph is *convex* or equivalently,

convex function

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d, \forall \theta \in [0,1] : f\left((1-\theta)\mathbf{x} + \theta \mathbf{x}_2\right) \leq (1-\theta) f\left(\mathbf{x}_1\right) + \theta f\left(\mathbf{x}_2\right).$$

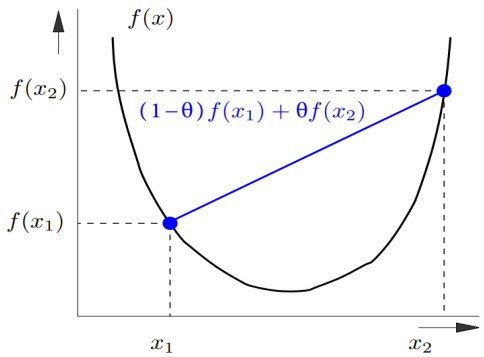

Figure 3.1.1: A convex function.

Definition 3.1.3 (Concave Function)**.** A function f is *concave* if and only if $-f$ is convex.

concave function

Recall from Chapter 1 that f is convex if and only if the Hessian matrix $H_f(\mathbf{x})$ is positive semi-definite for all $\mathbf{x} \in \mathbb{R}^d$.

Theorem 3.1.4. *The following statements regarding a symmetric matrix $M \in \mathbb{R}^{d\times d}$ are equivalent:*

1. *M is positive semi-definite (we write $M \succeq 0$).*
2. *$\mathbf{x}^\top M\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^d$.*
3. *All eigenvalues of M are non-negative.*

▼———————————————

Example 3.2. [Convex Function] Define f as follows:

$$f : \begin{array}{c} \mathbb{R}^2 \to \mathbb{R} \\ (x_1, x_2) \mapsto x_1^2 + x_2^2 - 7. \end{array}$$

Note that

$$\nabla f \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix} \text{ and } H_f(\mathbf{x}) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.$$

Observe that $H_f(\mathbf{x})$ is a diagonal matrix, and we can read the eigenvalues from the diagonal. As the eigenvalues are non-negative, it follows that $H_f(\mathbf{x})$ is positive semi-definite and that f is convex.

▲———————————————

3.2 Convex Optimization Problems

Let $f_0, f_1, \ldots, f_n, g_1, \ldots, g_m : \mathcal{X} \to \mathbb{R}$.

optimization problem

Definition 3.2.1 (Optimization Problem)**.** An *optimization problem* (OP) is of the form:

$$\begin{array}{rll} \min_{\mathbf{x}\in\mathcal{X}} & f_0(\mathbf{x}) & \\ \text{s.t.} & f_i(\mathbf{x}) \leq 0, & i = 1, \ldots, n \\ & g_j(\mathbf{x}) = 0, & j = 1, \ldots, m. \end{array}$$

where we call:

- f_0 the *objective function.*
- $f_i \quad \forall i = 1 \ldots n$ the *inequality constraints.*
- $g_j \quad \forall j = 1 \ldots m$ the *equality constraints..*

We observe that this special form is not really a restriction. As for a set S, we have that

$$\max_{x\in S} x = -\min_{x\in S} -x.$$

The inequality constraints can be brought into this form by bringing everything to one side and multiplying by -1, if necessary.

Definition 3.2.2 (Convex Optimization Problem)**.** An OP is called convex if and only if the functions $f_0, f_1, \ldots, f_n$ are convex and $g_1, \ldots, g_m$ are linear.

convex optimization problem

▼———————————————

Example 3.3. [Convex Optimization Problem] Recall the hard-margin SVM (2.3.3).

$$\begin{aligned} \max_{\gamma, b \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d \setminus \{0\}} & \gamma \\ \text{s.t.} \quad & y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right) \geq \|\mathbf{w}\| \gamma \quad \forall i = 1 \ldots n. \end{aligned}$$

Let us check if this is a convex OP. According to the definition above, we need to check whether the following is convex for each i:

$$f(\gamma, b, \mathbf{w}) = \|\mathbf{w}\| \gamma \ - y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right).$$

We can show that even in the case of $b = 0$ and $\mathbf{w} > 0 \in \mathbb{R}_+$, the function is not convex. Let us verify this for the simple case of $d = 1$. Then the objective function is just:

$$f(\gamma, w) = w\gamma - y_i(wx_i).$$

Let us calculate the gradient:

$$\nabla f \begin{pmatrix} \gamma \\ w \end{pmatrix} = \begin{pmatrix} w \\ \gamma - y_i x_i \end{pmatrix}.$$

Then the Hessian matrix

$$H_f(\mathbf{x}) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

is not positive semi-definite as $\det(H_f(x)) = -1$, which shows that the hard-margin SVM is not a convex OP.

▲———————————————

3.3 Making the SVM Convex

In the previous example, we saw that the hard-margin SVM is not a convex OP, but, as we will show, we can make it convex for practical purposes.

Theorem 3.3.1. *Assuming the data is linearly separable, then the linear hard-margin SVM, that is,*

$$\max_{\gamma, b \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d \setminus \{0\}} \gamma \ \textit{s.t.} \ y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right) \geq \|\mathbf{w}\| \gamma,$$

can be equivalently rewritten in convex form as given below:

$$\begin{aligned} \min_{\boldsymbol{b} \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d} & \ \tfrac{1}{2} \|\mathbf{w}\|^2 \\ \textit{s.t.} \quad & 1 - y_i \left(\mathbf{w}^\top \mathbf{x}_i + b \right) \leq 0 \quad \forall i = 1, \ldots, n. \end{aligned}$$

Proof. As we assume the data to be linearly separable (that is we can find a hyperplane separating our datapoints), it suffices to find a convex OP for

$$\max_{\gamma>0, b\in\mathbb{R}, \mathbf{w}\in\mathbb{R}^d\setminus\{0\}} \gamma \text{ s.t. } y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq \|\mathbf{w}\|\gamma. \tag{3.3.1}$$

Our SVM will give us the parameters $\mathbf{w}$ and b, which we will use to define the classifier

$$f(\mathbf{x}) = \text{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right).$$

Notice that these parameters are not unique as

$$\forall \lambda > 0 : \text{sign}\left(\mathbf{w}^\top \mathbf{x} + b\right) = \text{sign}(\underbrace{\lambda \mathbf{w}^T}_{=:\mathbf{w}_\lambda}\mathbf{x} + \underbrace{\lambda b}_{=:b_\lambda}). \tag{3.3.2}$$

Now observe that $\max \gamma$ has the same behavior as $\min \frac{1}{2\gamma^2}$. Decreasing/increasing γ has the same effect on both. So we can write (3.3.1) as

$$\min_{\gamma>0, b\in\mathbb{R}, \mathbf{w}\in\mathbb{R}^d\setminus\{0\}} \frac{1}{2\gamma^2} \text{ s.t. } y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq \|\mathbf{w}\|\gamma. \tag{3.3.3}$$

According to (3.3.2), we get that (3.3.3) is equivalent to

$$\min_{\gamma>0, b\in\mathbb{R}, \mathbf{w}\in\mathbb{R}^d\setminus\{0\}} \quad \frac{1}{2\gamma^2} \quad \text{s.t.} \quad y_i\left(\lambda\mathbf{w}^\top \mathbf{x}_i + \lambda b\right) \geq \lambda\|\mathbf{w}\|\gamma. \tag{3.3.4}$$

We can rename λb as b as both are any scalar in $\mathbb{R}$. So (3.3.4) is equivalent to

$$\min_{\gamma>0, b\in\mathbb{R}, \mathbf{w}\in\mathbb{R}^d\setminus\{0\}} \quad \frac{1}{2\gamma^2} \quad \text{s.t.} \quad y_i\left(\lambda\mathbf{w}^\top \mathbf{x}_i + b\right) \geq \lambda\|\mathbf{w}\|\gamma. \tag{3.3.5}$$

Choose $\lambda := \frac{1}{\|\mathbf{w}\|\,\gamma} > 0$ to write (3.3.5) in the form

$$\min_{\gamma>0, b\in\mathbb{R}, \mathbf{w}\in\mathbb{R}^d\setminus\{0\}} \frac{1}{2\gamma^2} \text{ s.t. } y_i\left(\frac{\mathbf{w}^\top \mathbf{x}_i}{\|\mathbf{w}\|\gamma} + b\right) \geq 1. \tag{3.3.6}$$

Substituting $\tilde{\mathbf{w}} := \frac{\mathbf{w}}{\|\mathbf{w}\|\gamma}$ and rewriting $\|\tilde{\mathbf{w}}\| = \left\|\frac{\mathbf{w}}{\|\mathbf{w}\|\gamma}\right\| = \frac{\|\mathbf{w}\|}{\|\mathbf{w}\|\gamma} = 1/\gamma$, we get:

$$\min_{b\in\mathbb{R}, \tilde{\mathbf{w}}\in\mathbb{R}^d\setminus\{0\}} \frac{1}{2}\|\tilde{\mathbf{w}}\|^2 \quad \text{s.t.} \quad y_i\left(\tilde{\mathbf{w}}^\top \mathbf{x}_i + b\right) \geq 1. \tag{3.3.7}$$

Assuming the set of data points contains at least one point from each class ($\exists i, j : (\mathbf{x}_i, 1), (\mathbf{x}_j, -1)$), we can now allow $\tilde{\mathbf{w}} = 0$, since $y_i\left(\tilde{\mathbf{w}}^\top \mathbf{x}_i + b\right) = y_i b \geq 1$ cannot be satisfied for $\tilde{\mathbf{w}} = 0$ because $\mathbf{w} = 0$ will never be in the set we optimize over. Finally, we get (3.3.7) as

$$\min_{b\in\mathbb{R}, \mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad 1 - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \leq 0. \tag{3.3.8}$$

□

It is easy to verify that (3.3.8) is indeed convex: The objective is convex and quadratic, and the constraints are linear.

Theorem 3.3.2. *We can formulate the linear soft-margin SVM, that is,*

$$\begin{aligned} \max_{\gamma,b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d\setminus\{0\},\xi_1,\ldots,\xi_n\geq 0} \quad & \gamma - C\sum_{i=1}^{n}\xi_i \\ s.t.\ \ & y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right) \geq \|\mathbf{w}\|\gamma - \xi_i, \quad \forall i = 1,\ldots,n \\ & \xi_i \geq 0 \qquad\qquad\qquad\qquad\quad \forall i = 1,\ldots,n, \end{aligned}$$

as a convex OP of the form:

$$\begin{aligned} \min_{\mathbf{b}\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d,\boldsymbol{\xi}\in\mathbb{R}^n} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C\textstyle\sum_{i=1}^{n}\xi_i \\ s.t. \quad & 1 - \xi_i - y_i\left(\mathbf{w}^\top\mathbf{x}_i + b\right) \leq 0, \quad -\xi_i \leq 0 \quad \forall i = 1,\ldots,n. \end{aligned} \tag{3.3.9}$$

3.4 Solving Convex Optimization Problems

We saw that we can rewrite our soft-/hard-margin SVM as convex optimization problems. Now we still need to find an algorithm to solve them. We will use the following useful theorem:

Theorem 3.4.1. *Every locally optimal point of a convex optimization problem is also globally optimal.*

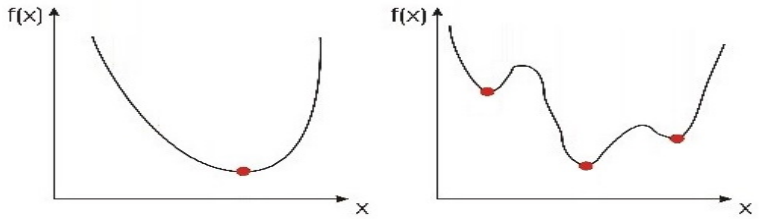

Figure 3.4.1: On the left, we see a convex function, on the right a non-convex function

Apparently it suffices to search only for a local optimum to find a global one in convex functions. The algorithm we will use is *gradient descent*. Its idea is to start at a random point and continuously go in the direction of the steepest descent. The direction of the steepest descent is the negative of the direction of the steepest ascent, which is the direction of the gradient.

Algorithm Gradient Descent

Input: A convex function f, point $\mathbf{x_1}$ (e.g., chosen randomly), T number of iteration steps.

1: **function** GRADDESCENT($f, \mathbf{x_1}$)
2: **for** $t \leftarrow 1$ to T **do**
3: $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \lambda_t \nabla f_0(\mathbf{x}_t)$
4: **end for**
5: **return** $\mathbf{x_T}$
6: **end function**

Definition 3.4.2 (Step Size)**.** We call λ_t the *step size* or *learning rate*.

Observe that finding a good step size is important. If the step size is too small, we need a higher T to reach a global minimum. If the step size is too big, we can overshoot the minimum or the algorithm might even not find a minimum at all. A typical choice would be $\lambda_t := \frac{1}{t}$.

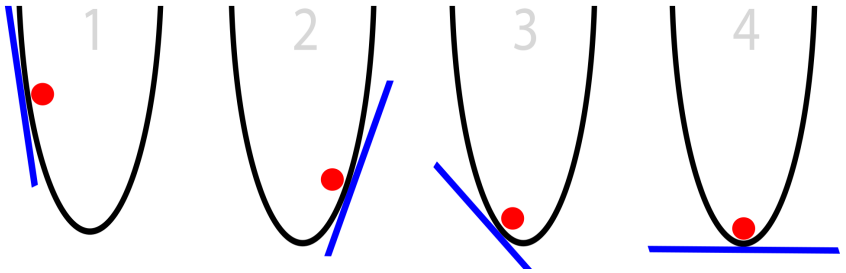

Figure 3.4.2: The behavior of the gradient descent algorithm.

Theorem 3.4.3. *Let $f_0 : \mathbb{R}^d \to \mathbb{R}$ be an arbitrary (possibly non-convex) objective function. Then, under some assumptions, [6] we have:*

- *Gradient descent converges towards a stationary point (maximum, minimum, saddle point), which, for practical purposes, is usually a minimum.*
- *For choosing the ideal learning rate, the convergence rate is at least as good as:*

$$f_0(\mathbf{x}_t) - f_0(\mathbf{x}^*_{local}) \leq O(1/t).$$

[6] The theorem assumes Lipschitz-continuous gradients with a uniformly bounded Lipschitz constant $\exists L : \|\nabla f(\mathbf{x}) - \nabla f(\tilde{\mathbf{x}})\| \leq L\|\mathbf{x} - \tilde{\mathbf{x}}\|$. Convergence is guaranteed for all learning rate schedules satisfying $\sum_{t=1}^{\infty} \lambda_t = \infty$ and λ_t with $t \to \infty$ and $\lambda_t \to 0$, but with varying rates of convergence. The favorable $O(1/t)$ rate is achieved using the minimization rule: $\lambda_t := \arg\min_\lambda f_0(\mathbf{x}_t - \lambda \nabla f_0(\mathbf{x}_t))$. See [1].

3.5 Applying Gradient Descent on the Convex SVM

Let us recall the convex soft-margin SVM

$$\begin{array}{ll} \min\limits_{\mathbf{b}\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d,\boldsymbol{\xi}\in\mathbb{R}^n} & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^n \xi_i \\ \text{s.t.} & 1-\xi_i - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \leq 0, \quad -\xi_i \leq 0 \quad \forall i = 1,\dots n. \end{array}$$

If we look at the gradient descent algorithm, we notice that the input is a convex function. But our OP has one objective function and two inequality constraints for each data point. A quick fix would be to include the two inequality constraints in the objective function. Let us look at the inequality constraints

$$1-\xi_i - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \leq 0 \iff 1 - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \leq \xi_i$$

and

$$-\xi_i \leq 0 \iff 0 \leq \xi_i.$$

Together we get that

$$\max\left(1 - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right), 0\right) \leq \xi_i,$$

which we can safely plug in as an equality into our objective function as we are minimizing, resulting in the following Proposition:

Proposition 3.5.1. *The convex soft-margin SVM can be rewritten as*

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^n \max\left(0, 1 - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right)\right).$$

Our new unconstrained objective becomes

$$f_0(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^n \max\left(0, 1 - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right)\right).$$

Let us look at the part

$$\max\left(1 - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right), 0\right).$$

This part of the function is not differentiable. We can see this by replacing $y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right)$ (a linear growth function) by z, resulting in the so-called *hinge function*

$$\max\left(1 - z, 0\right),$$

which has two tangent lines at $z = 1$, one with slope -1 and the other with slope 0. A potential solution would be to just choose any of the derivatives at this point, so we consider the subgradient:

$$\nabla \max\left(1 - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right), 0\right) := \begin{cases} \nabla(1 - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right)) & \text{if } y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) < 1 \\ 0 & \text{else.} \end{cases}.$$

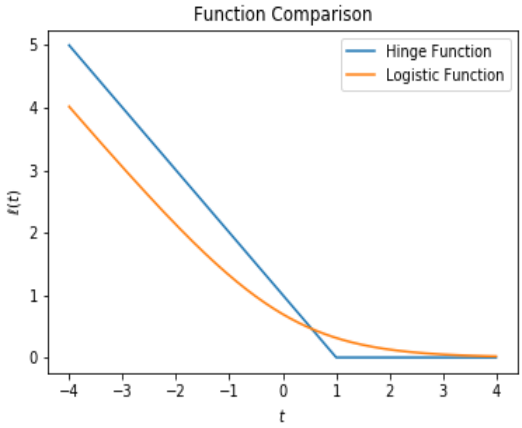

Figure 3.5.1: We observe that the logistic function is a differentiable approximation to the hinge function.

Finally, we use the subgradient descent algorithm, which is the normal gradient descent algorithm, but using the subgradient in place of the gradient. An alternative solution could be to replace this function with something differentiable exhibiting similar behavior. Let us look at the *logistic function*

$$l(z) := \ln(1 + \exp(-z)).$$

logistic regression

Definition 3.5.2 (Logistic Regression)**.** Replacing, in the unconstrained convex soft-margin SVM (3.3.9), the hinge function by the logistic function, we obtain the *logistic regression*:

$$f_0(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ln(1 + \exp(-y_i(\mathbf{w}^\top \mathbf{x_i} + b))). \tag{3.5.1}$$

As the logistic regression is differentiable, we can use the gradient descent algorithm. Let us discuss a final problem. Notice that we need to iterate through all data points to calculate the (sub)gradient. That would mean n iterations, where n can be large for big data. A solution would be to try and approximate the value of the function, leading us to the following algorithm:

Algorithm Stochastic Subgradient Descent (SGD) SVM

Input: A starting point $(\mathbf{w}, b)_1$, batch size B, T number of iteration steps

1: **for** $t \leftarrow 1$ to T **do**
2: Randomly select B data points
3: Denote their indices by $I \subseteq \{1 \dots n\}$
4: $(\mathbf{w}, b)_{t+1} \leftarrow (\mathbf{w}, b)_t - \lambda_t \nabla_{(\mathbf{w},b)} \left(\frac{1}{2}\|\mathbf{w_t}\|^2 + \frac{Cn}{B} \sum_{i \in I} \max\left(0, 1 - y_i\left(\mathbf{w_t}^\top \mathbf{x}_i + b_t\right)\right) \right)$
5: **end for**
6: **return** $(\mathbf{w}, b)_T$

Observation 3.5.3. In SGD, we approximate

$$C \sum_{i=1}^{n} \max\left(0, 1 - y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right)\right)$$

by

$$\frac{Cn}{B} \sum_{i \in I} \max\left(0, 1 - y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right)\right).$$

The scalar $\frac{n}{B}$ is necessary to rescale the value up as we only calculated the value for B points. The value of $B \in [1 \dots N]$ called the *batch size* needs to be chosen beforehand. We can use the exact same algorithm also for logistic regression, by just substituting it in step 4 of the algorithm.

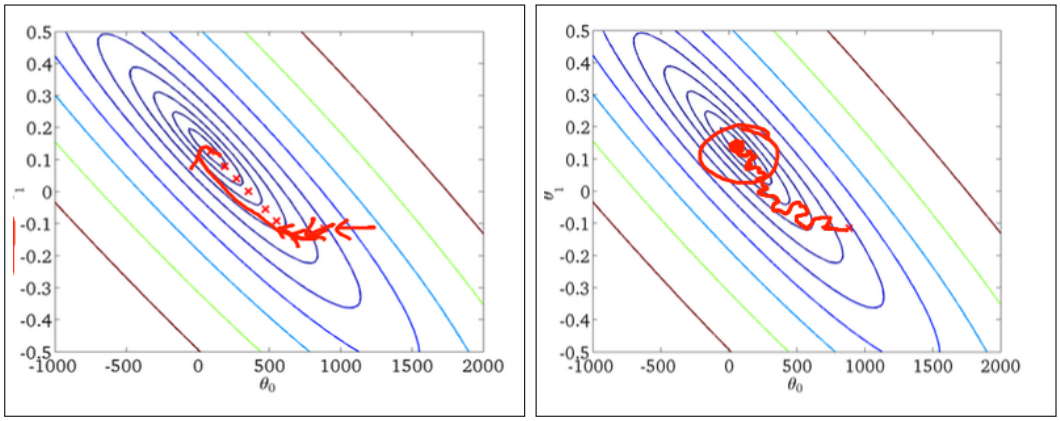

Figure 3.5.2: Visualization of the gradient descent algorithms. To the left, we have the classical gradient descent algorithm and to the right, the stochastic gradient descent. Notice how on the right we are not always perfectly going towards the steepest descent as the gradient is only approximated.

Theorem 3.5.4. *Consider SGD using the learning rate $\lambda_t := 1/t$. Then, under mild assumptions, SGD converges with high probability to a stationary point (see [1]), which is usually a minimum with the rate:*

$$f_0\left(\mathbf{x}_t\right) - f_0\left(\mathbf{x}^*_{local}\right) \leq O(1/t).$$

3.6 Exercises

1. In this exercise, we will try to understand the gradient descent algorithm, its initialization value, and its learning rate schedule. Consider only the functions

$$f : \mathbb{R} \to \mathbb{R}.$$

   a) Find a constant learning rate schedule ($\lambda_i = c$), a convex function f (with a global minimum), and an initialization value x_0 such that the global minimum is never reached if you apply gradient descent with λ_i on f starting at x_0. Prove that the function is convex. Prove that the global minimum is never reached.

b) For your choice of convex function and initialization value, is it possible to choose a learning rate schedule (constant or otherwise) such that the global minimum is reached? Prove your claim.

c) Give an example of an unbounded function ($f(\mathbb{R}) = \mathbb{R}$), a learning rate schedule, and an initialization value for which gradient descent converges to a plateau (a critical point that is neither a minimum nor a maximum). The initialization value cannot be chosen as this plateau.

d) Consider $f(x) = x^3$. Is it possible to find a learning rate schedule that converges to the plateau at $x = 0$ for any initialization value? Prove your claim.

2. Solve Sheet 2.ipynb.

4 Kernel SVM

In the last chapters, we studied how to separate data. We mostly ignored the case where the data is not linearly separable, but we will come back to it now.

4.1 Kernel Methods

Kernel methods are a paradigm for efficiently converting linear learning machines into non-linear ones. At a high level, this works in the following way:

1. Define, in a clever way, a non-linear map

$$\phi : \mathbb{R}^d \to \mathbb{R}^D,$$

   where $\mathbb{R}^D$ is a very high dimensional space ($D >> d$).

2. Map the inputs onto that space,

$$\mathbf{x_i} \mapsto \phi(\mathbf{x_i}), \quad i = 1, \ldots, n.$$

3. Separate the data linearly in that space,

$$f(x) = \text{sign}(\langle \mathbf{w}, \phi(x) \rangle + b).$$

4. This linear separation in the high dimensional space corresponds to a non-linear separation in the input space.

▼———————————————————

Example 4.1. Consider the map:

$$\begin{aligned} \phi :& \mathbb{R}^2 \to \mathbb{R}^3 \\ & (x_1, x_2) \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2). \end{aligned}$$

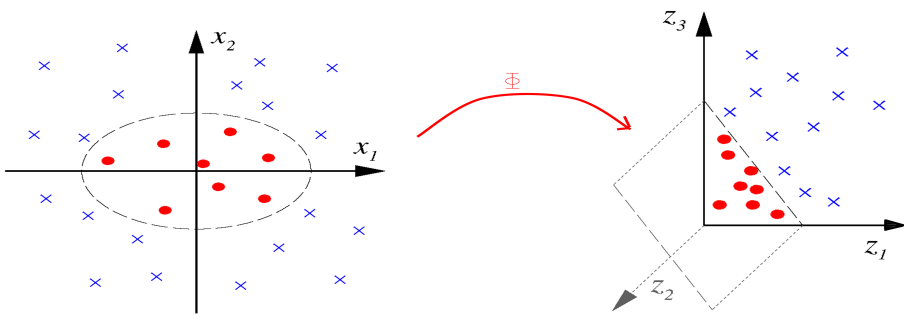

Figure 4.1.1: Visualization of the mapping ϕ.

Observation 4.1.1. Notice how in Figure 4.1.1, a linear separation in $\mathbb{R}^3$ corresponds to a non-linear separation in $\mathbb{R}^2$: For instance, for the hyperplane $\mathbf{w}^T\mathbf{x} + b$ defined by:

$$\mathbf{w} := \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \quad \text{and} \quad b := -1,$$

the corresponding separation for $\mathbf{x} = (x_1, x_2)$ in $\mathbb{R}^2$ would be

$$\mathbf{w}^\top \phi(\mathbf{x}) + b = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}^\top \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} + (-1) = x_1^2 + x_2^2 - 1,$$

which would be the unit circle. The image space is usually too high-dimensional to perform operations such as scalar products in it. In our example, we can do the following: Let $\mathbf{x} = (x_1, x_2)$ and $\tilde{\mathbf{x}} = (\tilde{x_1}, \tilde{x_2})$. Let us try to compute their inner product in the image space:

$$\begin{aligned} \phi(\mathbf{x})^\top \phi(\tilde{\mathbf{x}}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}^\top \begin{pmatrix} \tilde{x_1}^2 \\ \sqrt{2}\tilde{x_1}\tilde{x_2} \\ \tilde{x_2}^2 \end{pmatrix} &= x_1^2\tilde{x_1}^2 + \sqrt{2}\tilde{x_1}\tilde{x_2}\sqrt{2}x_1x_2 + x_2^2\tilde{x_2}^2 \\ &= (x_1\tilde{x_1})^2 + 2(x_1\tilde{x_1}x_2\tilde{x_2}) + (x_2\tilde{x_2})^2 \\ &= (\mathbf{x}^\top\tilde{\mathbf{x}})^2. \end{aligned}$$

Apparently we can compute the scalar product in the image space by only computing scalar products in $\mathbb{R}^2$. This is an example of the kernel trick and $k(\mathbf{x}, \tilde{\mathbf{x}}) := \langle \mathbf{x}, \tilde{\mathbf{x}} \rangle^2$ is an example of a kernel.

▲———————————————————

kernel function

Definition 4.1.2 (Kernel)**.** A function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called a *kernel function* (or simply *kernel*) if all of the following holds:

1. It is symmetric, that is: $\forall \mathbf{x}, \mathbf{x}' : k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$
2. There exists a map $\phi : \mathbb{R}^d \to \mathcal{H}$ called kernel feature map into some high-dimensional kernel feature space $\mathcal{H}$ (e.g., $\mathcal{H} = \mathbb{R}^l$ or $\mathcal{H} = \mathbb{R}^\infty$ [12]) such that:
$$\forall \mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{R}^d : \quad k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle.$$

We can now formalize the *kernel trick*.

1. Formulate the (linear) learning machine (training and prediction) solely in terms of inner products.
2. Replace the inner products $\langle \mathbf{x_i}, \mathbf{x_j} \rangle$ by the kernel $k(\mathbf{x_i}, \mathbf{x_j})$.

Definition 4.1.3 (Linear Kernel)**.** The *linear kernel* defined as

$$k(\mathbf{x}, \tilde{\mathbf{x}}) := \langle \mathbf{x}, \tilde{\mathbf{x}} \rangle.$$

is a kernel. **linear kernel**

Proof. Choose the identity map $\phi = \mathrm{id}$. Then for all $\mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{R}^d$ we have:

$$k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \mathbf{x}, \tilde{\mathbf{x}} \rangle = \langle \mathrm{id}(\mathbf{x}), \mathrm{id}(\tilde{\mathbf{x}}) \rangle = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle.$$

Symmetry follows by the symmetry of the scalar product. □

Definition 4.1.4 (Polynomial Kernel)**.** The kernel called *polynomial kernel* of degree $m \in \mathbb{N}$ is defined as **polynomial kernel**

$$k(\mathbf{x}, \tilde{\mathbf{x}}) := (\langle \mathbf{x}, \tilde{\mathbf{x}} \rangle + c)^m$$

where $c \geq 0$ is a parameter.

We finally define one of the most important kernels in practice.

Definition 4.1.5 (Gaussian RBF Kernel)**.** The *Gaussian RBF kernel* is a kernel defined as **Gaussian RBF kernel**

$$k(\mathbf{x}, \tilde{\mathbf{x}}) := \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \tilde{\mathbf{x}}\|^2\right).$$

We call the parameter $\sigma^2 > 0$ the kernel width (or bandwith).

Definition 4.1.6 (Kernel Matrix)**.** Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ be the input data, and let $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a kernel function. Then the matrix

$$K := \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \ldots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \ldots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \in \mathbb{R}^{n \times n}$$

is called a kernel matrix.

Theorem 4.1.7 (Closure Properties)**.** *The following properties hold for given functions k, k_1 and k_2.*

1. *If k is a kernel and $c \in \mathbb{R}_+$, then ck is a kernel.*
2. *If k_1 and k_2 are kernels, then $k_1 + k_2$ is a kernel.*
3. *If k_1 and k_2 are kernels, then $k_1 \cdot k_2$ is a kernel*

Theorem 4.1.8. *A function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathcal{H}$ is a kernel if and only if for any $n \in \mathbb{N}$ and any input points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, the matrix K is positive semi-definite (meaning $\forall \mathbf{v} \in \mathbb{R}^n : \mathbf{v}^\top K \mathbf{v} \geq 0$).*

As seen in the following, Theorem 4.1.8 is quite useful for proving that functions are kernels.

▼———————————————

Example 4.2. [Closure of Kernels under Addition] Let $\mathbf{x_1}, \ldots, \mathbf{x_n} \in \mathbb{R}^d$ be arbitrary data points, k_1, k_2 be two kernels, and K_1, K_2 be the corresponding kernel matrices. Define $k := k_1 + k_2$. The kernel matrix K of k will look as follows:

$$\begin{aligned} K =& \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \ldots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \ldots & k(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \\ =& \begin{pmatrix} k_1(\mathbf{x}_1, \mathbf{x}_1) + k_2(\mathbf{x}_1, \mathbf{x}_1) & \ldots & k_1(\mathbf{x}_1, \mathbf{x}_n) + k_2(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & & \vdots \\ k_1(\mathbf{x}_n, \mathbf{x}_1) + k_2(\mathbf{x}_n, \mathbf{x}_1) & \ldots & k_1(\mathbf{x}_n, \mathbf{x}_n) + k_2(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} \\ =& K_1 + K_2. \end{aligned}$$

We know K_1, K_2 are positive semi-definite. We now prove that K is positive semi-definite. It then follows by Theorem 4.1.8 that k is a kernel. Let $\mathbf{v} \in \mathbb{R}^d$. We have:

$$\mathbf{v}^\top K \mathbf{v} = \mathbf{v}^\top (K_1 + K_2)\mathbf{v} = \mathbf{v}^\top (K_1 \mathbf{v} + K_2 \mathbf{v}) = \mathbf{v}^\top K_1 \mathbf{v} + \mathbf{v}^\top K_2 \mathbf{v} \geq 0.$$

Thus K is positive semi-definite and k is a kernel.

▲———————————————

4.2 Applying the Kernel Trick to the SVM

Let us recall the unconstrained version of our soft-margin SVM (3.3.9).

$$\min_{b \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right)\right).$$

Let $\mathbf{w}^*$ be the optimal solution. We use the fact that $\mathbf{w}^* \in \operatorname{span}(\{\mathbf{x_1}, \ldots \mathbf{x_n}\})$, which is a special case of the general representer theorem, which we will prove later on. So there exists $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_n) \in \mathbb{R}^n$ such that

$$\mathbf{w}^* = \sum_{j=1}^{n} \alpha_j \phi(\mathbf{x_j}). \tag{4.2.1}$$

So we can also find the optimal solution by optimizing over the possible $\boldsymbol{\alpha}$. By plugging (4.2.1) into (3.3.9), we get

$$\min_{b \in \mathbb{R}, \boldsymbol{\alpha} \in \mathbb{R}^n} \frac{1}{2}\|\sum_{j=1}^{n} \alpha_j \phi(\mathbf{x_j})\|^2 + C \sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\left(\sum_{j=1}^{n} \alpha_j \phi(\mathbf{x_j})\right)^\top \phi(\mathbf{x}_i) + b\right)\right). \tag{4.2.2}$$

By inserting the definition of the norm and then multiplying out, we equivalently get

$$\min_{b\in\mathbb{R},\boldsymbol{\alpha}\in\mathbb{R}^n} \frac{1}{2}\left(\sum_{i=1}^{n}\alpha_i\phi(\mathbf{x_i})\right)^\top\left(\sum_{j=1}^{n}\alpha_j\phi(\mathbf{x_j})\right)+C\sum_{i=1}^{n}\max\left(0,1-y_i\left(\sum_{j=1}^{n}\alpha_j\phi(\mathbf{x_j})^\top\phi(\mathbf{x}_i)+b\right)\right), \tag{4.2.3}$$

which, by multiplying out and reordering in the last sum, is equivalent to

$$\min_{b\in\mathbb{R},\boldsymbol{\alpha}\in\mathbb{R}^n} \frac{1}{2}\left(\sum_{i,j=1}^{n}\alpha_i\alpha_j\phi(\mathbf{x_i})^\top\phi(\mathbf{x_j})\right)+C\sum_{i=1}^{n}\max\left(0,1-y_i\left(\sum_{j=1}^{n}\alpha_j\phi(\mathbf{x_i})^\top\phi(\mathbf{x}_j)+b\right)\right). \tag{4.2.4}$$

As the SVM is used in the higher-dimensional space, we want to make use of our kernel function. So we replace all occurrences of $\phi(\mathbf{x_i})^\top\phi(\mathbf{x}_j)$ by $k(\mathbf{x}_i,\mathbf{x}_j)$ to finally get:

$$\min_{b\in\mathbb{R},\boldsymbol{\alpha}\in\mathbb{R}^n} \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j)+C\sum_{i=1}^{n}\max\left(0,1-y_i\left(\sum_{j=1}^{n}\alpha_j k(\mathbf{x}_i,\mathbf{x}_j)+b\right)\right). \tag{4.2.5}$$

We finally get the prediction function:

$$\begin{aligned} f(\mathbf{x}) =& \operatorname{sign}\left(\mathbf{w}^{*\top}\phi(\mathbf{x})+b\right)\\ =& \operatorname{sign}\left(\left(\sum_{j=1}^{n}\alpha_j\phi(\mathbf{x_j})\right)^\top\phi(\mathbf{x})+b\right)\\ =& \operatorname{sign}\left(\sum_{j=1}^{n}\alpha_j k(\mathbf{x},\mathbf{x}_j)+b\right). \end{aligned}$$

Notice that we actually do not need the feature map ϕ. Now that we have the training and prediction algorithm, we still need to solve the optimization problem (4.2.5). For this, we use a similar approach to the stochastic gradient method from Chapter 3.

Algorithm Doubly SGD algorithm for Kernel SVM

Input: A starting point $(b.\boldsymbol{\alpha})_1$, batch size B, T number of iteration steps.

1: **for** $t \leftarrow 1$ to T **do**
2: Randomly select B data points
3: Denote their indexes by $J \subset \{1,\dots,n\}$
4: $(b,\boldsymbol{\alpha})_{t+1} := (b,\boldsymbol{\alpha})_t - \lambda_t\nabla_{(b,\boldsymbol{\alpha}}\left(\frac{n^2}{2B^2}\sum_{i,j\in J}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j)\right.$
$\left.+\frac{Cn}{B}\sum_{i\in J}\max\left(0,1-y_i\left(\frac{n}{B}\sum_{j\in J}\alpha_j k(\mathbf{x}_i,\mathbf{x}_j)+b\right)\right)\right)$
5: **end for**
6: **return** $(b,\boldsymbol{\alpha})_T$

Observation 4.2.1. This is basically the same algorithm as SGD. Notice that we approximate

$$\sum_{i,j=1}^{n} \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right)$$

by

$$\frac{n^2}{B^2} \sum_{i,j \in J} \alpha_i \alpha_j k\left(\mathbf{x}_i, \mathbf{x}_j\right),$$

as we only calculated the sum for B^2 items, so we need to scale it up. Similar approximations are done in the rest of the algorithm. Just like in SGD, we are giving up a bit of precision to be faster.

In the construction of the Kernel SVM, we used the fact that $\mathbf{w}^* \in \operatorname{span}(\{\mathbf{x_1}, \ldots \mathbf{x_n}\})$. We will now prove this using a more general statement.

Theorem 4.2.2 (General Representer Theorem)**.** *Let $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a kernel over a Hilbert space $\mathcal{H}$. Let $L : \mathbb{R}^n \to \mathbb{R}$ be any function. Then any solution*

$$\mathbf{w}^* \in \arg\min_{\mathbf{w} \in \mathcal{H}} \frac{1}{2}\|\mathbf{w}\|^2 + L\left(\left\langle \mathbf{w}, \phi\left(\mathbf{x}_1\right)\right\rangle, \ldots, \left\langle \mathbf{w}, \phi\left(\mathbf{x}_n\right)\right\rangle\right)$$

satisfies:

$$\text{there } \exists \alpha_1, \ldots, \alpha_n \text{ such that } \mathbf{w}^* = \sum_{i=1}^{n} \alpha_i \phi\left(\mathbf{x}_i\right).$$

Proof. Let $\mathcal{H}$ be a Hilbert space. Consider now

$$V := \operatorname{span}(\{\phi(\mathbf{x_1}), \ldots, \phi(\mathbf{x_n}\}),$$

which is clearly a finitely dimensional subspace and therefore closed. Denote by

$$U := V_\perp = \{\mathbf{u} \in H \mid \forall \mathbf{v} \in V : \langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{0}\},$$

the orthogonal complement of V. According to the orthogonal decomposition theorem of Hilbert spaces [12], we can write every vector $\mathbf{z} \in \mathcal{H}$ in a unique form:

$$\mathbf{w} = \underbrace{\mathbf{v}}_{\in V} + \underbrace{\mathbf{u}}_{\in U}.$$

Consider now that there exists an optimal solution $\mathbf{w}^*$ with $\mathbf{v}^* \in V$ and $\mathbf{u}^* \in U$ of the form:

$$\mathbf{w}^* = \mathbf{v}^* + \underbrace{\mathbf{u}^*}_{\neq 0}.$$

For the loss function, we observe the following $\forall i = 1, \ldots, n$:

$$\begin{aligned} \left\langle \mathbf{w}^*, \phi\left(\mathbf{x}_i\right)\right\rangle &= \left\langle \mathbf{v}^* + \mathbf{u}^*, \phi\left(\mathbf{x}_i\right)\right\rangle \\ &= \left\langle \mathbf{v}^*, \phi\left(\mathbf{x}_i\right)\right\rangle + \underbrace{\left\langle \mathbf{u}^*, \phi\left(\mathbf{x}_i\right)\right\rangle}_{=0} \\ &= \left\langle \mathbf{v}^*, \phi\left(\mathbf{x}_i\right)\right\rangle. \end{aligned}$$

So whether we choose $\mathbf{w}^*$ or $\mathbf{v}^*$ does not influence the values of the loss function, but it does influence the value of the regularizer as we have:

$$\|\mathbf{w}^*\|^2 = \|\mathbf{v}^*\|^2 + \underbrace{\|\mathbf{u}^*\|^2}_{>0} > \|\mathbf{v}^*\|^2 .$$

This clearly contradicts the optimality of $\mathbf{w}^*$ as we found the better solution $\mathbf{v}^* \in V$. □

4.3 Exercises

1. Suppose that $k_1, \ldots, k_n : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ are kernels. Let $c_1, \ldots, c_n \in \mathbb{R}^+$ and $p \in \mathbb{N}$. Prove that the following functions k are also kernels.
   a) **Scaling**: $k(\mathbf{x}, \mathbf{x}') := c_1 k_1(\mathbf{x}, \mathbf{x}')$
   b) **Linear combination**: $k(\mathbf{x}, \mathbf{x}') := \sum_{i=1}^n c_i k_i(\mathbf{x}, \mathbf{x}')$
   c) **Product**: $k(\mathbf{x}, \mathbf{x}') := k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}')$
   d) **Power**: $k(\mathbf{x}, \mathbf{x}') := k_1(\mathbf{x}, \mathbf{x}')^p$.
2. Prove the following statements:
   a) **Polynomial kernel**: $k(\mathbf{x}, \mathbf{x}') := (\mathbf{x}^T \mathbf{x}' + c)^d$ is a kernel.
   b) **Limits**: If $k_i : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, $i \in \mathbb{N}$ are kernels and $k(\mathbf{x}, \mathbf{x}') := \lim_{n\to\infty} k_n(\mathbf{x}, \mathbf{x}')$ exists for all $\mathbf{x}, \mathbf{x}'$, then $k(\mathbf{x}, \mathbf{x}')$ is a kernel. Use the definition of positive semi-definiteness.
   c) **Exponents**: If $\tilde{k}$ is a kernel, then $k(\mathbf{x}, \mathbf{x}') := \exp(\tilde{k}(\mathbf{x}, \mathbf{x}'))$ is a kernel.
   d) **Functions**: If $\tilde{k}$ is a kernel and $f : \mathbb{R}^d \to \mathbb{R}$ then $k(\mathbf{x}, \mathbf{x}') := f(\mathbf{x})\tilde{k}(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$ is a kernel.
   e) **Gaussian RBF kernel**: $k(\mathbf{x}, \mathbf{x}') := \exp(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2})$ is a kernel.

   Hint: Use the results from Exercise 1 above.
3. Let $k(\cdot, \cdot)$ be a kernel on $\mathbb{R}^d$. Let $\phi(\cdot)$ be the kernel mapping, i.e., $\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = k(\mathbf{x}, \mathbf{y})$. Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, $a = [a_1, \ldots, a_n]^T \in \mathbb{R}^n$ and $b = [b_1, \ldots, b_n]^T \in \mathbb{R}^n$. Let $K \in \mathbb{R}^{n\times n} = [k(\mathbf{x}_i, \mathbf{x}_j)]_{i,j}$ be the kernel matrix. Prove that

$$\left\langle \sum_{i=1}^n a_i \phi(\mathbf{x}_i), \sum_{j=1}^n b_j \phi(\mathbf{x}_j) \right\rangle = a^T K b$$

4. Solve Sheet 3.ipynb.

5 Neural Networks

In the previous chapters, we saw the advantages of linear and kernel learning methods. Kernel methods work well in separating non-linearly separable data, assuming we have a good feature map ϕ. We will now turn to neural networks, whose main advantage is that they can also learn this feature map. As motivation, let us look at logistic regression in a high-dimensional space:

$$\min_{b\in\mathbb{R},\boldsymbol{w}\in\mathbb{R}^d,\phi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ln\left(1+\exp\left(-y_i\left(\mathbf{w}^\top\phi\left(\mathbf{x}_i\right)+b\right)\right)\right). \tag{5.0.1}$$

To find this minimum, we also optimize over possible mappings ϕ. The main problem is that there are too many mappings ϕ to consider. As motivation, we can first look at how our brain does this.

5.1 The Brain Graph

In simple terms, we can consider the brain to be a graph. We call a node *neuron* and an edge *synapse*. Another word for graph is network. As the nodes are called neurons, we call the brain graph a *neural network*. Let us, in

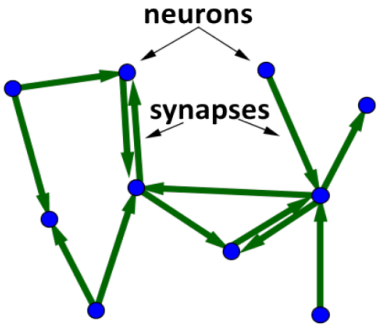

Figure 5.1.1: Visualization of the brain graph.

very simple terms, look at what the brain does when it sees something.

1. Some neurons will light up (are activated).
2. An activated neuron i will shoot an electrical signal v (*spike*) to neuron j with strength proportional to W_{ij} (the weight of the synapse).
3. If the in-going electrical signals of neuron j (called the *potential* of j) exceed a threshold, it will also become activated and can thus shoot electrical signals v to its neighboring neurons. See Fig. 5.1.2.

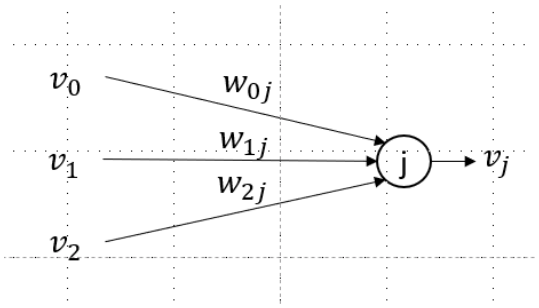

Figure 5.1.2: Illustration of the process explained above. The spike in v_j of neuron j is dependent on the spikes v_0, v_1, v_2 and the weights w_{0j}, w_{1j}, w_{2j}.

McCulloch and Pitts model

Denote by u the neuron's potential and by v the emitted spike. Then:

$$v = \sigma(u),$$

where σ is the *sigmoid function*: $\sigma(u) = \frac{1}{1+e^{-u}}$. Today's artificial neural

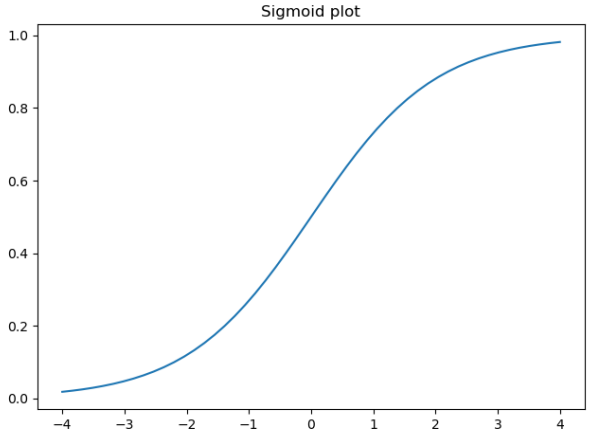

Figure 5.1.3: The sigmoid function fits the values u between 0 and 1.

ReLU activation function

networks use the *ReLU activation* function instead of the sigmoid function

$$\sigma(u) := \max(0, u).$$

We will use the ReLU activation function in the following, but generally, it does not matter which type of activation function we use.

5.2 Artifical Neural Networks (ANN)

Let us formalize the propagation of neuron activation. Neuron j is connected with strength W_{ij} to other neurons who emit spike v_j. The potential is

$$u_j = \sum_i w_{ij} v_i,$$

and its activation

$$v_j = \sigma(u_j) = \sigma\left(\sum_i w_{ij} v_i\right).$$

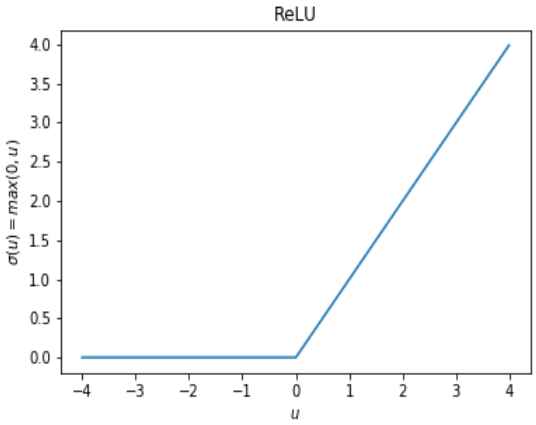

Figure 5.1.4: The ReLU activation function, which is commonly used in practice.

Let W be the adjacency matrix of the neural network, $\mathbf{u} = (u_1, \ldots, u_d)^\top$ and $\mathbf{v} = \sigma(\mathbf{u})$ be the activation of all neurons as a vector. Notice that by transposing W, the potential can be calculated as

$$\mathbf{u} = W^\top \mathbf{v}$$

and the activation as

$$\mathbf{v} = \sigma(\mathbf{u}) = \sigma\left(W^\top \mathbf{v}\right).$$

Notice that our equation is dependent on $\mathbf{v}$ on the left and right side. To not allow a cyclic dependency, we restrict our neural network to being an acyclic graph.

5.3 Feed-Forward ANN

The acyclic construction from above is called *feed-forward ANN* or *multi-layer perceptron*.

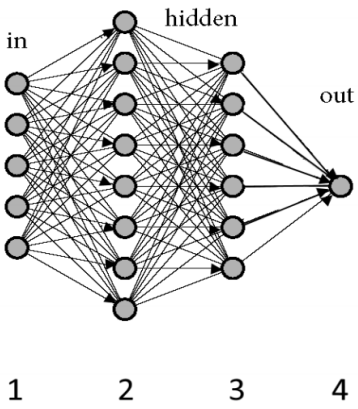

Figure 5.3.1: Example of a feed forward ANN. Have a look at `https://playground.tensorflow.org/` to get a better understanding of ANN and play around with small networks.

We use Fig. 5.3.1 to define some notions. Shown in Fig. 5.3.1, we have a feed-forward network with an input layer (1) of 5 nodes, two hidden layers (2,3) of 8 and 6 nodes, and an output layer (4) with 1 node. Let $\mathbf{v_l}$ be the vector of activation of neurons in the l-th layer and W_l be the submatrix of the strengths of the connections between the $l-1$-th and the l-th layer. Recall that

$$\mathbf{v} = \sigma(\mathbf{u}) = \sigma\left(W^\top \mathbf{v}\right).$$

In a feed-forward network, we can calculate v_{l+1} by

$$\mathbf{v}_{l+1} = \sigma(\underbrace{W_{l+1}^\top \mathbf{v}_l}_{=\mathbf{u}_{l+1}}).$$

Let us use this observation. Let 0 and $\mathbf{x} := \mathbf{v_0}$ be our starting layer and L be the number of hidden layers sorted from left to right with $L+1$ being the output layer. We can calculate $\mathbf{v_l}$ as:

$$\phi_W(\mathbf{x}) := \mathbf{v_l} = \sigma\underbrace{\left(\underbrace{\underbrace{W_L^\top \sigma(\dots \sigma(W_1^\top \underbrace{\mathbf{x}}_{=\mathbf{v}_0})\dots)}_{=\mathbf{u}_1}}_{=\mathbf{v1}}\right)}_{=\mathbf{v}_l},$$

and the potential of the nodes in the last layer as

$$\mathbf{u_{l+1}} = W_{L+1}^\top \phi_w(\mathbf{x}).$$

So the feature map we just created will have as output the potential u_{L+1}, which is a scalar if we have one output node. To wrap it all up, we will get the following optimization problem, which was motivated by (5.0.1):

$$\min_{\mathbf{w},W} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2 + C\sum_{i=1}^{n}\log\left(1+\exp\left(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)\right), \quad (5.3.1)$$

where

- $W := (W_1, \dots, W_L)$,
- $\mathbf{w} = W_{L+1}$,
- $\|W_k\|_{\text{Fro}}^2 = \sum_{ij} w_{kij}^2$,
- $\phi_W\left(\mathbf{x}_i\right) := \sigma\left(W_L^\top \sigma\left(\dots \sigma\left(W_1^\top \mathbf{x}_i\right)\dots\right)\right)$,

which is just logistic regression with the feature map ϕ_W and the regularizer

$$\frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2,$$

which is a generalization of a vector norm to a matrix norm. Observe that the hyperplane bias parameter b is left out. We can do this as, if needed, the machine can put the bias inside the feature map ϕ_W.

5.4 Training ANN

Now we turn to solving the optimization problem (5.3.1). Our approach will again be that of stochastic gradient descent. We first need to compute some gradients. For the sake of simplicity, let us call

$$F(\mathbf{w}, W) := \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2 + C\sum_{i=1}^{n}\log\left(1+\exp\left(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)\right).$$

Let us first compute the gradient with respect to $\mathbf{w}$. By linearity, we have:

$$\nabla_{\mathbf{w}}F(\mathbf{w}, W) = \underbrace{\nabla_{\mathbf{w}}\left(\frac{1}{2}\|\mathbf{w}\|^2\right)}_{=\mathbf{w}} + \underbrace{\nabla_{\mathbf{w}}\left(\frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2\right)}_{=0} + C\sum_{i=1}^{n}\nabla_{\mathbf{w}}\left(\log\left(1+\exp\left(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)\right)\right).$$

If we denote

$$g(x) := \log(1+\exp(x)) \text{ with } g'(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)}$$

and

$$f_i(\mathbf{w}) = -y_i\mathbf{w}^T\phi_W(\mathbf{x_i}) \text{ with } \nabla_{\mathbf{w}}f_i(\mathbf{w}) = -y_i\phi_W(\mathbf{x_i}),$$

we get:

$$\begin{aligned}\nabla_{\mathbf{w}}\left(\log\left(1+\exp\left(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)\right)\right) &= \nabla_{\mathbf{w}}\left(g(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right))\right)\\ &= \nabla_{\mathbf{w}}(g(f_i(\mathbf{w}))\\ &= \nabla g\left(f_i\left(\mathbf{w}\right)\right)\nabla f_i\left(\mathbf{w}\right)\\ &= \frac{1}{1+\exp(y_i\mathbf{w}^T\phi_W(\mathbf{x_i}))}\cdot -y_i\phi_W(\mathbf{x_i}).\end{aligned}$$

In conclusion, we get

$$\nabla_{\mathbf{w}}F(\mathbf{w}, W) = \mathbf{w} - C\sum_{i=1}^{n}\frac{y_i\phi_W\left(\mathbf{x}_i\right)}{1+\exp\left(y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)}.$$

Now we still need to find the gradient with respect to $W = (W_1, \ldots, W_L)$. Let us take the gradient with respect to each W_l. Again by linearity, we have:

$$\begin{aligned}\nabla_{W_l}F(\mathbf{w}, W) = &\underbrace{\nabla_{W_l}\left(\frac{1}{2}\left\|\mathbf{w}\right\|^2\right)}_{=0} + \underbrace{\nabla_{W_l}\left(\frac{1}{2}\sum_{t=1}^{L}\|W_t\|_{\text{Fro}}^2\right)}_{=W_l}\\ &+ C\sum_{i=1}^{n}\nabla_{W_l}\left(\log\left(1+\exp\left(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)\right)\right)\\ &= W_l - C\sum_{i=1}^{n}\frac{y_i\mathbf{w}^\top\nabla_{W_l}\phi_W\left(\mathbf{x}_i\right)}{1+\exp\left(y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)}.\end{aligned}$$

We still need to calculate

$$\nabla_{W_l}\phi_W\left(\mathbf{x}_i\right).$$

For a better overview, we calculate $\nabla_{W_{ijl}}\phi_W(\mathbf{x_i})$ with W_{ijl} being the i,j-th entry of the matrix W_l According to the definition of $\phi_W(\mathbf{x})$, we have:

$$\nabla_{W_{ijl}}\phi_W(\mathbf{x}) := \nabla_{W_{ijl}}\, \underbrace{\sigma\left(\begin{array}{c} W_L^\top \sigma(\dots \sigma(W_1^\top \underbrace{\mathbf{x}}_{=\mathbf{v}_0}) \dots) \\ \underbrace{\qquad\qquad}_{=\mathbf{u}_1} \\ \underbrace{\qquad\qquad\qquad}_{=\mathbf{v_1}} \end{array}\right)}_{=\mathbf{v}_l}.$$

We apparently need the derivative of a nested function, for which we will use the chain rule, going up to $\mathbf{u}_l$ for the matrix W_l:

$$\nabla_{w_{ijl}}\phi_W(\mathbf{x}) = \frac{\partial \mathbf{v}_L}{\partial w_{ijl}} = \frac{\partial \mathbf{v}_L}{\partial \mathbf{u}_L} \cdot \frac{\partial \mathbf{u}_L}{\partial \mathbf{v}_{L-1}} \cdot \frac{\partial \mathbf{v}_{L-1}}{\partial \mathbf{u}_{L-1}} \cdots \frac{\partial \mathbf{u}_{l+1}}{\partial \mathbf{v}_l} \cdot \frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l} \cdot \frac{\partial \mathbf{u}_l}{\partial w_{ijl}}.$$

So we need to calculate three derivatives:

- $\frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l}$
- $\frac{\partial \mathbf{u}_l}{\partial \mathbf{v}_{l-1}}$
- $\frac{\partial \mathbf{u}_l}{\partial w_{ijl}}$.

Let us first define the following auxiliary function

$$\Theta : \mathbb{R} \to \mathbb{R}$$

$$\Theta(c) := \begin{cases} 0 & c \leq 0 \\ 1 & \text{otherwise} \end{cases},$$

where Θ of a vector is applied element-wise:

$$\Theta(\mathbf{x}) := \begin{pmatrix} \Theta(x_1) \\ \vdots \\ \Theta(x_d) \end{pmatrix}.$$

Now, we go back to calculating the sought derivatives. For the activation function, we use the ReLU $\sigma(\mathbf{v}) = \max(0, \mathbf{u})$. We have:

$$\frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l} = \frac{\partial \sigma(\mathbf{u}_l)}{\partial \mathbf{u}_l} = \frac{\partial \max(0, \mathbf{u}_l)}{\partial \mathbf{u}_l} = \Theta(\mathbf{u}_l).$$

Moving on to the next derivative, we have:

$$\frac{\partial \mathbf{u}_l}{\partial \mathbf{v}_{l-1}} = \frac{\partial \left(W_l^\top \mathbf{v}_{l-1}\right)}{\partial \mathbf{v}_{l-1}} = W_l^\top.$$

Let us briefly recall the column-wise matrix multiplication

$$A\mathbf{x} = A_{\cdot,1}x_1 + A_{\cdot,2}x_2 + \cdots + A_{\cdot,n}x_n,$$

where $A_{\cdot,i}$ stands for the i-th column in the matrix A. We will use this to calculate our final derivative. Noticing that transposition interchanges indices, we get:

$$\begin{aligned}\frac{\partial \mathbf{u}_l}{\partial W_{ijl}} &= \frac{\partial \left(W_l^\top \mathbf{v}_{l-1}\right)}{\partial W_{ijl}} = \frac{\partial \left(W_{\cdot,1,l}^\top \mathbf{v}_{1,l-1} + W_{\cdot,2,l}^\top \mathbf{v}_{2,l-1} + \cdots + W_{\cdot,n}^\top \mathbf{v}_{n,l-1}\right)}{\partial W_{ijl}} \\ &= \frac{\partial \left(W_{\cdot,1,l}^\top \mathbf{v}_{1,l-1}\right)}{\partial W_{ijl}} + \frac{\partial \left(W_{\cdot,2,l}^\top \mathbf{v}_{2,l-1}\right)}{\partial W_{ijl}} + \cdots + \frac{\partial \left(W_{\cdot,n,l}^\top \mathbf{v}_{n,l-1}\right)}{\partial W_{ijl}} \\ &= 0 + 0 + \cdots + \frac{\partial \left(W_{\cdot,i,l}^\top \mathbf{v}_{i,l-1}\right)}{\partial W_{ijl}} + 0 + \cdots + 0 \\ &= \mathbf{v}_{i,l-1}\mathbf{e}_j.\end{aligned}$$

We can now look back to the chain-rule formulation of $\nabla_{w_{ijl}}\phi_W(\mathbf{x})$. With our calculations so far, it translates to

$$\begin{aligned}\nabla_{w_{ijl}}\phi_W(\mathbf{x}) &= \frac{\partial \mathbf{v}_L}{\partial \mathbf{u}_L} \cdot \frac{\partial \mathbf{u}_L}{\partial \mathbf{v}_{L-1}} \cdot \frac{\partial \mathbf{v}_{L-1}}{\partial \mathbf{u}_{L-1}} \cdots \frac{\partial \mathbf{u}_{l+1}}{\partial \mathbf{v}_l} \cdot \frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l} \cdot \frac{\partial \mathbf{u}_l}{\partial w_{ijl}} \\ &= \Theta\left(\mathbf{u}_L\right) W_L^\top \Theta\left(\mathbf{u}_{L-1}\right) \cdots W_{l+1}^\top \Theta\left(\mathbf{u}_l\right) v_{i,l-1}\mathbf{e}_j.\end{aligned}$$

Notice that we need to compute each $\mathbf{u}_l$ to use $\Theta(\mathbf{u}_l)$. We can achieve this with the following algorithm:

Algorithm Forward Propagation

1: Initialize $\mathbf{v_0} = \mathbf{x}$
2: **for** $l \leftarrow 1$ to $(L-1)$ **do**
3: $\quad \mathbf{u}_l := W_l^\top \mathbf{v}_{l-1}$
4: $\quad \mathbf{v}_l := \sigma\left(\mathbf{u}_l\right)$
5: **end for**

Now, we can finally compute our gradient $\nabla_{w_{ijl}}\phi_W(\mathbf{x})$ for all i, j, l with the following algorithm.
The algorithm just calculates our chain-rule formulation.

Algorithm Back Propagation

1: Initialize $\boldsymbol{\delta_L} := \Theta\left(\mathbf{u}_L\right)$
2: $\forall i, j : \nabla_{w_{ijL}}\phi_W(\mathbf{x}) := \boldsymbol{\delta_L} v_{i,L-1}\mathbf{e}_j$
3: **for** $l \leftarrow (L-1)$ to 1 **do**
4: $\quad \boldsymbol{\delta}_l := \boldsymbol{\delta}_{l+1} W_{l+1}^\top \Theta\left(\mathbf{u}_l\right)$
5: $\quad \forall i, j : \nabla_{w_{ijl}}\phi_W(\mathbf{x}) := \boldsymbol{\delta_l}\mathbf{v_{i,l-1}}\mathbf{e}_j$
6: **end for**

5.5 Convolutional Neural Networks (CNN)

CNN are a type of artificial neural network. They are mostly used in image and speech recognition. In the following, we will use the example of the picture of an apple for image recognition. In image processing, it makes sense to

process portions of an image (called *patches*). An apple could be anywhere in the image, and we need to make sure to identify it regardless of its position. To focus on 'interesting parts' of the image (e.g., the apple stalk), we apply so-called *filters*. The high-level idea is to identify parts of an object and then use their position in relation to each other to determine whether the object is present. However, in practice, it is difficult to affirm whether CNN actually follow this basic idea.

CNN incorporate spatial information and weight sharing. Weight sharing is the idea of having multiple weights being equal. To understand spatial information, consider an image of an apple. Now randomly permute the pixels of this image. The apple will no longer be recognizable, even though the set of pixel values is still the same. In images, pixels close to each other have more interaction than pixels far appart, which is what we try to capture in a network. A *convolutional filter* is a filter based on convolutions. Examples include the Gaussian filter (blurring) and the Laplace filter (edge detection). You can think of the filters as being the weights of the edges in a neural network. The key point of CNN is to learn the convolutional filter that helps us recognize important parts of the image. Notice that a picture can have many pixels and thus a CNN can get quite large, so we need techniques to reduce their size. *Pooling* is a common choice to reduce a CNN's size. Pooling shrinks many neighboring pixels into one. The typical pooling method is max pooling. However, other pooling methods are also used in specific settings. In the following, we will give two examples.

- *Max Pooling:* The newly formed pixel is the maximum of all pixel values in the patch.
- *Average Pooling:* The newly formed pixel is the average of all pixel values in the patch.

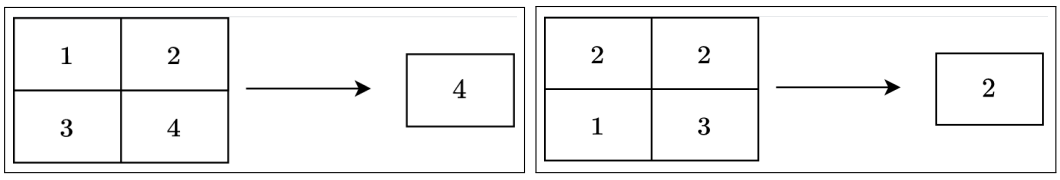

Figure 5.5.1: Examples of pooling: To the left, we see max pooling, to the right average pooling.

5.6 CNN Architecture

Next, we will explain how a CNN layer is set up. First we consider an input, which has a *width*, *height*, and *depth*. The depth is usually referred to as the channel dimension. For typical images, this dimension contains different color channels, such as RGB channels. We will denote width by w, height by h, and the channels by c. Then the input I is in $\mathbb{R}^{h\times w\times c}$. Now a CNN layer is made up of multiple filters, also called kernels. Let f be the number of such

filters and let g be their size; i.e., one filter is of size $g \times g \times c$, typically only referred to as $g \times g$. Each filter is applied using a stride, which we call s, and a padding, which we call p. Now to formalize what the CNN layer actually does, we define the size of the output of the layer and then specify how each component of the output can be computed. Afterwards, we will give insights into how the output can be interpreted. The spatial dimension of the output can be computed by the following equation:

$$o(i, g, p, s) = \lfloor \frac{i - g + 2p}{s} + 1 \rfloor, \tag{5.6.1}$$

where i refers to the input size, and o refers to the output size. Now we can compute the output width $ow = o(w, g, p, s)$ and the output height $oh = o(h, g, p, s)$. Thus the output O is finally in $\mathbb{R}^{oh \times ow \times f}$. We typically refer to both the input and the output as images, even though the output can typically not be visualized in the same way the input can. Finally, each component of the output can be computed as

$$O_{i,j,k} := \sigma \left(\sum_{l \in \{1,\dots,g\}} \sum_{m \in \{1,\dots,g\}} \sum_{n \in \{1,\dots,c\}} I_{si+l-p, sj+m-p, n} W^k_{g-l, g-m, c-n} \right), \tag{5.6.2}$$

where $W^k \in \mathbb{R}^{g \times g \times c}$ are the weights of the k-th filter. Note that in this equation, i, j, k, l, m, and n are used as indices and do not refer to the variables defined above. However s, p, and g are the variables as defined above. The indices for W^k might be confusing. This is due to the equation being a convolution. They can also be thought of as $W^k_{l,m,n}$ for simplicity's sake. Now we can talk about the insights into what the CNN layer actually computes. Each channel of the output refers to an image of where the feature that is defined by the filter is observable. If, for example, the first filter is an edge detector, i.e., the Laplace filter, then the first slice of O, i.e., $O_{\cdot,\cdot,1}$ would be a black and white image of all edges, or simply the Laplace filter applied to the original image. However, a slight complication has to be taken into account: A filter does not only have width and height but also depth. Thus, this Laplace filter analogy only works if we either consider only one color channel of the input, or if we consider an input with only one channel, since the Laplace filter is only defined in two dimensions, not three.

One component of the output can be thought of as a neuron. As such, it is possible to think of a CNN layer as a modified NN layer. Let $vec(M)$ be the vectorization of matrix M and W be the weight matrix of the layer, i.e., $W vec(I) = vec(O)$. Since each component in one channel was produced using the same filter, these components share their weights. If $vec(O)_i$ is in the same channel as $vec(O)_j$, then $W_{i,\cdot} = W_{j,\cdot}$. This is weight sharing in mathematical terms: It is effectively a constraint in a typical NN. Also, W is quite sparse, as two input components in the same channel with a distance greater than g will never be present in the same sum of equation 5.6.2. One line in this matrix contains at most $g^2 c$ (the weight parameters of the filter) non-zero values, while

its actual size is $hwc >> g^2c$. A typical example of this would be the MNIST dataset, where one image has the size $28 \times 28 \times 1$, whereas a filter typically has the size 3×3 or 5×5, $5^2 = 25 < 784 = 28^2$. This is how the parameter count in a CNN is kept quite low.

5.7 Deep Learning

deep neural network

An ANN having 8 or more layers is called a *deep neural network*. Deep CNN are the state of the art in image classification. We now take a look at the AlexNet [10], a CNN that competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. AlexNet was trained by GPUs as they are faster in matrix and vector operations.

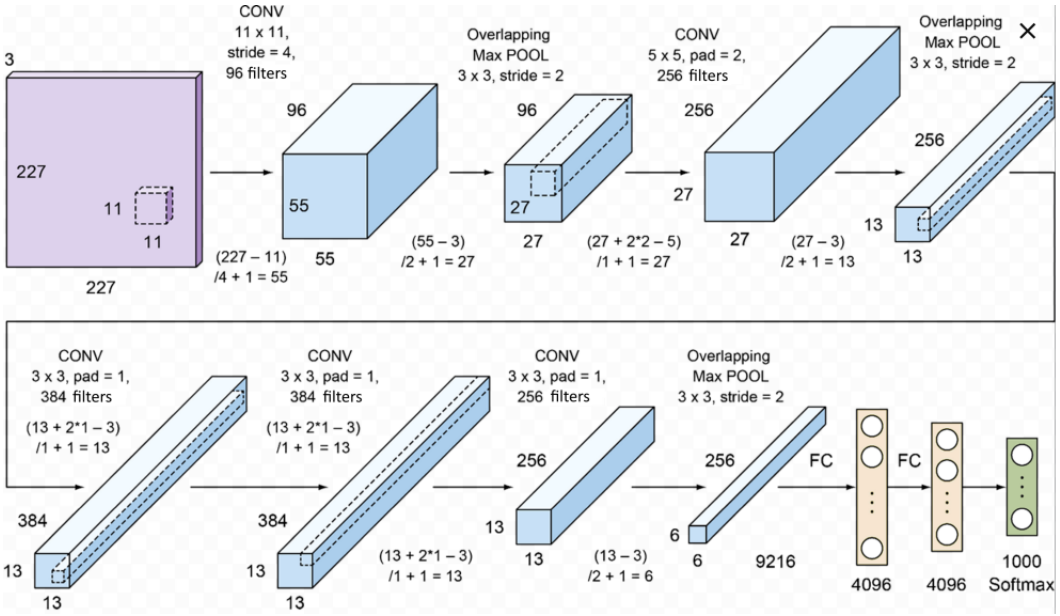

Figure 5.7.1: The AlexNet [10] consists of 8 layers. It used max pooling and the ReLu activation function. In the image, you can see the size of the patches, the number of filters used, and the size of the stride. Ultimately, we end up with fully connected layers leading us to the output.

5.8 Exercises

1. The following equation can be used to calculate the output size after a convolutional layer:

$$o = \lfloor \frac{i - k + 2p}{s} + 1 \rfloor,$$

where i is the input size, o is the output size, k is the kernel size, p is the padding, and s is the stride.

   a) Put this equation into the context of convolutional neural networks by explaining the variables i, o, k, p, and s in more detail than in this assingment.

   b) When talking about a convolutional layer, the quantities c, the number of channels, and f the number of filters are usually also important. However, they do not appear in the above formula. Why is that? What changes in a convolutional layer if you change c or f?

   c) Give one example of what the above formula could be used for, or why it is important.

   d) Consider the following. We want to change a convolutional layer to use not only one kernel size k, but two different sizes because we expect the spatial dimension of features to vary. What problems will occur, and how can they be resolved? In this assignment, consider two kernels like 3×3 and 5×5, not a non-square kernel like 3×5.

2. Solve Sheet 4.ipynb.

3. Solve Sheet 5.ipynb.

6 Overfitting & Regularization

6.1 Overfitting & Underfitting

In the previous chapters, we learned about prediction functions but never talked about how well they describe our data. Our prediction function might be too simple, leading to inaccurate predictions. Then again, our prediction function might be too complex and adapt too strongly to the data. In machine learning, we want to find a middle ground: The prediction function should neither be too complex nor too simple.

Figure 6.1.1: Various SVM prediction functions using various Gaussian kernel width. We see that the middle one with moderate width describes our data best.

Definition 6.1.1 (Underfitting)**.** Learning a classifier that is too simple (not complex enough to describe the training data well) is called *underfitting*. **underfitting**

Definition 6.1.2 (Overfitting)**.** Learning a classifier that is too complex and fits the training data too well (does not "generalize" to new data) is called *overfitting*. **overfitting**

We can observe the underfitting and overfitting behavior in other learning machines as well. Let us consider the k-nearest neighbor algorithm:

- If we choose k too small, we consider too few points to make generalized predictions, which leads to overfitting.
- If we choose k too large, we include points that might be too far away to be locally useful, which leads to underfitting.

Our high-level idea to avoid overfitting and underfitting is as follows:

1. Choose a sufficiently complex classifier to avoid underfitting.

2. Use a technique called *regularization* to avoid overfitting.

6.2 Unifying Loss View

We have already seen quite a few loss functions. Different learning machines try to minimize other loss functions. In the following, we would like to take a more abstract view of the learning machines we have studied so far (SVM, LR, ANN). Loss functions can also be viewed as an asset of a learning machine which we can change to obtain other learning machines. Let us take a look at the following optimization problem, which tries to unify all loss functions we have covered so far in one equation:

$$\min_{[W,]b,\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ell\left(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right)\right) \left[+\frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\mathrm{Fro}}^2\right]. \tag{UE}$$

The above general formulation lets us recover several learning problems introduced earlier in this course, as shown below:

1. Let $\ell(t) := \max\{0, 1-t\}$ and $\phi := \mathrm{id}$. By leaving out the bracketed values, we obtain our standard soft-margin SVM (3.3.9):

$$\min_{b,\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right)\right). \tag{SVM}$$

2. Let $\ell(t) := \max\{0, 1-t\}$ and $\phi := \phi_k$. By leaving out the bracketed values, we obtain our kernel SVM (4.2.5):

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\langle \mathbf{w}, \phi_k\left(\mathbf{x}_i\right)\rangle + b\right)\right). \tag{Kernel SVM}$$

3. Let $\ell(t) := \ln((1+\exp(-t))$ and $\phi := \mathrm{id}$. By leaving out the bracketed values, we obtain logistic regression (3.5.1):

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ln\left(1 + \exp\left(-y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right)\right)\right). \tag{LR}$$

4. Let $\ell(t) := \ln((1+\exp(-t))$ and $\phi := \phi_k$. By leaving out the bracketed values, we obtain kernelized logistic regression:

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ln\left(1 + \exp\left(-y_i\left(\langle \mathbf{w}, \phi_k\left(\mathbf{x}_i\right)\rangle + b\right)\right)\right). \tag{Kernel LR}$$

5. Let $\ell(t) := \ln((1+\exp(-t))$, $\phi := \phi_W$, then taking into account the bracketed values, we obtain our ANN (5.3.1):

$$\min_{b,\mathbf{w},W} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\mathrm{Fro}}^2 + C\sum_{i=1}^{n} \ln\left(1 + \exp\left(-y_i\left(\mathbf{w}^\top \phi_W\left(\mathbf{x}_i\right) + b\right)\right)\right). \tag{ANN}$$

loss ℓ \ map ϕ	id	ϕ_k	ϕ_W
hinge	linear SVM	kernel SVM	[7]
logistic	linear LR	kernel LR	ANN

Table 6.1: The 5 big learning machines summarized in a table.

We summarize the above results in Table 6.1. Note that the unifying equation (UE) contains, for every training example $(\mathbf{x}_i, y_i)$, a term

$$\ell(\underbrace{y_i\langle \mathbf{w}, \phi(\mathbf{x})\rangle + b}_{=:t_i}),$$

which we call the *loss* of the i-th example.

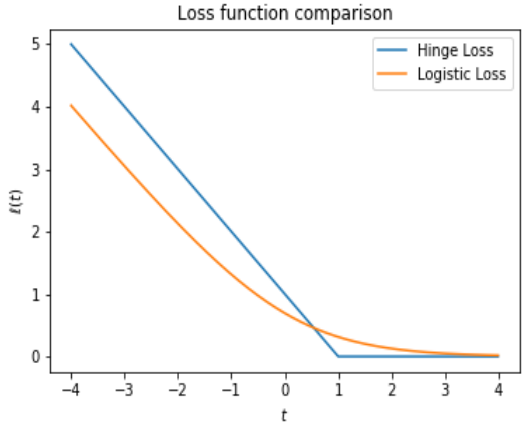

Figure 6.2.1: Large t leads to lower $\ell(t)$ values.

We try to minimize:

$$\sum_{i=1}^{n} \ell\left(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right)\right),$$

which promotes large t_i values. This is intuitive as $t_i > 0$ means classifying the i-th example correctly. In theory, we are allowed to choose arbitrary loss functions. Let us have a look at the 0-1 loss function.

Definition 6.2.1 (0-1 Loss)**.** The function $\ell(t) := \text{sign}(-t)$ is called *0-1 loss*. **0-1 loss**

Observation 6.2.2. We consider

$$\ell(t_i) := -\text{sign}(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right)).$$

As discussed in Chapter 1, we have $\ell(t_i) = 1$ when the label is misclassified $(t_i < 0)$ and $\ell(t_i) = 0$ otherwise. Thus the cumulative $0-1$ loss,

$$\sum_{i=1}^{n} \ell\left(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right)\right),$$

[7] Hinge loss is theoretically possible but uncommon in neural networks.

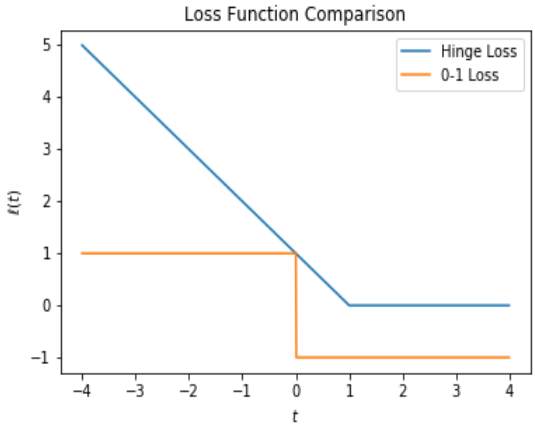

Figure 6.2.2: Hinge loss and 0-1 loss in comparison.

measures the number of training errors. Unfortunately, optimizing over this discrete problem is NP-hard, so we need to consider convex alternatives such as hinge loss and logistic loss.

6.3 Regularization

Let us recall the unifiying equation with the parameters $\theta := (b, \mathbf{w}, [, W])$ and with the bracketed terms only for ANNs:

$$\min_{[W,]b,\mathbf{w}} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2 \left[+\frac{1}{2}\sum_{l=1}^{L}\|W_l\|^2\right]}_{\text{regularizer } R(\theta)} + \underbrace{C}_{\text{regularization constant}} \left(\underbrace{\sum_{i=1}^{n} \ell\left(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right)\right)}_{\text{loss } L(\theta)} \right).$$

We want to discuss the influence of the regularization constant C. Increasing the value of C means higher importance of the loss term $L(\theta)$ and lower importance of the regularizer $R(\theta)$. The regularizer $R(\theta)$ imposes a penalty on the complexity of our chosen prediction function. Regularization is a general concept restricting the choice of parameters. We observe that:

High regularization constant $C \implies$ low regularization $\implies$ high overfitting

Low regularization constant $C \implies$ high regularization $\implies$ low overfitting.

Theorem 6.3.1. *If $R(\theta)$ and $L(\theta)$ are convex functions, then (under mild assumptions[8])*

$$\min_{\theta} \underbrace{R(\theta)}_{\text{regularizer}} + \underbrace{C}_{\text{regularization constant}} \underbrace{L(\theta)}_{\text{loss}},$$

[8] https://en.wikipedia.org/wiki/Slater%27s_condition

is equivalent to

$$\begin{aligned} \min_\theta \quad & L(\theta) \\ \text{s.t.} \quad & R(\theta) \leq \tilde{C} \end{aligned}$$

for an appropriate choice of $\tilde{C}$.

Proof. By the Lagrangian duality theorem (L) (strong duality)

$$\begin{aligned} \min_\theta L(\theta)) \quad & \overset{(L)}{=} \max_{\lambda \geq 0} \min_\theta L(\theta) + \lambda(R(\theta) - \tilde{C}) \\ \text{s.t.} \quad R(\theta) \leq \tilde{C} \quad & \overset{(L)}{=} \min_\theta \max_{\lambda \geq 0} L(\theta) + \lambda(R(\theta) - \tilde{C}) \\ & = \min_\theta L(\theta) + \lambda^*(R(\theta) - \tilde{C}) \end{aligned}$$

Observe now that

$$\begin{aligned} \arg\min_\theta L(\theta)) \quad & = \arg\min_\theta L(\theta) + \lambda^*(R(\theta) - \tilde{C}) \\ \text{s.t.} \quad R(\theta) \leq \tilde{C} \quad & = \arg\min_\theta L(\theta) + \lambda^* R(\theta) \\ & = \arg\min_\theta \frac{1}{\lambda^*} L(\theta) + \frac{1}{\lambda^*} \lambda^* R(\theta) \\ & = \arg\min_\theta R(\theta) + \underbrace{\frac{1}{\lambda^*}}_{:=C} L(\theta) \end{aligned}$$

where λ^* denotes the optimal value for λ in the maximization problem. □

The proof is important because it tells us how to generate a regularizer from a hard constraint. In other words, the regularizer origin is a constraint. By changing $\tilde{C}$, we can control the size of the set

$$\{\theta : R(\theta) \leq \tilde{C}\}.$$

The larger the set, the more likely our learning machine will pick a function that describes our training data too well, so we risk overfitting of our prediction function.

6.4 Regularization in Deep Learning

Deep neural networks have a high risk of overfitting due to their complexity. We will discuss some special strategies to avoid overfitting in the setting of neural networks.

Consider a large neural network involving many layers and neurons. Clearly, a large neural network can adapt to our training data much better than a small network, leading to the potential of overfitting. Below, we present several regularization techniques for neural networks.

Strategy 1: Early Stopping

- Split the training data into two sets:
  1. New smaller training set
  2. Validation set
- Monitor the validation set errors every couple of mini-batch iterations.
- Stop SGD when the validation error goes up.

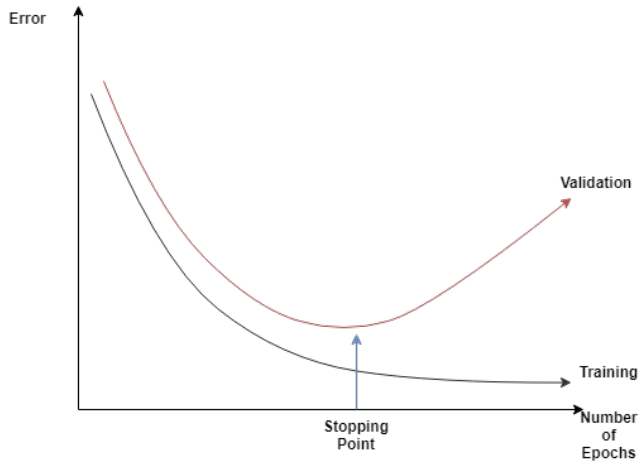

Figure 6.4.1: Initially, as the number of iterations increases, the neural network describes our control data (validation set) better, so the validation error decreases. Once the network starts overfitting, the validation error will start increasing. This is our ideal stopping point.

Strategy 2: Batch normalization

Batch normalization is a strategy in which we attempt to normalize the input of the activation function. The gradient of an activation function is typically most informative when its input has mean 0 and variance 1. However, this is dependent on the activation function. Some examples:

- ReLU: If its input is very large (very small), its gradient is 1 (0). As such, ReLU does not seem to be relevant unless its input is both large and small within a batch, in which case its mean will be around 0.
- All sigmoidal functions (i.e., sigmoid, tanh, err) have vanishing gradients, i.e., for very small or very large input, their gradients are close to 0. As such, we want their input to have mean 0 and small variance.

With this in mind, we can learn parameters allowing us to normalize the input of an activation function. Batch normalization first normalizes the input by learning the mean μ and variance σ and subtracting this learned mean from the input and dividing by the variance, normalizing the input to mean 0 and variance 1. It additionally allows learning the parameters γ and β, which rescale and shift the normalized input. After batch normalization, the input to the activation can have any learned mean and variance.

Algorithm Batch Normalization

Denote, for a neuron f, its activation values on a mini-batch by $\mathbf{f} = (f_1, \dots f_B)$. Then, for each mini-batch in the SGD optimization and every neuron in the ANN, do:

1: Center each neuron activation: $\forall i = 1, \dots, B : f_i \leftarrow f_i - \mu_f$.

2: Normalize the spread: $f_i \leftarrow f_i / \sigma_f$.

3: Overwrite the neuron's activation by the learned parameters (γ, β) with $\gamma \mathbf{f} + \beta$.

Definition 6.4.1 (Ensemble Methods)**.** *Ensemble methods* are methods that aggregate the prediction from multiple predictors.

ensemble methods

▼———————————

Example 6.1. [Ensemble Method] A typical ensemble method would be:

1. Train k machine learning algorithms, resulting in prediction functions $f_1, \dots, f_k$.
2. Predict by majority vote: $f(\mathbf{x}) = +1$ if $|i : f_i(\mathbf{x}) = +1| \geq |i : f_i(\mathbf{x}) = -1|$ and $f(\mathbf{x}) = -1$ elsewise.

The drawback to ensemble methods is that training multiple deep ANNs is very costly. We try to bypass this problem by using the following trick.

▲———————————

Strategy 3: Dropout Regularization

We simulate many ANNs by randomly hiding some neurons from our original net in each SGD step. This process is formally called *dropout regularization*. Dropout regularization randomly hides (makes unusable) in each mini-batch SGD iteration a fixed percentage of randomly selected neurons of the network. The idea is that the network cannot focus on one feature, as this feature might not be present during optimization because it might be hidden. As such, the network learns more general representations for concepts.

Strategy 4: Data Augmentation

The optimal setting for optimization is online learning. In this setting, there is an infinite amount of data and in each optimization step, new, previously unseen data is used. Obviously, online learning is not very realistic, save for a very few problems. However, *data augmentation* creates "new training data" from old data. For example, an image of a cat could be mirrored, translated, noised, rescaled, zoomed in, zoomed out, etc. This leads to more informative training data. It promotes generalization, as we are adding variety to our training data.

Strategy 5: Transfer Learning

Assuming we have a small training dataset, we can do the following:

1. Pre-training: Download an ANN from the web that was pre-trained on huge numbers of images, or alternatively train an ANN on some huge dataset.

2. Fine-tuning: Run SGD optimization on our small dataset, but use the ANN from the previous step, effectively "fine-tuning" it to the small dataset.

This regularization strategy is known as *transfer learning*, that is, transferring information from a related problem onto a learning problem.

6.5 Exercises

1. In this question, we investigate what influences models to overfit or underfit. Please list every ML model we have investigated so far in ML1. For each model, list what influences the model's power, i.e., what would have to be changed in the model for it to potentially overfit, underfit, or fit well. Try to list each possible change for each model (most models have methods to change their representative power without changing their regularization).

2. Consider you are given an ML model together with hyperparameters for adjusting the model's power, i.e., you have options to change whether the model overfits, underfits, or fits well. There might be an infinite number of possibilities for setting the hyperparameters, but you may assume that each hyperparameter can only be set to a rational number. You may also assume each hyperparameter influences the model's power continuously. That is, if the hyperparameter increases, the power increases and vice versa. This exercise is not about finding the optimal procedure, which is an active research question. This exercise is about your approach and you will be graded on how good your approach is, given the information provided in the lecture.

    a) Describe a methodology for using these adjustments to determine the appropriate power for a given problem. Be exact with your methodology. Your methodology should set the hyperparameters somehow.

    b) Now consider your methodology as the entire training procedure, resulting in a model. Does your model have hyperparameters?

    c) When might your model overfit or underfit, i.e., how can you determine your model's power?

    d) How do your model's hyperparameters influence computation times?

3. Solve Sheet 6.ipynb.

7 Regression

Until now, we have only looked at classification problems. For a prediction function f, we made the restriction that $f(x) \in \{-1, +1\}$. However, real-world problems are not only of the classification kind. Suppose we are given data of houses as training data, and we want to predict the price of a house. A binary classifier does not suffice in this case. We need a real-valued prediction function.

The formal setting will be the following:

- Given training data $(\mathbf{x_i}, (y_i))$, with $\mathbf{x_i} \in \mathbb{R}^d, y_i \in \mathbb{R}$.
- Find $f : \mathbb{R}^d \to \mathbb{R}$ with $f(\mathbf{x}) \approx y$ for new data $\mathbf{x}, y$.

In the following, we will look at a classical method for solving these kinds of problems, which is called *regression*.

7.1 Linear Regression

linear regression

The idea of *linear regression* is quite simple. Suppose we are given some data points. A first idea would be to approximate these values linearly by a hyperplane:

$$y \approx \mathbf{w}^\top \mathbf{x}.$$

For now, we will intentionally ignore the displacement parameter $+b$ because it will turn out that we do not need it (more on this later). A line would be good if it reduced the error. One way to measure the error of data point i would be the squared distance

$$(y_i - \mathbf{w}^\top \mathbf{x_i})^2.$$

Observe that this is an intuitive choice of error because it is always positive and it is higher when the value predicted by the hyperplane is far away from y. Finally, we want to minimize the error for all data points, so we just sum over them, leading us to the following optimization problem:

least-squares regression

Definition 7.1.1 (Least-squares regression (Legendre, 1805))**.**

$$\mathbf{w}_{\text{LS}} := \arg\min_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^\top \mathbf{x}_i\right)^2 = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \left\|\mathbf{y} - X^\top \mathbf{w}\right\|^2.$$

We will also define a loss function exhibiting our choice of error:

Definition 7.1.2 (Least-squares loss)**.** The function $\ell(t, y) := (t - y)^2$ is called *least-squares loss*. Our idea so far is promising but prone to overfitting because our model lacks regularization. To make our model less prone to overfitting, we add a regularizer, resulting in:

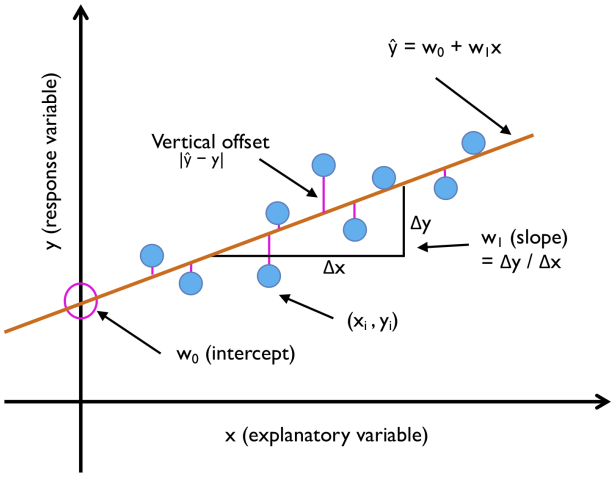

Figure 7.1.1: Illustration of the linear regression problem in two dimensions. The hyperplane $\hat{y} = w_0 + w_l x$ will predict the y-value of a new input x. This hyperplane is chosen, as it minimizes the sum of the squared distances (vertical offsets).

ridge regression

Definition 7.1.3 (Ridge regression)**.**

$$\mathbf{w}_{\mathrm{RR}} := \arg\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2 + C\left\|\mathbf{y} - X^\top \mathbf{w}\right\|^2}_{=:\mathcal{L}(\mathbf{w})}.$$

Let us now look into how to find the optimum of the above regression problems. One possibility would be to use SGD, but it turns out that our problem is even simpler. Ridge regression is an unconstrained optimization problem. Thus, we find the global optima by differentiation. Let us calculate:

$$\begin{aligned}
\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) =& \nabla_{\mathbf{w}}\left(\frac{1}{2}\|\mathbf{w}\|^2 + C\left\|\mathbf{y} - X^\top\mathbf{w}\right\|^2\right)\\
=& \nabla_{\mathbf{w}}\left(\frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\left(\mathbf{y} - X^\top\mathbf{w}\right)^\top\left(\mathbf{y} - X^\top\mathbf{w}\right)\right)\\
=& \nabla_{\mathbf{w}}\left(\frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\left(\mathbf{y}^\top - \mathbf{w}^\top X\right)\left(\mathbf{y} - X^\top\mathbf{w}\right)\right)\\
=& \nabla_{\mathbf{w}}\left(\frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\left(\mathbf{y}^\top\mathbf{y} - \mathbf{y}^\top X^\top\mathbf{w} - \mathbf{w}^\top X\mathbf{y} + \mathbf{w}^\top XX^\top\mathbf{w}\right)\right)\\
=& \nabla_{\mathbf{w}}\left(\frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\mathbf{y}^\top\mathbf{y} - 2C\mathbf{w}^\top X\mathbf{y} + C\mathbf{w}^\top XX^\top\mathbf{w}\right)\\
=& \mathbf{w} - 2CX\mathbf{y} + 2CXX^\top\mathbf{w}.
\end{aligned}$$

Finally, we set $\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = 0$ to find our optimum:

$$\begin{aligned}
\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = 0 &\iff \mathbf{w} - 2CX\mathbf{y} + 2CXX^\top\mathbf{w} = 0 \\
&\iff -2CX\mathbf{y} + (I + 2CXX^\top)\mathbf{w} = 0 \\
&\iff (I + 2CCXX^\top)\mathbf{w} = 2CX\mathbf{y} \\
&\iff \mathbf{w} = (I + 2CXX^\top)^{-1}2CX\mathbf{y} \\
&\iff \mathbf{w} = 2C(I + 2CXX^\top)^{-1}Xy \\
&\iff \mathbf{w} = \left(\frac{1}{2C}I + XX^\top\right)^{-1}Xy.
\end{aligned}$$

The above calculation proves the following theorem:

Theorem 7.1.4.

$$\mathbf{w}_{\mathrm{RR}} = \left(XX^\top + \frac{1}{2C}I\right)^{-1}Xy.$$

We have considered a linear model without bias b. However, we can easily incorporate bias into any linear learning machine (regression, SVM, etc.) by applying the following trick: Let $\mathbf{x_1}, \dots \mathbf{x_n}$ be our training data and define

$$\tilde{\mathbf{x}}_i := \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix},$$

and change our training data matrix into

$$\tilde{X} := (\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_n) = \begin{pmatrix} X \\ \mathbf{1}^\top \end{pmatrix}.$$

We can conclude that every solution:

$$\begin{aligned}
\mathbf{w}^* :=& \arg\min_{\mathbf{w}\in\mathbb{R}^{d+1}} \frac{1}{2}\|\tilde{\mathbf{w}}\|^2 + C\left\|\mathbf{y} - \tilde{X}^\top\tilde{\mathbf{w}}\right\|^2 \\
=& \arg\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}} \frac{1}{2}\left(\|\mathbf{w}\|^2 + b^2\right) + C\left\|\mathbf{y} - X^\top\mathbf{w} - b\mathbf{1}\right\|^2.
\end{aligned}$$

Finally, observe the difference to ridge regression. Here we are also regularizing the bias b. If we wish to remove it, we just subtract it out of the minimization.

7.2 Leave-one-out cross-validation (LOOCV)

Let us think about how we can choose our regularization constant C. One idea would be to separate our training data set into a smaller training data set and a test set. We would then measure the average error and choose our best regularization constant. To make things fair, we would also alternate through all possible test sets and training sets. Let us make this idea more formal in the following algorithm.

Algorithm *k*-fold Cross-Validation

1: split data into $k \overset{\text{e.g.}}{=} 10$ equally-sized chunks (called "folds")
2: **for** $i \leftarrow 1$ to k and $C \overset{\text{e.g.}}{\in} \{0.01, 0.1, 1, 10, 100\}$ **do**
3: use i-th fold as test set and union of all others as training set
4: train learner on training set (using C) and test on test set
5: **end for**
6: output learner with lowest average error

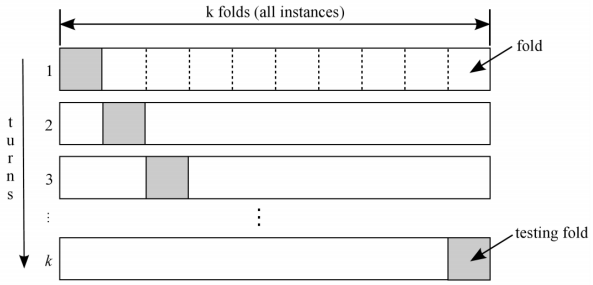

Figure 7.2.1: Visualization of the k-fold Algorithm.

Back in the classification setting, the meaning of error was pretty clear. Let us see how we can define the meaning of error in the regression setting.

root mean square error

Definition 7.2.1 (Root Mean Square Error)**.** Let $\{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n)\}$ be a test set, and let f be a learned regression function. The *root mean squared error* (RMSE) of f is:

$$\text{RMSE}(f) := \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(f\left(\mathbf{x}_i\right) - y_i\right)^2}.$$

Sometimes our dataset is very small. In such a dataset, it would be intuitive to choose $k = n$.

leave-one-out Cross-Validation

Definition 7.2.2 (Leave-One-Out Cross-Validation (LOOCV))**.** The special case of $k = n$ in k-fold Cross-Validation is called *leave-one-out cross-validation.*

We can also use LOOCV for determining other parameters, like kernel width or learning rate in ANNs. Unfortunately, the runtime of LOOCV is relatively high.

- In ridge regression, we loop over all data points with runtime $O(n)$.
- Each time we train with $n-1$ data points, which needs to invert a matrix with runtime $O(d^3)$.
- Resulting in a total runtime of $O(nd^3)$.

Turns out, we can be faster. Let us look at the particular setting of LOOCV. Recall:

$$\mathbf{w}_{\text{RR}} = (\underbrace{XX^\top}_{=\sum_{i=1}^n \mathbf{x}_i\mathbf{x}_i^\top} + \frac{1}{2C} I)^{-1} \underbrace{Xy}_{=\sum_{i=1}^n \mathbf{x}_i y_i} .$$

Let $\mathbf{w_i}$ be the RR solution when the i-th data point is left out during training. We have:

$$\mathbf{w}_i = \left(XX^\top - \mathbf{x}_i\mathbf{x}_i^\top + \frac{1}{2C} I \right)^{-1} (Xy - \mathbf{x}_i y_i) .$$

The error in LOOCV is defined as:

$$\text{RSME}_{\text{loocv}} = \sqrt{\frac{1}{n}\sum_{i=1}^n \left(\mathbf{x}_i^\top \mathbf{w}_i - y_i\right)^2}.$$

We will now see that we actually do not need to invert n times. We will make use of the following theorem:

Theorem 7.2.3 (Sherman-Morrison Formula)**.** *Let $A \in \mathbb{R}^{d\times d}$ be an invertible matrix, and let $\mathbf{u} \in \mathbb{R}^d$. If $\mathbf{u}^\top A^{-1}\mathbf{u} \neq 1$, then:*

$$\left(A - \mathbf{u}\mathbf{u}^\top\right)^{-1} = A^{-1} + \frac{A^{-1}\mathbf{u}\mathbf{u}^\top A^{-1}}{1 - \mathbf{u}^\top A^{-1}\mathbf{u}}$$

First, let us rewrite:

$$\mathbf{w}_i = \left(\underbrace{XX^\top + \frac{1}{2C} I}_{:=A} - \mathbf{x}_i\mathbf{x}_i^\top \right)^{-1} (Xy - \mathbf{x}_i y_i)$$

By using the theorem, we get:

$$\begin{aligned}
\text{RSME}_{\text{loocv}} &= \sqrt{\frac{1}{n}\sum_{i=1}^n \left(\mathbf{x}_i^\top \mathbf{w}_i - y_i\right)^2} \\
&= \sqrt{\frac{1}{n}\sum_{i=1}^n \left(\mathbf{x}_i^\top \left(A^{-1} + \frac{A^{-1}\mathbf{x}_i\mathbf{x}_i^\top A^{-1}}{1 - \mathbf{x}_i^\top A^{-1}\mathbf{x}_i} \right) (Xy - \mathbf{x}_i y_i) - y_i \right)^2}
\end{aligned}$$

This shows we only need $O(d^3)$ iterations to compute the LOOCV error. But we can further simplify the equation.

Theorem 7.2.4. *The LOOCV-RMSE of ridge regression can be computed in $O\left(d^3\right)$ by:*

$$\text{RSME}_{\text{loocv}} = \sqrt{\frac{1}{n}\sum_{i=1}^n \left(\frac{\mathbf{x}_i^\top \mathbf{w}_{\text{RR}} - y_i}{1 - \mathbf{x}_i^\top A^{-1}\mathbf{x}_i} \right)^2},$$

where $A := XX^\top + \frac{1}{2C} I$.

Proof. Recalling $\mathbf{w}_{\text{RR}} = A^{-1}Xy$ and denoting $\beta_i := \mathbf{x}_i^\top A^{-1}\mathbf{x}_i$, we get:

$$\begin{aligned}
\text{RSME}_{\text{loocv}} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^\top\left(A^{-1}+\frac{A^{-1}\mathbf{x}_i\mathbf{x}_i^\top A^{-1}}{1-\mathbf{x}_i^\top A^{-1}\mathbf{x}_i}\right)(Xy-\mathbf{x}_i y_i)-y_i\right)^2} \\
&= \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{x}_i^\top \mathbf{w}_{RR}+\frac{\beta_i \mathbf{x}_i^\top \mathbf{w}_{RR}}{1-\beta_i}-\beta_i y_i-\frac{\beta_i^2}{1-\beta_i}y_i-y_i\right)^2} \\
&= \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\left(1+\frac{\beta_i}{1-\beta_i}\right)\mathbf{x}_i^\top \mathbf{w}_{\text{RR}}-\left(\beta_i+\frac{\beta_i^2}{1-\beta_i}+1\right)y_i\right)^2} \\
&= \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathbf{x}_i^\top \mathbf{w}_{\text{RR}}-y_i}{1-\beta_i}\right)^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathbf{x}_i^\top \mathbf{w}_{\text{RR}}-y_i}{1-\mathbf{x}_i^\top A^{-1}\mathbf{x}_i}\right)^2}.
\end{aligned}$$

□

7.3 Non-Linear Regression

As one might imagine, a linear function is not always the best prediction function. We will now see how to employ non-linearity in regression.

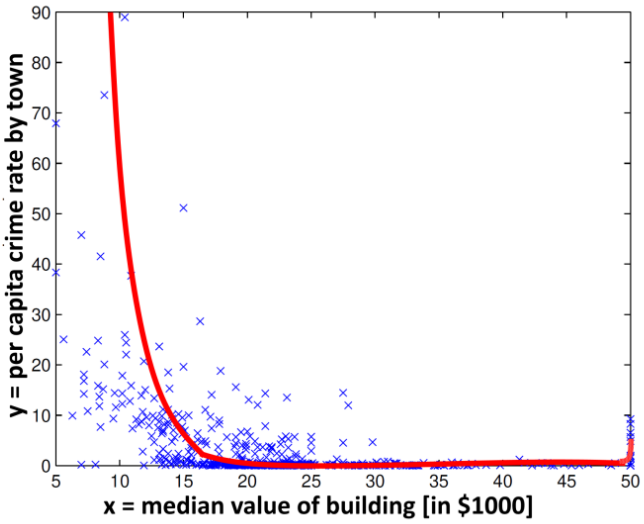

Figure 7.3.1: The red non-linear function approximates this dataset much better than any linear function would.

But first let us try a quick fix, with which we could still use linear regression. Consider applying a log transformation on the label $y = log(y)$. We take a look at a generalization of the above method.

Box Cox transformation

Definition 7.3.1 (Box Cox Transformation)**.** The *Box-Cox transformation*

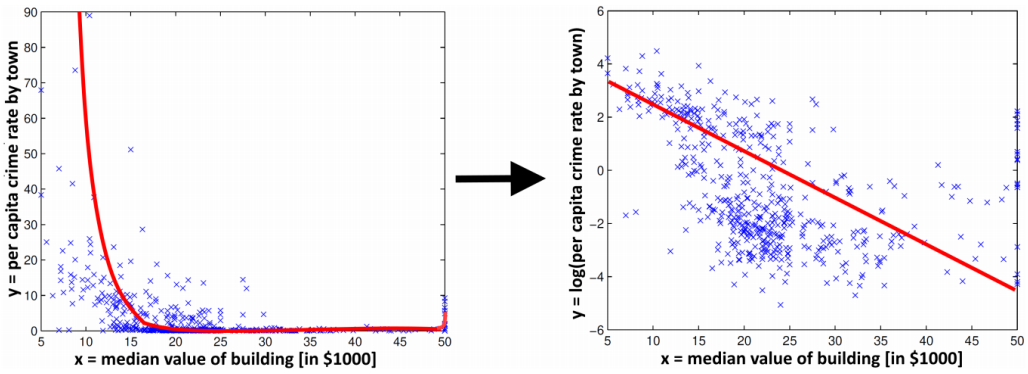

Figure 7.3.2: After the log transformation, a linear regression line is a reasonable approach.

(or 'power transformation') with the parameter λ is:

$$x_{\text{new}} \leftarrow \begin{cases} \log(x_{\text{old}}) & \text{if } \lambda = 0 \\ \frac{x_{01}^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0 \end{cases}$$

The parameter λ has intuitive interpretations, as shown below:

- $\lambda > 1:$ data is stretched(e.g $\cdot\ \lambda = 2:$ quadratically).
- $0 < \lambda < 1:$ data is concentrated (e.g. $\lambda = 0.5:$ square root.)
- $\lambda = 0:$ log transform.
- $\lambda < 0:$ analogously, with the order of data reversed.

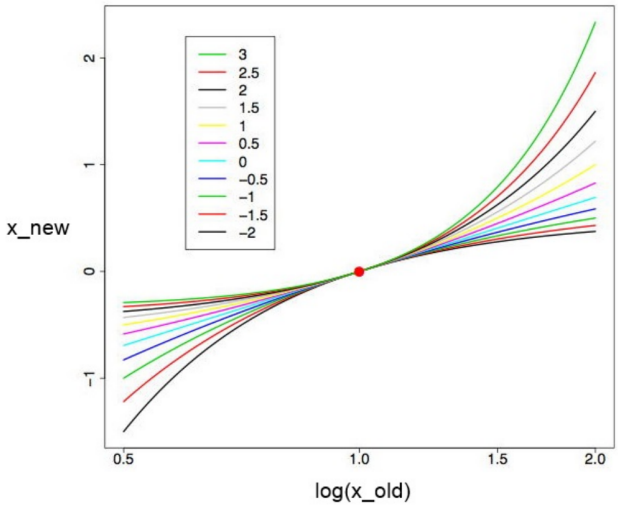

Figure 7.3.3: Visualization of the Box-Cox transformation. Here we see the parameters λ and their influence.

In general, before using any other method, one would try out some transformations on the dataset. If this still does not help, we can use kernel ridge regression.

kernel ridge regression

Definition 7.3.2 (Kernel Ridge Regression)**.** Let k be a kernel with an associated feature map $\phi : \mathbb{R}^d \to \mathcal{H}$, i.e., $k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle$. Then *kernel ridge regression* (KRR) is defined as:

$$\mathbf{w}_{KRR} := \arg\min_{\mathbf{w} \in \mathcal{H}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \|y_i - \langle \mathbf{w}, \phi(x_i) \rangle\|^2 .$$

Let us try to apply the kernel trick on KRR. Recall the representer theorem statement: The optimal solution is in the span of the dataset:

$$\exists \boldsymbol{\alpha} \in \mathbb{R}^n : \quad \mathbf{w}_{KRR} = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i) = \phi(X)\boldsymbol{\alpha},$$

with

$$\phi(X) := (\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)).$$

Recalling the definition of the kernel matrix K, we have:

$$\begin{aligned}
&\min_{\mathbf{w} \in \mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 + C \left\|\mathbf{y} - \phi(X)^\top \mathbf{w}\right\|^2 \\
&= \min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{1}{2} \underbrace{\|\phi(X)\alpha\|^2}_{\boldsymbol{\alpha}^\top \underbrace{\phi(X)^\top \phi(X)}_{=K} \boldsymbol{\alpha}} + C\|\mathbf{y} - \underbrace{\phi(X)^\top \phi(X)}_{=K} \alpha\|^2 \\
&= \min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{1}{2}\boldsymbol{\alpha}^\top K \boldsymbol{\alpha} + C\|\mathbf{y} - K\boldsymbol{\alpha}\|^2.
\end{aligned}$$

By differentiation, we can also find the optimal solution, leading us to the following theorem:

Theorem 7.3.3. *The solution of KRR is given by*

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) \qquad \textit{with} \qquad \alpha = \left(K + \frac{1}{2C} I_{n \times n}\right)^{-1} y$$

.

Proof. Left as an exercise. □

The solution can thus be computed in $O(n^3)$. Usually we use KRR together with a Gaussian kernel, but sometimes it can make sense to use a linear kernel. By definition, linear kernel regression is the same as ridge regression. If $n < d$, it makes sense to use a linear kernel instead of ridge regression, as the matrix we need to invert is of size $n \times n$ as seen above, and the matrix inversion is in $O(n^3)$ time instead of $O(d^3)$. We also use regression in deep learning, for tasks like determining the price of a house, given a picture. We only need to change the loss to least squares.

deep regression

Definition 7.3.4 (Deep Regression)**.** Deep regression predicts $f(\mathbf{x}) = \mathbf{w}_*^\top \phi_{W_*}(\mathbf{x})$, where:

$$(\mathbf{w}_*, W_*) := \arg\min_{\mathbf{w}, W} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2} \sum_{l=1}^{L} \|W_l\|_{\text{Fro}}^2 + C \sum_{i=1}^{n} \left(y_i - \mathbf{w}^\top \phi_W(\mathbf{x}_i)\right)^2$$

7.4 Unifying View

Recall our unifying equation:

$$\min_{[W,]b,\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ell\left(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right)\right) \left[+\frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2\right]. \tag{UE}$$

We would like to also add regression to this equation, making it even more general. In order to do so, we define a new notation for our loss function:

$$l(t,y) := \begin{cases} (t-y)^2 & \text{for regression} \\ \ell(yt) & \text{for classification} \end{cases}$$

We can now get all of our known loss functions by plugging in:

- Use $l(t,y) := \max(0, 1-yt)$ for SVM ("hinge loss").
- Use $l(t,y) := \ln(1+\exp(-yt))$ for LR and ANN ("logistic loss").
- Use $l(t,y) := (t-y)^2$ for regression.

Similarly, by plugging the following kernel function in each loss function from above into UE, we get:

- $\phi :=$ id for linear SVM, linear LR, and RR.
- $\phi := \phi_k$ for kernel SVM, kernel LR, and KRR.
- $\phi := \phi_W$ for ANN and DR.

Definition 7.4.1 (Support Vector Regression (SVR))**.** Using a loss function called ϵ *insensitive loss:* **support vector regression**

$$\ell(t,y) := \max(0, |y-t| - \epsilon),$$

we can define *support vector regression*, whose aim it is to find a line predicting $f(\mathbf{x_i})$, such that the predicted value does not deviate more than ϵ to y_i. This error is captured by the distance of the prediction line and y_i as

$$\mathbf{w}^*_{\text{SVR}} := \arg\min_{\mathbf{w}\in\mathcal{H}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, |y_i - \langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle| - \epsilon\right).$$

7.5 Exercises

1. Consider the kernel ridge regression optimization problem. Let $\alpha^* \in \mathbb{R}^d$ be the vector that minimizes the loss function. Show that:

$$\alpha^* = \left(K + \frac{1}{2C}\mathrm{I}_{n\times n}\right)^{-1} y$$

2. In the lecture, we found a closed-form solution for linear ridge regression and afterwards incorporated b by simply changing the dataset slightly.

This means, however, that b is regularized during optimization. What would happen if we introduced b in a different way? Consider linear ridge regression with offset

$$\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}} \frac{1}{2}\|\mathbf{w}\|^2 + C\left\|\mathbf{y} - (X^T\mathbf{w} + \hat{\mathbf{b}})\right\|^2, \tag{7.5.1}$$

where $\forall i : \hat{b}_i = b$. $\hat{\mathbf{b}}$ simply copies b into each component. Alternatively, the norm could be written as a sum incorporating only b, as follows:

$$\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \left(y_i - (\mathbf{x}_i^T\mathbf{w} + b)\right)^2 \tag{7.5.2}$$

(7.5.1) and (7.5.2) have the same closed-form solution. Find this solution. In doing so, choose the version from the above two that you prefer ((7.5.1) or (7.5.2)).

3. Solve Sheet 7.ipynb.

8 Clustering

So far, we have only looked at so-called *supervised learning* settings. In these scenarios, we are given

$$\begin{aligned} &\text{Inputs:} && \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d \\ &\text{Labels:} && y_1, ..., y_n \\ & && y_i \in \begin{cases} \{-1, +1\} & \text{for binary classification} \\ \mathbb{R} & \text{for regression} \end{cases} \end{aligned}$$

Now we will take a look at what happens if no labels are given, i.e., if $y_1, ..., y_n$ are unknown. This setting is called *unsupervised learning* and we will start by looking at so-called *clustering* problems.

8.1 Linear Clustering

Definition 8.1.1 (Clustering)**.** *Clustering* is the process of organizing objects into groups - called clusters - whose members are similar in some way, while objects in different clusters are non-similar.

clustering

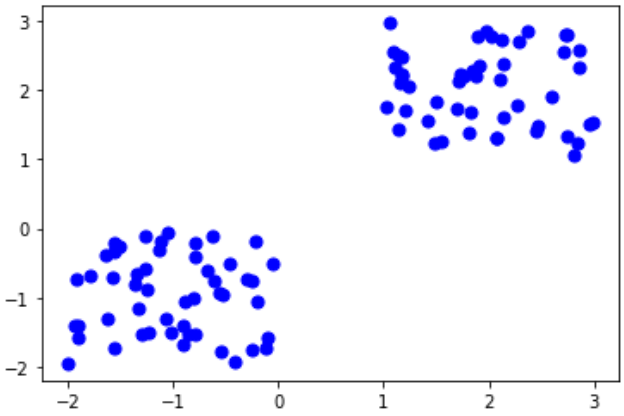

Figure 8.1.1: Two sets of points in $\mathbb{R}^2$ that could be clustered.

We are now looking for an algorithm that is capable of correctly clustering our inputs $\mathbf{x}_1, ..., \mathbf{x}_n$ in a way that allows us to make meaningful interpretations of the resulting clusters.

▼———————————————

Example 8.1. In Fig. 8.1.1, the two axes could represent the weight and tail length of two types of mice (transformed such as to fit neatly into the plot), which we want to distinguish from each other. In that case, a good algorithm should yield the following clusters:

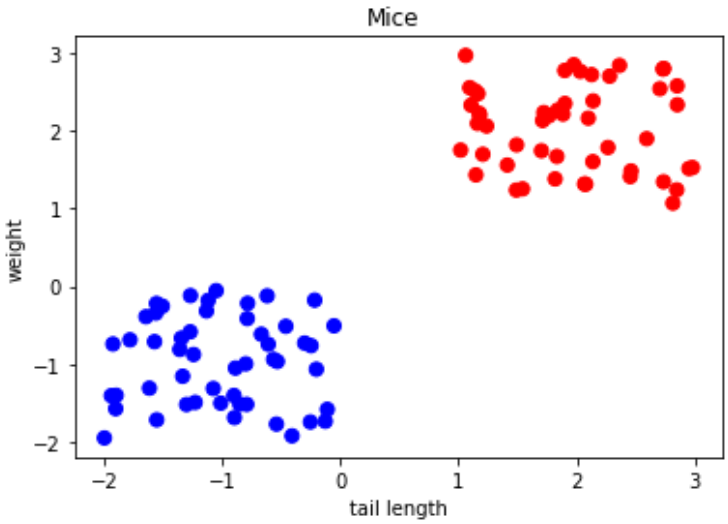

Figure 8.1.2: Clustered mice.

▲———————————————

In the above scenario, the clustering was easy to see, but since we do not know what the ground truth labels are, we cannot be sure that this is the optimal way to cluster our mice. Therefore, the results usually need to be inspected manually afterwards.

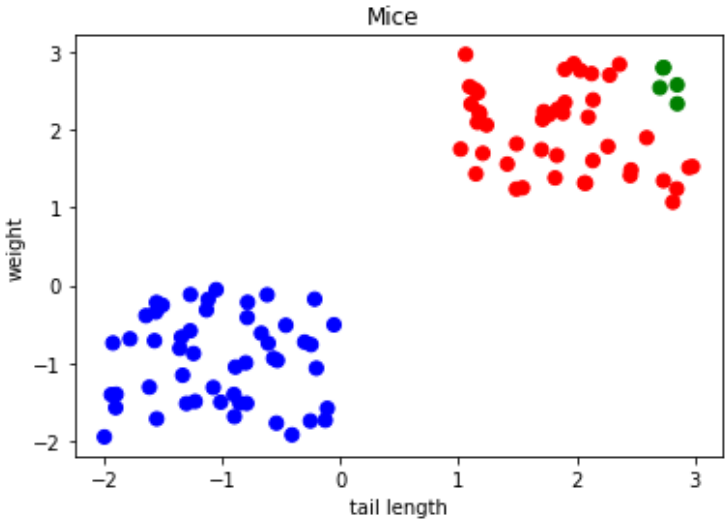

Figure 8.1.3: What an even better clustering could look like.

8.2 K-means

Now we will look at arguably the most popular clustering algorithm.

Algorithm *K*-**means**

Input: $k \in \mathbb{N}$ and $\mathbf{X} = \mathbf{x_1}, ..., \mathbf{x_n} \in \mathbb{R}^d$

1: **function** K-MEANS($k, \mathbf{X}$)
2: Initialize cluster centers $\mathbf{c_1}, ..., \mathbf{c_k}$ (e.g., randomly drawn inputs)
3: **repeat**
4: **for** i = 1 : n **do**
5: Label the input $\mathbf{x}_i$ as belonging to the nearest cluster

$$y_i = \arg\min_{j=1,...,k} \|\mathbf{x}_i - \mathbf{c}_j\|^2 .$$

6: **end for**
7: **for** j = 1 : k **do**
8: Compute cluster center $\mathbf{c}_j$ as the mean of all inputs of the j-th cluster,

$$\mathbf{c}_j := \text{mean}(\{\mathbf{x}_i : y_i = j\}).$$

9: **end for**
10: **until** Clusters do not change between subsequent iterations.
11: **return** Cluster centers $\mathbf{c_1}, ..., \mathbf{c_k}$
12: **end function**

The k-means algorithm takes two inputs, k the number of clusters we want to create and a set of inputs $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}$, which is usually be passed in the form of a matrix $X \in \mathbb{R}^{n \times d}$ where d is the number of features each input $\mathbf{x}_i$ has ($d = 2$ in our mouse example). We then initialize the cluster centers $\mathbf{c_1}, ..., \mathbf{c_k}$ by randomly picking k of our input data points $\mathbf{x_1}, ...\mathbf{x_n}$. Afterwards, we repeat the procedure described in lines 4-9 until our clusters do not change anymore. In the following, we have applied the algorithm to our mouse example. The white crosses represent the centroids. The colored regions indicate to which cluster a new input $\mathbf{x}_{n'}$ would be assigned.

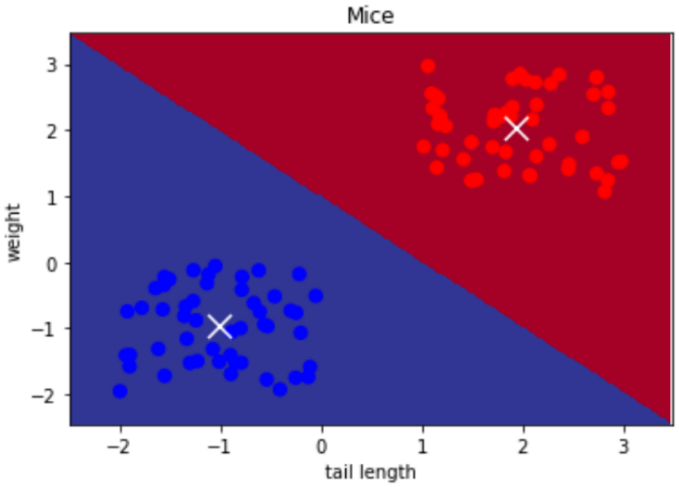

Figure 8.2.1: K-means clustering applied to our mouse example

▼———————————————

Example 8.2. [Limitations] We will now see that k-means is limited to finding linear boundaries between classes. This means it will partition the feature space into polygons. This partition is also called a *Voronoi diagram*. The polygons consist of a centroid and all points of the space that are closer to this particular centroid than to any other.

▲———————————————

Theorem 8.2.1. *K-means finds piecewise linear cluster boundaries.*

Proof. We first look at case $k = 2$. Since we only have two clusters, we can partition our inputs $\mathbf{x_1}, ..., \mathbf{x_n}$ by assigning the label $y_i = -1$ to every input that lies in cluster one and $y_i = 1$ to every input that lies in cluster two. Therefore, we get the partition

$$A := \{x_i : (x_i, y_i) \in D, y_i = -1\} \quad B := \{x_i : (x_i, y_i) \in D, y_i = +1\}.$$

From line 8 of the algorithm, we know that the centroids can be computed as

$$\mathbf{c}_{-1} = \frac{1}{|A|} \sum_{x \in A} \mathbf{x}, \quad \mathbf{c}_{+1} = \frac{1}{|B|} \sum_{x \in B} \mathbf{x}.$$

Look back at 2.2 Linear Classifier, where we presented the nearest centroid classifier. Our formulation here is equivalent to it. What happened is that for $k = 2$ k-means outputs the centroids for an NCC, which we can then use to predict new points. This also means that the cluster boundaries have to be linear.

If $k > 2$, we look at pairwise linear boundaries between clusters and use the fact that the intersection of linear boundaries creates polygons that are piece-wise linear. □

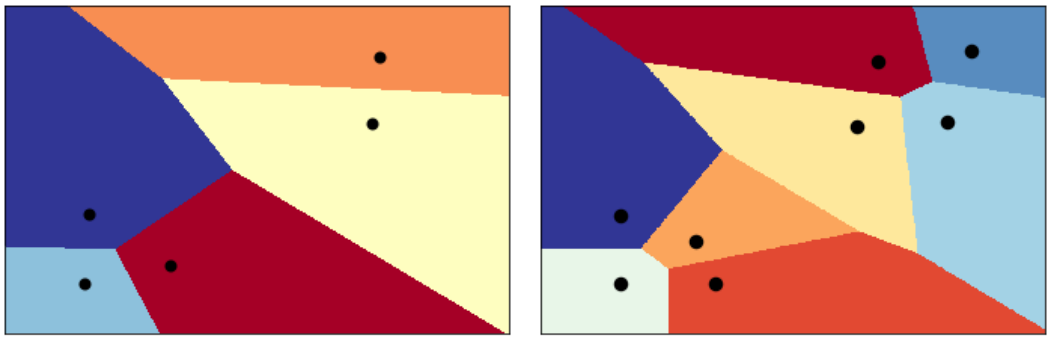

Figure 8.2.2: Voronoi diagram of our mouse example for $k = 5$ and $k = 8$.

Next up, we will see how to deal with clustering tasks where the optimal decision boundary is not linear anymore.

8.3 Non-Linear Clustering

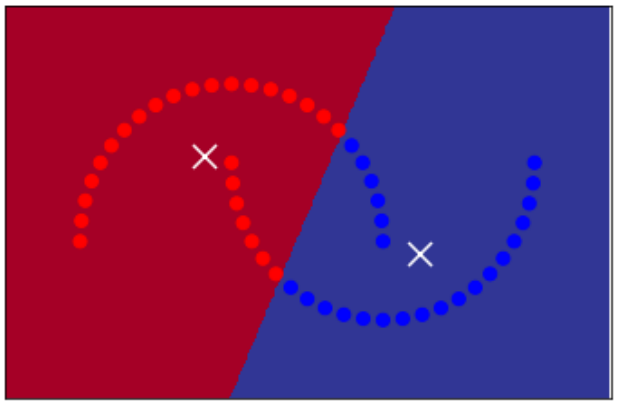

Figure 8.3.1: K-means cannot cluster this data effectively.

Kernel k-means

Just like the SVM, the k-means clustering algorithm can be kernelized. To do so, we have to change two parts of our algorithm. First of all, we will compute the norm $\|\mathbf{x}_i - \mathbf{c}_i\|^2$ line 5 via a kernel function and then we will not compute the mean in line 8 explicitly but rather find an implicit representation for it. We will now see how this works in detail. Let $\phi : \mathbb{R}^d \to \mathcal{H}$ be a kernel feature map for a kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and

$$l_j := \{i \in \{1, \dots, n\} : y_i = j\} \text{ for all } j = 1, \dots, k.$$

Then

$$\mathbf{c}_j := \frac{1}{|I_j|} \sum_{i \in I_j} \phi(\mathbf{x}_i).$$

That is, we map all inputs $\mathbf{x_1}, ..., \mathbf{x_n}$ to a higher-dimensional space via ϕ where they can be linearly separated and then apply k-means in that space. We can then completely express the norm $\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|^2$ in terms of kernel functions as

$$\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|^2 = \underbrace{\|\phi(\mathbf{x}_i)\|^2}_{=k(\mathbf{x}_i, \mathbf{x}_i)} - 2\langle \phi(\mathbf{x}_i), \mathbf{c}_j \rangle + \|\mathbf{c}_j\|^2, \tag{8.3.1}$$

where

$$\|\mathbf{c}_j\|^2 = \left\langle \frac{1}{|I_j|} \sum_{i \in I_j} \phi(\mathbf{x}_i), \frac{1}{|I_j|} \sum_{i' \in I_j} \phi(\mathbf{x}_{i'}) \right\rangle \tag{8.3.2}$$

$$= \frac{1}{|I_j|^2} \left\langle \sum_{i \in I_j} \phi(\mathbf{x}_i), \sum_{i' \in I_j} \phi(\mathbf{x}_{i'}) \right\rangle \tag{8.3.3}$$

$$= \frac{1}{|I_j|^2} \sum_{i, i' \in l_j} k(\mathbf{x}_i, \mathbf{x}_{i'}) \tag{8.3.4}$$

Here we used the kernel property $k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \phi(\mathbf{x}), \phi(\tilde{\mathbf{x}}) \rangle$ and the properties of dot products in general to move the sum and $\frac{1}{|I_j|^2}$ out of the dot product.

Through an analogous calculation, we get

$$\langle \phi(\mathbf{x}_i), \mathbf{c}_j \rangle = \frac{1}{|I_j|} \sum_{i' \in I_j} k(\mathbf{x}_i, \mathbf{x}_{i'}).$$

Therefore, (8.3.1) can be completely expressed in terms of kernel functions.

$$\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|^2 = k(\mathbf{x}_i, \mathbf{x}_i) - \frac{2}{|I_j|} \sum_{i' \in I_j} k(\mathbf{x}_i, \mathbf{x}_{i'}) + \frac{1}{|I_j|^2} \sum_{i,i' \in I_j} k(\mathbf{x}_i, \mathbf{x}_{i'}) \quad (8.3.5)$$

Since we usually cannot compute $\phi(\cdot)$ explicitly, we have to find a different way to assign $\mathbf{c}_j$ in the next step, e.g., line 8.
The trick we are applying is to assign points to a cluster C_j directly (notice the capital non-bold letter). Since we showed above that we do not need to know the cluster center $\mathbf{c}_j$, we can just skip its calculation in line 8 altogether. What counts for the classification is not a point $\mathbf{c}_j$ but rather the indices in I_j, which then determine the value of the R.H.S of equation (8.3.4). We can say that I_j is basically an approximation for $\mathbf{c}_j$ that encodes the same information, meaning which points should be clustered together. In case of I_j, this works by explicitly giving their indices and in case of $\mathbf{c}_j$, by being able to calculate the distance from every input $\mathbf{x}_1, ..., \mathbf{x}_n$ and then choosing the centroid $\mathbf{c}_j$ that is the nearest.

Algorithm Kernel k-means

Input: $k \in \mathbb{N}$ and $\mathbf{X} = \mathbf{x_1}, ..., \mathbf{x_n} \in \mathbb{R}^d$, a kernel function $\mathbf{h}$.

1: **function** K-MEANS($k, \mathbf{X}, \mathbf{h}$)
2: Randomly assign every $\mathbf{x}_i$ to one of the k clusters C_j, $j = 1, ..., k$.

3: **repeat**
4: **for** i = 1 : n **do**
5: Label the input $\mathbf{x}_i$ as belonging to the nearest cluster C_j.

$$y_i = \arg\min_{j=1,...,k} \|\phi(\mathbf{x}_i) - \mathbf{c}_j\|^2$$

$$= \arg\min_{j=1,...,k} \quad k(\mathbf{x}_i, \mathbf{x}_i) - \frac{2}{|I_j|} \sum_{i' \in I_j} k(\mathbf{x}_i, \mathbf{x}_{i'}) + \frac{1}{|I_j|^2} \sum_{i,i' \in I_j} k(\mathbf{x}_i, \mathbf{x}_{i'}).$$

6: **end for**
7: **until** Clusters do not change between subsequent iterations.

8: **return** Final clusters $C_1, ..., C_k$.
9: **end function**

We can see that the only part that really changes in comparison to k-means is line 8, where we computed each cluster as the set of inputs $\mathbf{x}_i$ whose label

y_i is j. With this algorithm, we can indeed improve our results drastically in some cases. Take a look at Fig. 8.3.1 again and then at Fig. 8.3.2.

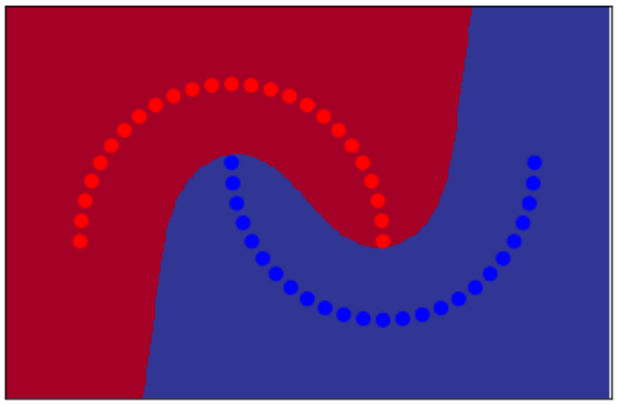

Figure 8.3.2: Kernel k-means can improve clustering a lot.

So far we had to choose the number of clusters k ourselves. In the next part, we will deal with an algorithm that somewhat automates this process.

8.4 Hierarchical Clustering

Hierarchical clustering is an approach that generates bigger clusters from smaller clusters, the hierarchy going from small to big. Bigger clusters are formed out of smaller clusters with the biggest possible cluster being the entire set of inputs $\{\mathbf{x}_i, ..., \mathbf{x_n}\}$, and the smallest possible clusters being the inputs $\mathbf{x}_i$ themselves.

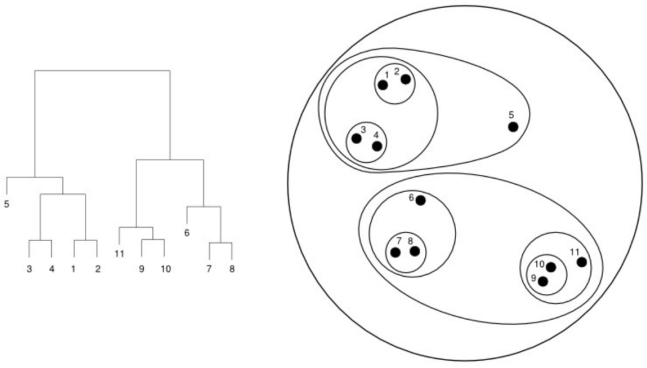

Figure 8.4.1: Example of hierarchical clustering of a set.

In Fig. 8.4.1, we see that the first clusters are assigned as $C_1 = \{1, 2\}, C_2 = \{3, 4\}, C_3 = \{7, 8\}, C_4 = \{9, 10\}, C_5 = \{5\}, C_6 = \{6\}, C_7 = \{11\}$. Notice how this corresponds to the leaves of the tree on the left side of Fig. 8.4.1.

Algorithm Hierarchical Clustering

Input: $\mathbf{X} = \mathbf{x_1}, ..., \mathbf{x_n} \in \mathbb{R}^d$.

1: **function** HIERARCHICAL CLUSTERING($\mathbf{X}$)
2: *Assign* each input $\mathbf{x}_i$ to a cluster.
3: **repeat**
4: Link the two clusters with *minimal distance*.
5: **until** Only one cluster is left.
6: **return** Tree describing cluster hierarchy.
7: **end function**

The cursively written "*assign*" in line 2 and "*minimal distance*" in line 4 merit further attention. For the assign step, we can define our own criteria for how big our initial clusters should be and by which criterion they should be assigned. We might be inclined to not just assign every point to its own cluster, since the whole point of clustering is to associate points with others. A simple approach would be to use k-means for some k to initialize our first clusters.

▼———————————————

Example 8.3. As we can see in Fig. 8.4.1, it would have been possible to put point 5 and 6 into one cluster; however, they are rather far apart, so there might have been a limit on how far points can be from each other before becoming ineligible to be clustered together.

▲———————————————

To calculate the minimal distance in line 4, there are three common options. Let $S_j \subseteq \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be the set of inputs contained in the $j-$th cluster.

$$\begin{aligned}
\text{Simple linkage: } d(i,j) &:= \min_{\mathbf{x}\in S_i, \tilde{\mathbf{x}}\in S_j} \|\mathbf{x}-\tilde{\mathbf{x}}\|^2 \\
\text{Average linkage: } \mathrm{d}(i,j) &:= \operatorname*{mean}_{\mathbf{x}\in S_i, \tilde{\mathbf{x}}\in S_j} \|\mathbf{x}-\tilde{\mathbf{x}}\|^2 \\
\text{Complete linkage: } d(i,j) &:= \max_{\mathbf{x}\in S_i, \tilde{\mathbf{x}}\in S_j} \|\mathbf{x}-\tilde{\mathbf{x}}\|^2.
\end{aligned}$$

Notice that all these expressions can be kernelized again. Which linkage we choose depends on our application. If we wanted to prevent "large" distances between points of two clusters, we would choose complete linkage, for instance.

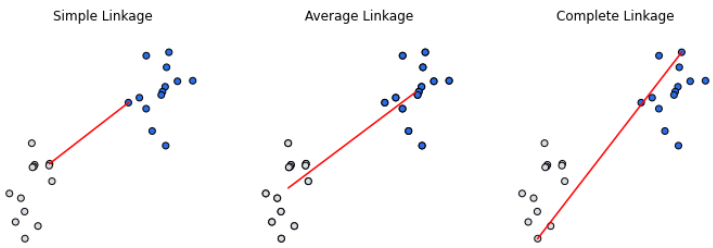

Figure 8.4.2: Geometric intuition behind the linkage types.

8.5 Exercises

1. Recall the $k-$nearest neighbor algorithm from Sheet 0.ipynb. Prove that $k-$nearest neighbor can be kernelized.

2. Solve Sheet 8.ipynb

9 Dimensionality Reduction

Definition 9.0.1 (Dimensionality Reduction)**.** With *dimensionality reduction*, we want to represent the data $\{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ in a lower dimensional space $\mathbb{R}^k$ with $k < d$ with as little loss of relevant information as possible.

dimensionality reduction

There are three main reasons why this is important: First of all, data can be visualized in two or three dimensions. Also, fewer dimensions mean less data to be stored and worked with, which is more resourceful and leads to faster computation. Finally, fewer dimensions decrease the risk of overfitting since this decreases the complexity of the data.

9.1 Linear Dimensionality Reduction

We are given $\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_n} \in \mathbb{R}^d$. We assume w.l.o.g that

$$\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i = 0.$$

This means that the data has been centered in a pre-processing step. We then have to find a $k-$dimensional subspace $\mathcal{U}_W = \operatorname{span}(\mathbf{w}_1, \ldots, \mathbf{w}_k)$ where $W = (\mathbf{w}_1, \ldots, \mathbf{w}_k) \in \mathbb{R}^{d \times k}$ is an orthonormal basis, such that projected onto that space via the projection

$$\Pi_{\mathcal{U}_W}(\mathbf{x}_i) := \arg\min_{\mathbf{x} \in \mathcal{U}_W} \|\mathbf{x} - \mathbf{x}_i\|^2, \quad i = 1, \ldots, n$$

is as "close" to the original data as possible. As a measure for closeness, we use the *average squared error* and pick the subspace $\mathcal{U}_{W^*}$ with

$$W^* = \arg\min_{W \in \mathbb{R}^{d \times k}} \quad \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - \Pi_{\mathcal{U}_W}(\mathbf{x}_i)\|^2.$$

Next, we will compute $\Pi_{\mathcal{U}_W}(\mathbf{x}_i)$ explicitly. Recall that orthonormality means $\langle \mathbf{w}_i, \mathbf{w}_j \rangle = 0$ for all $i \neq j$ and $\|\mathbf{w}_j\| = 1$. In the simplest case $k = 1$, we have $\mathcal{U}_W = \operatorname{span}(\mathbf{w_1})$. Then

$$\Pi_{\mathcal{U}_W}(\mathbf{x}_i) \overset{\text{def}}{=} \arg\min_{\mathbf{x} \in \mathcal{U}_W} \|\mathbf{x} - \mathbf{x}_i\|^2 = \operatorname*{arg\,min}_{\mathbf{x} \in \mathbb{R}^d : \exists \lambda \in \mathbb{R} \text{ with } \mathbf{x} = \lambda \mathbf{w}_1} \|\mathbf{x} - \mathbf{x}_i\|^2.$$

We then calculate the derivative of $f(\lambda) := \|\lambda \mathbf{w}_1 - \mathbf{x}_i\|^2$ and set it to zero:

$$\begin{aligned} f'(\lambda) &= \frac{d}{d\lambda} (\lambda \mathbf{w}_1 - \mathbf{x}_i)^T (\lambda \mathbf{w}_1 - \mathbf{x}_i) \\ &= \frac{d}{d\lambda} (\lambda \mathbf{w}_1)^T (\lambda \mathbf{w}_1) - 2 \frac{d}{d\lambda} \lambda \mathbf{w}_1^T \mathbf{x}_i \\ &= 2\lambda \mathbf{w}_1^T \mathbf{w}_1 - 2 \mathbf{w}_1^T \mathbf{x}_i \end{aligned}$$

Now, since $\mathbf{w}_1^T\mathbf{w}_1 = \|\mathbf{w}_1\|^2 = 1$, we get $\lambda^* = \mathbf{w}_1^T\mathbf{x}_i$ as the optimal value. Thus,

$$\Pi_{\mathcal{U}_W}(\mathbf{x}_i) = \lambda^*\mathbf{w}_1 = \mathbf{w}_1\lambda^* = \mathbf{w}_1\mathbf{w}_1^\top\mathbf{x}_i.$$

In the general case $k \geq 1$, we have $\mathcal{U}_W := \text{span}(\mathbf{w}_1 \ldots, \mathbf{w}_k)$ with $\mathbf{w}_i \perp \mathbf{w}_j$ for all $i \neq j$ and $\|\mathbf{w}_1\| = \cdots = \|\mathbf{w}_k\| = 1$. Hence:

$$\begin{aligned}\Pi_{\mathcal{U}_W}(\mathbf{x}_i) &\overset{\text{def.}}{=} \arg\min_{\mathbf{x}\in\mathcal{U}_W} \|\mathbf{x} - \mathbf{x}_i\|^2 \\ &= \arg\min_{\substack{\mathbf{x}\in\mathbb{R}^d : \exists \boldsymbol{\lambda}\in\mathbb{R}^k \\ \text{with } \mathbf{x}=\sum_{j=1}^k \lambda_j \mathbf{w}_j}} \|\mathbf{x} - \mathbf{x}_i\|^2 .\end{aligned}$$

As for $k = 1$, we can then set the derivative of $f(\lambda) := \left\|\sum_{j=1}^k \lambda_j\mathbf{w}_j - \mathbf{x}_i\right\|^2$ to zero, which reveals the optimum $\lambda_j^* = \mathbf{w}_j^\top\mathbf{x}_i$ for all $j = 1, \ldots, k$. Therefore:

$$\Pi_{\mathcal{U}_W}(\mathbf{x}_i) = \sum_{j=1}^k \lambda_j^*\mathbf{w}_j = \sum_{j=1}^k \mathbf{w}_j\lambda_j^* = \sum_{j=1}^k \mathbf{w}_j\mathbf{w}_j^\top\mathbf{x}_i = WW^\top\mathbf{x}_i.$$

Then the projected data can be written as:

$$\begin{aligned}\hat{X} &:= \left(\Pi_{\mathcal{U}}(\mathbf{x}_1), \ldots, \Pi_{\mathcal{U}_W}(\mathbf{x}_n)\right) \\ &= \left(WW^\top\mathbf{x}_1, \ldots, WW^\top\mathbf{x}_n\right) \\ &= WW^\top X.\end{aligned}$$

If we look at this with respect to the basis $\mathbf{w_1}, \ldots \mathbf{w_k}$, the coordinates of the projection of point $\mathbf{x_i}$ would clearly just be:

$$\widetilde{\mathbf{x}}_i := \begin{pmatrix} \mathbf{w}_1^\top\mathbf{x}_i \\ \vdots \\ \mathbf{w}_k^\top\mathbf{x}_i \end{pmatrix} = W^\top\mathbf{x}_i.$$

Repeating this for each data point would lead to the following matrix:

$$\tilde{X} := (\tilde{\mathbf{x}}_1, \ldots, \widetilde{\mathbf{x}}_n) = \left(W^\top\mathbf{x}_1, \ldots, W^\top\mathbf{x}_n\right) = W^\top X.$$

Now that we know what the projection looks like, our *PCA* problem becomes the following optimization problem: to a find matrix $W \in \mathbb{R}^{d\times k}$ that solves

$$\begin{aligned}&\arg\min_{W\in\mathbb{R}^{d\times k}} \sum_{i=1}^n \left\|x_i - WW^\top\mathbf{x}_i\right\|^2 \\ &\text{s.t.} \quad \mathbf{w}_i \perp \mathbf{w}_j \text{ for all } i \neq j \text{ and } \|\mathbf{w}_1\| = \cdots = \|\mathbf{w}_k\| = 1.\end{aligned}$$

Let us now look at how to solve this optimization problem. First recall that the projection is orthogonal. We will have the same situation as seen in Fig. 9.1.1.

According to the Pythagorean theorem, we get that the following equality holds:

$$\|\Pi_{\mathcal{U}_W}(\mathbf{x}_i)\|^2 + \|\mathbf{x}_i - \Pi_{\mathcal{U}_W}(\mathbf{x}_i)\|^2 = \|\mathbf{x}_i\|^2$$

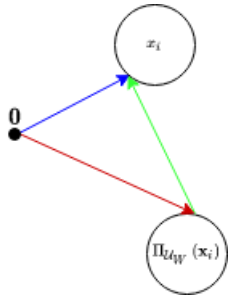

Figure 9.1.1: An illustration of the data point $\mathbf{x_i}$, its orthogonal projection, and the origin $\mathbf{0}$.

Observe the fact that minimizing $\|\mathbf{x}_i - \Pi_{\mathcal{U}_W}(\mathbf{x}_i)\|^2$ would lead to maximization of $\|\Pi_{\mathcal{U}_W}(\mathbf{x}_i)\|^2$. So we can safely exchange this in our optimization problem. Furthermore, we have that

$$\begin{aligned}\sum_{j=1}^{n} \|\Pi_{\mathcal{U}_w}(\mathbf{x}_i)\|^2 &= \sum_{i=1}^{n} \mathbf{x}_i^\top W W^\top \underbrace{W W^\top}_{=\sum_{j=1}^{k} \mathbf{w}_j \mathbf{w}_j^\top} \mathbf{x}_i \\ &= \sum_{j=1}^{k} \mathbf{w}_j^\top \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^\top \mathbf{w}_j \\ &= \sum_{j=1}^{k} \mathbf{w}_j^\top X X^\top \mathbf{w}_j.\end{aligned}$$

These observations lead us to the following theorem:

Theorem 9.1.1. *The PCA optimization problem can be equivalently written as*

$$\begin{aligned}W_* := \arg\max_{W \in \mathbb{R}^{d\times k}} &\sum_{j=1}^{k} \mathbf{w}_j^\top X X^\top \mathbf{w}_j \\ \text{s.t.} \quad &\mathbf{w}_i \perp \mathbf{w}_j \text{ for all } i \neq j \text{ and } \|\mathbf{w}_1\| = \cdots = \|\mathbf{w}_k\| = 1\end{aligned}$$

Definition 9.1.2 (Scatter Matrix)**.** The matrix $S_n := XX^\top$ is called the *scatter matrix*. **scatter matrix**

Theorem 9.1.3. *The optimal PCA solution $W_* = (\mathbf{w}_1^*, \ldots, \mathbf{w}_k^*)$ is given by the k largest eigenvectors of the (centered) scatter matrix S_n.*

Proof. Left as an exercise. □

Theorem 9.1.3 gives us an algorithm to solve our PCA optimization problem.

Algorithm PCA Linear Dimensionality Reduction

Input: parameter k, inputs $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in \mathbb{R}^{d \times n}$

1: Compute sample mean $\hat{\boldsymbol{\mu}} := \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$.
2: Center each input: $\mathbf{x}_i \leftarrow \mathbf{x}_i - \hat{\boldsymbol{\mu}}$ and update X.
3: Compute scatter matrix $S_n := XX^\top$.
4: Compute k largest eigenvalues of S_n with eigenvectors $W = (\mathbf{w}_1, \ldots, \mathbf{w}_k)$.
5: **return** dim.-reduced data: $\widetilde{X} = W^\top X \in \mathbb{R}^{k \times n}$ and $\hat{X} = WW^\top X \in \mathbb{R}^{d \times n}$.

9.2 Kernel PCA

Linear dimensionality reduction will struggle on a few inputs because of its inherent linearity. As seen in previous chapters in similar situations, we can apply kernelization. First, we apply the general representer theorem, that is, for each basis vector $\mathbf{w_j}^*$ of $W_* = (\mathbf{w}_1^*, \ldots, \mathbf{w}_k^*)$, there exist coefficients $\boldsymbol{\alpha}_j^* = \left(\alpha_{1j}^*, \ldots, \alpha_{nj}^*\right)^\top \in \mathbb{R}^n$ so that we can write

$$\mathbf{w}_j^* = \sum_{i=1}^{n} \alpha_{ij}^* \phi\left(\mathbf{x}_i\right).$$

Let us denote $\phi(X) := \left(\phi\left(\mathbf{x}_1\right), \ldots, \phi\left(\mathbf{x}_1\right)\right)$ and $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1^*, \ldots, \boldsymbol{\alpha}_k^*)$ then we get that

$$\mathbf{w_j}^* = \phi(X)\boldsymbol{\alpha}_j^*, \tag{9.2.1}$$

and the matrix in general can be written as a matrix multiplication more compactly as

$$W_* = \phi(X)\boldsymbol{\alpha^*}.$$

Finally, the data in the dimensionality-reduced basis will be

$$\widetilde{X} := W_*^\top \phi(X) = \left(\phi(X)\boldsymbol{\alpha^*}\right)^\top \phi(X) = \boldsymbol{\alpha^*}^\top \phi(X)^\top \phi(X) = \boldsymbol{\alpha^*}^\top K.$$

Let us use this insight to kernelize our PCA optimization problem. We have that $\mathbf{w}_j = \phi(X)\boldsymbol{\alpha}_j$ and thus:

$$\begin{aligned}\max_{W \in \mathbb{R}^{d \times k}} \sum_{j=1}^{k} \mathbf{w}_j^\top \phi(X)\phi(X)^\top \mathbf{w}_j &= \max_{\boldsymbol{\alpha} \in \mathbb{R}^{n \times k}} \sum_{j=1}^{k} \boldsymbol{\alpha}_j^\top \underbrace{\phi(X)^\top \phi(X)\phi(X)^\top \phi(X)}_{=K^2} \boldsymbol{\alpha}_j. \\ &= \max_{\boldsymbol{\alpha} \in \mathbb{R}^{n \times k}} \operatorname{tr}\left(\boldsymbol{\alpha}^\top K^2 \boldsymbol{\alpha}\right).\end{aligned}$$

We want the orthonormality constraints by:

$$\begin{aligned}&\forall z : ||\mathbf{w_z}|| = 1, \forall i \neq j : \mathbf{w_i}^\top \mathbf{w_j} = \mathbf{0} \\ \iff &\forall z : \left(\phi(X)\boldsymbol{\alpha}_z\right)^\top \left(\phi(X)\boldsymbol{\alpha}_z\right) = 1, \forall i \neq j : \left(\phi(X)\boldsymbol{\alpha}_i\right)^\top \left(\phi(X)\boldsymbol{\alpha}_j\right) = 0 \\ \iff &\forall z : \boldsymbol{\alpha}_z^\top K \boldsymbol{\alpha}_z = 1, \forall i \neq j : \boldsymbol{\alpha}_i^\top K \boldsymbol{\alpha}_j = 0 \\ \iff &\boldsymbol{\alpha}^\top K \boldsymbol{\alpha} = I.\end{aligned}$$

Ultimately, we get the following:

Theorem 9.2.1. *The objective function for kernel PCA (kPCA) is*

$$\begin{aligned} &\arg\max_{\boldsymbol{\alpha}\in\mathbb{R}^{n\times k}} \operatorname{tr}\left(\boldsymbol{\alpha}^\top K^2 \boldsymbol{\alpha}\right) \\ &\text{s.t. } \boldsymbol{\alpha}^\top K \boldsymbol{\alpha} = I. \end{aligned}$$

Definition 9.2.2 (Centered Kernel Matrix)**.** The centered kernel matrix is the kernel matrix computed on the data that has been centered in the feature space.

centered kernel matrix

Theorem 9.2.3. *The centered kernel matrix $\widetilde{K}$ can be computed from the (uncentered) kernel matrix K by:*

$$\widetilde{K} = \left(I - \frac{11^\top}{n}\right) K \left(I - \frac{11^\top}{n}\right).$$

Theorem 9.2.4. *The kPCA solution $\boldsymbol{\alpha}_* = (\boldsymbol{\alpha}_1^*, \ldots, \boldsymbol{\alpha}_k^*)$ can be computed by the k largest eigenvectors of the (centered) kernel matrix K.*

Proof. Left as an exercise. □

Both of the last two theorems pave the way for the following algorithm to solve the kPCA optimization problem.

Algorithm kPCA Kernelized Dimensionality Reduction

Input: parameter k, kernel matrix $K \in \mathbb{R}^{n\times n}$

1: Center the kernel matrix: $K \leftarrow \left(I - \frac{11^\top}{n}\right) K \left(I - \frac{11^\top}{n}\right)$.
2: Compute k largest eigenvectors $\boldsymbol{\alpha} = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k)$ of K.
3: Compute: $\widetilde{X} := \boldsymbol{\alpha}^\top K$.
4: **return** dim.-reduced data: $\widetilde{X} \in \mathbb{R}^{k\times n}$

As in the case of kernelized regression, linear kernels are useful if $d > n$, as the matrix we are looking the eigenvectors for, is smaller.

9.3 Autoencoders

Dimensionality reduction can also be achieved by means of neural networks, leading us to the concept of *autoencoders*. Let

$$f_W : \mathbb{R}^d \to \mathbb{R}^k$$

be a neural network called encoder and

$$g_{\tilde{W}} : \mathbb{R}^k \to \mathbb{R}^d,$$

be a neural netword called decoder. An autoencoder is defined as

$$\min_{W,\tilde{W}} \sum_{i=1}^{n} \left\| \mathbf{x}_i - g_{\tilde{W}} \left(f_W \left(\mathbf{x}_i \right) \right) \right\|^2 .$$

At first glance, this might not make sense: We are trying to minimize the loss between the original data point $\mathbf{x_i}$ and the decoded encoded version of $\mathbf{x_i}$. This can easily be achieved by not changing the data point at all, but we will not allow this behavior. A simple way to prohibit this is to enforce some bottleneck on a layer of neurons in our neural network as depicted in Fig. 9.3.1. On a high level, if done right, the output layer will resemble the input layer as

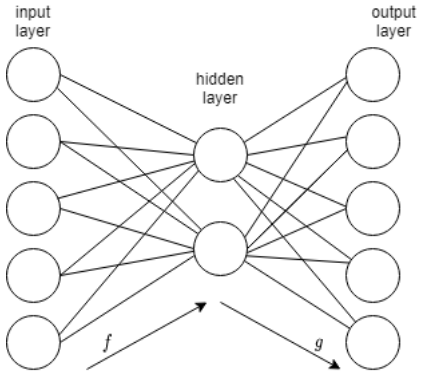

Figure 9.3.1: Illustration of an autoencoder neural network.

closely as possible but with missing details. In the subcase of linear encoders and decoders, the linear encoder will look like $f_W(\mathbf{x}) := W^\top \mathbf{x}$ and the linear decoder will look like $g_{\tilde{W}}(\mathbf{x}) := \tilde{W}^\top \mathbf{x}$. In the same fashion as in PCA, the data in our basis would be

$$\hat{X} := g_{\tilde{W}} \left(f_W(X) \right) \in \mathbb{R}^{d \times n},$$

and the data in the encoded basis would be

$$\widetilde{X} := f_W(X) \in \mathbb{R}^{k \times n}.$$

For the optimal choice of $\tilde{W}$, $\tilde{W} := W^\top$, we have that

$$\widetilde{X} = f_W(X) = W^\top X,$$

and that

$$\hat{X} = g_{\tilde{W}} \left(f_W(X) \right) = W W^\top X.$$

Observation 9.3.1. The solution of the linear autoencoder corresponds to the same solution as PCA. One thing to keep in mind is that the autoencoder does not constrain W to be orthonormal. The real power of autoencoders comes from their non-linearity. For more on autoencoders, have a look at [7]. On a final note: Observe that for $I(t) := t$ and $f(\mathbf{x}) = \left\| \phi(\mathbf{x}) - \mathbf{w}\mathbf{w}^\top \phi(\mathbf{x}) \right\|^2$, we can fit dimensionality reduction into our known unifying view

$$\min_{[W,]b,\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{n} I \left(f \left(\mathbf{x}_i \right) [, y_i] \right) + \left[+ \frac{1}{2} \sum_{l=1}^{L} \|W_l\|_{\text{Fro}}^2 \right] .$$

9.4 Exercises

1. Prove Theorem 9.2.3.
2. Let $X \in \mathbb{R}^{2\times n}$ be the data matrix, $S_n = (X - \hat{\mu})(X - \hat{\mu})^\top$ be the scatter matrix, and $S_n = \begin{bmatrix} 3 & -4 \\ -4 & 3 \end{bmatrix} \begin{bmatrix} 0.63 & 0 \\ 0 & 2.74 \end{bmatrix} \begin{bmatrix} -\frac{3}{7} & -\frac{4}{7} \\ -\frac{4}{7} & -\frac{3}{7} \end{bmatrix}$ be its diagonalization.
   a) What is the first principal component of the data according to PCA?
   b) Project the point $\mathrm{x} = \begin{bmatrix} 2 & -4 \end{bmatrix}^\top$ using PCA with $k = 1$ into the coordinate system with the basis made up of the first principal component.
   c) What happens to data points if we project them using PCA as in c) but with $k = 2$? Illustrate this using a sketch.
   d) What could the process from c) be used for?
   e) Assume that the S_n above is not centered and we do not have access to the data points. Can we use the same trick we used to center the kernel matrix to center S_n? Explain your answer.
3. Prove Theorem 9.2.4.

10 Decision Trees & Random Forests

In this chapter, we will consider tree-based learning, encompassing decision trees, bagging of decision trees, and random forests. Decision tree learning is one of the oldest non-linear machine learning methods. Bagging and random forests are methods for growing an ensemble of decision trees. The authors M. Fernandez-Delgado et al. [6] evaluated 179 classifiers from 17 families (such as neural networks, support vector machines, nearest-neighbors, and random forests), finding out that random forests and SVM with Gaussian kernel are the most likely the best classifiers. In practice, having a standard data set with some numerical features, random forests, or SVM with Gaussian kernels will likely be the best method for this data set. However, for example, in the case of image classification, where one needs a deep neural network, this result does not hold.

In the following chapter, we only look at numerical datasets. Those are given by d-dimensional data points $\mathbf{x} \in \mathbb{R}^d$ each equipped with a label $y \in \mathbb{R}$ in case of Regression problems or $y \in \{-1, +1\}$ in case of Classification problems. Given some training data, we construct our decision tree or random forest and want to predict labels for new data points.

10.1 Decision Trees

Definition 10.1.1 (Decision Trees)**.** A decision tree is a decision support tool from operations research that uses a tree-like graph or model of decisions and their possible consequences.

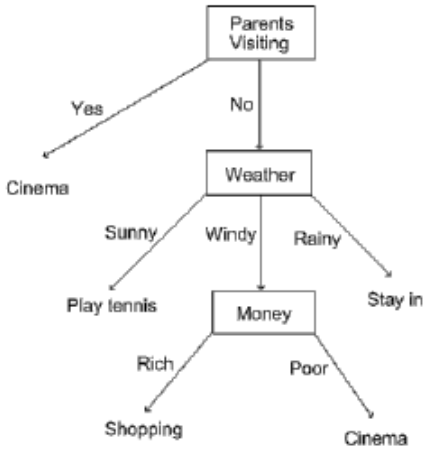

Figure 10.1.1: An example of a decision tree for "what to do at a normal day".

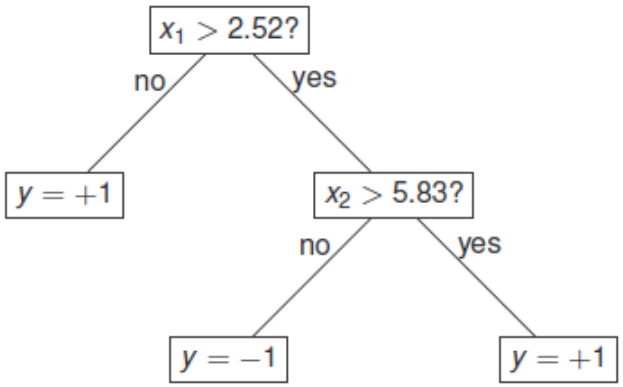

Figure 10.1.2: An example of a decision tree for predicting labels based on 2-dimensional data points $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$.

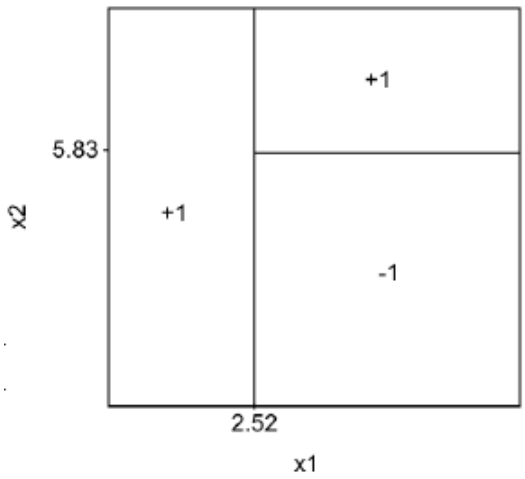

Figure 10.1.3: A visualisation of the induced decision regions of the decision tree in Figure 10.1.2.

The Figure 10.1.1 shows a general example of such a decision-supporting tree. An example of such a tree in the case of Machine Learning can be found in Figure 10.1.2. Here we have 2-dimensional data points $\mathbf{x} = (x_1, x_2)^T \in \mathbb{R}^2$. For each given data point, we follow the tree down to a terminal node and take this label as the predicted label for $\mathbf{x}$. The induced decision region of this decision tree is shown in Figure 10.1.3. We assign the appropriate label from point (x_1, x_2) to each given data point (x_1, x_2) in the figure. The first node '$x_1 > 2.52$' splits according to the x_1-coordinate at 2.52. For the left region, +1 is assigned; for the right region, the decision tree performs another check of node $x_2 > 5.83$. Note that such a visualization can only be done for 2-dimensional data.
Concluding, we now know what a decision tree looks like and how to apply a given tree to new data points. In the following, we consider constructing a decision tree that predicts well on a given dataset.

Assume we have d-dimensional data points. The general procedure is: We construct a complete binary tree with depth d and 2^d terminal nodes. After that, we delete some internal nodes that produce no substantial difference. So, firstly, we explain how to grow the tree and then how to reduce it again using pruning.

For growing the tree, three question arises.

1. For each internal node, which feature to take?
2. For each internal node, which threshold to take?
3. Which labels to use at the terminal nodes?

In other words: For a d-dimensional dataset an internal node is of the structure $x_j > t$. We have to specify which feature $j \in \{1, \dots, d\}$ (1. Question) and which threshold t (2. Question) we take. A terminal node has the structure $y = l$ for a label l, which we must choose (3. Question).

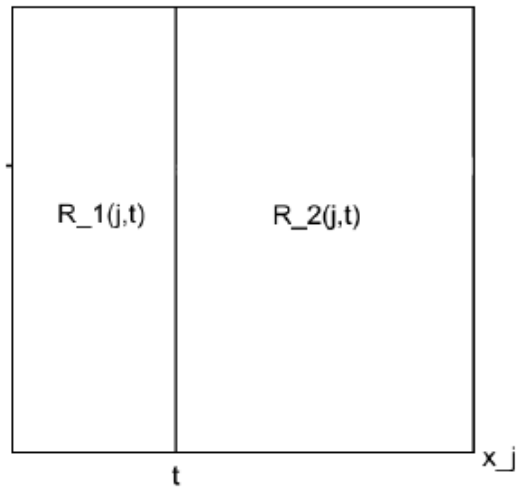

Figure 10.1.4: The two regions $R_1(j,t)$ and $R_2(j,t)$ for the internal node '$x_j > t$' given by j and t.

As Questions 1 and 2 correspond, we will answer them together.
But first, some notation. As before, we consider the node '$x_j > t$' with j as the feature that we split upon, t as the split threshold, and $R_1(j,t), R_2(j,t) \subset \mathbb{R}^d$ as the two induced regions under the internal node $x_j > t$. They are defined as following: $R_1(j,t) = \{\mathbf{x} \in \mathbb{R}^d \mid x_j < t\}$ and $R_2(j,t) = \{\mathbf{x} \in \mathbb{R}^d \mid x_j \geq t\}$. The two regions R_1 and R_2 are induced by splitting the j-th feature using the threshold t, where R_1 corresponds to everything under the left-hand subtree and R_2 to everything under the right-hand subtree as visualized in Figure 10.1.4. We will now look at the regression case, followed by the classification case.

For the regression case we have labels $l \in \mathbb{R}$ and we choose the feature j and threshold t by:

$$\arg\min_{j,t} \sum_{k=1}^{2} \min_{c \in \mathbb{R}} \sum_{i:\mathbf{x}_i \in R_k(j,t)} (y_i - c)^2$$

We choose j and t for each region $R_1(j,t)$ and $R_2(j,t)$, respectively, by finding an optimal value c such that the mean squared error of c and the labels y_i of the points $\mathbf{x}_i$ in $R_1(j,t)$ and $R_2(j,t)$, respectively, is minimal. One can show that the optimum c, here defined as $\hat{y}_{j,t,k}$, is the mean label of the training points falling into the appropriate region $R_1(j,t)$ or $R_2(j,t)$. Formalised this means, that for a region $R(j,t)$ the optimal c is given by:

$$c_k = \text{mean}\left(\{y_i : \mathbf{x}_i \in R_k(j,t)\}\right) =: \hat{y}_{j,t,k}$$

In summary, we choose the feature j and threshold t such that the average squared distance of labels of the points under a region to the mean label in that region is minimal.
To assign a label at a terminal node (3. Question), we just take the mean label of points in the region under this terminal. If this terminal node is the left child of the internal node given by j and t, we take $\hat{y}_{j,t,1}$ for the label.

For the Classification case, we have labels $l \in \{-1, +1\}$, and we want the predicted labels to be in the same range. Therefore, taking the mean label of points no longer works, so we use a different strategy.
Define

$$p_{j,t,k} = \frac{|\{i \mid \mathbf{x}_i \in R_k(j,t) \wedge y_i = +1\}|}{|\{i \mid \mathbf{x}_i \in R_k(j,t)\}|}$$

the fraction of instances in region $R_k(j,t)$ that are labeled $+1$. Hence, $1 - p_{j,t,k}$ is the fraction of instances in that region that are labeled -1. Now we choose the feature j and threshold t by

$$\arg\min_{j,t} \sum_{k=1}^{2} g\left(p_{j,t,k}\right)$$

where g is the *Gini impurity measure* given by $g(p) := 2p(1-p)$, see also Figure 10.1.5. Observe that the minimum of g is attained for very small or large values of p. Therefore, j and t are chosen such that $p_{j,t,k}$ is very small or large but preferably far from 0.5. $p = 0.5$ would mean we have as many points with positive labels as those with negative ones. Our algorithm searches for as much disparity as possible.
To assign a label at a terminal node (3. Question), we do a majority vote over the labels of the instances (that are the points $\mathbf{x} \in \mathbb{R}^d$ in the training data) falling into the region under the terminal node. If, for example, in the region, there are two points with a positive label and one with a negative one, this terminal node gets the label $+1$.

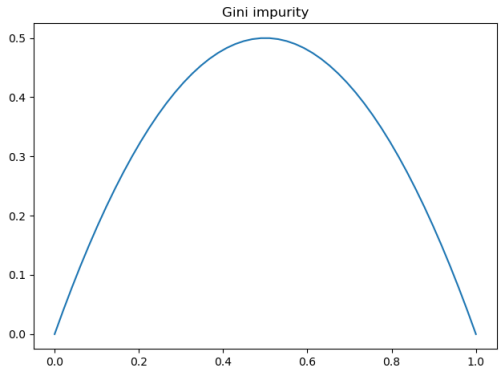

Figure 10.1.5: The Gini impurity measure $g(p) := 2p(1-p)$.

To reduce the tree size, we use *pruning*. Pruning is significant to remedy overfitting. A large, fully grown tree with 2^d terminal nodes is prone to overfit!

Definition 10.1.2 (Pruning)**.** Removing internal nodes that produce no substantial decrease in prediction accuracy as measured on a hold-out data set.

After training our tree on a training data set until it is fully grown, we start pruning it on another data set called the hold-out data set. Thereby, we successively remove nodes of the tree that do not decrease the prediction accuracy very much. For further information, one can read Hastie et al. [8] chapter 9, e.g., **cost-complexity pruning**. This chapter is based on chapters 9 and 15 of this book.

10.2 Random Forests

One major problem of the decision trees is their high variance: Often, a small change in data can result in a very different series of splits. For example, consider a feature that we split very late. We get a completely different tree if we split this feature at the beginning. This variance is bad for reproducibility, but we can use this to gain an advantage. The technique bagging averages many trees to reduce this variance. That means we repeat our method, decision tree, several times, each on a slightly different data set. Then, we combine the different results. In detail, this means:

1. We randomly draw each data set with replacement from the training data so that it has the same size as the original training set. (i.e., one data point could occur more than once)

2. Grow a tree on each data set.

3. In the regression case, we average the resulting prediction functions, and for the classification case, we perform a majority vote.

The resulting ensemble of trees is called a *forest*. In principle, this trick can be applied to any arbitrary machine learning algorithm (e.g., SVM, KRR, neural network). However, in practice, it works best with (small) decision trees because of their very high variance.

Definition 10.2.1 (Random Forests)**.** Random forests are pretty much the same as bagging of decision trees except for one difference: When growing a tree of the forest, for each split of an internal node in the tree, we randomly select a subset of $m < d$ many features and choose the optimal split feature j only among those m features.

So, for each internal node, we use a new random draw of m features, call the set of those features M, and we determine the optimal split feature j only among those m features: instead of $\arg\min_{j,t}$ we consider $\arg\min_{j \in M,t}$.
Typically, this subset of features is very small. If, for example, $d = 1000$, then a common choice for m is 40. Finally, we look at the spam data from [8] and compare the different methods in Figure 10.2.1 and see that random forests perform best with a lower testing error, already for less than 500 trees. Gradient Boosting, which performs best starting from 500 trees, is strongly related to random forests.

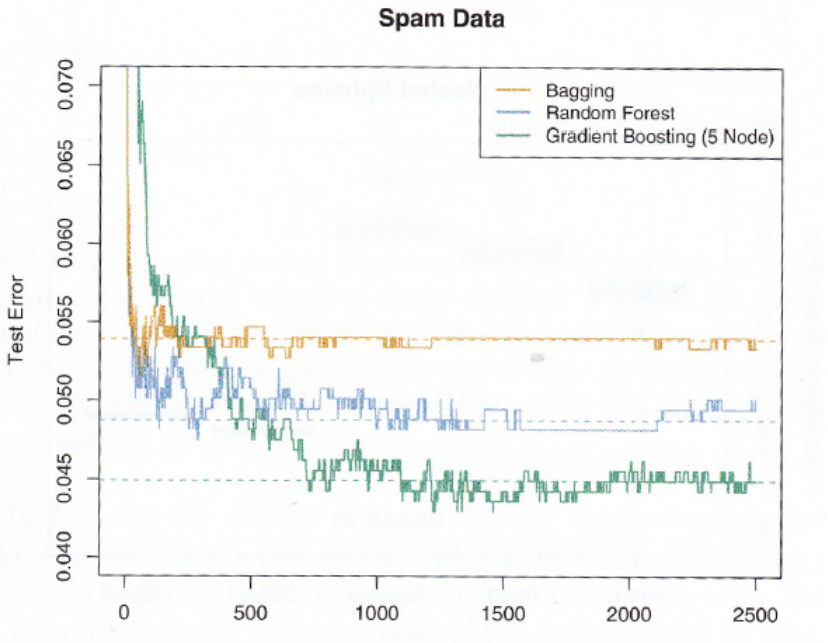

Figure 10.2.1: Comparison of bagging, random forests, and gradient boosting performance on the Spam dataset. The x axis gives the number of trees, and the y axis gives the testing error. Random forests perform best for less than 500 trees; starting from 500 trees, gradient boosting performs best. Figure taken from Hastie et al. [8].

Bibliography

[1] Léon Bottou, Frank E Curtis and Jorge Nocedal. 'Optimization methods for large-scale machine learning'. In: *Siam Review* 60.2 (2018), pp. 223–311.

[2] Stephen Boyd, Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[3] Leo Breiman. 'Random forests'. In: *Machine learning* 45.1 (2001), pp. 5–32.

[4] Corinna Cortes and Vladimir Vapnik. 'Support-vector networks'. In: *Machine learning* 20.3 (1995), pp. 273–297.

[5] Nello Cristianini, John Shawe-Taylor et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[6] Manuel Fernández-Delgado et al. 'Do we need hundreds of classifiers to solve real world classification problems?' In: *The journal of machine learning research* 15.1 (2014), pp. 3133–3181.

[7] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT Press Cambridge, 2016.

[8] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[9] Anil K Jain, M Narasimha Murty and Patrick J Flynn. 'Data clustering: a review'. In: *ACM Computing Surveys (CSUR)* 31.3 (1999), pp. 264–323.

[10] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. 'Imagenet classification with deep convolutional neural networks'. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[11] Kaare Brandt Petersen, Michael Syskind Pedersen et al. 'The matrix cookbook'. In: *Technical University of Denmark* 7.15 (2008), p. 510.

[12] Bernhard Schölkopf, Alexander J Smola, Francis Bach et al. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.

[13] Alex J Smola and Bernhard Schölkopf. 'A tutorial on support vector regression'. In: *Statistics and Computing* 14.3 (2004), pp. 199–222.

Index

0-1 Loss, 55

Artificial neural network, 42
Autoencoders, 85
Average linkage, 78

Back Propagation, 47
Bagging, 93
Batch normalization, 58
Box Cox Transformation, 66

Centered kernel matrix, 85
Classifier, 15
Clustering, 71
Complete linkage, 78
Convex function, 23
Convex Optimization Problem, 24
Convolutional filter, 48
Convolutional neural network, 47

Data augmentation, 59
Data matrix, 15
Decision Tree, 89
Deep regression, 68
Dimensionality reduction, 81

Eigenvalues, 10
Ensemble methods, 59

Feed-forward ANN, 43
Forward Propagation, 47

Gaussian RBF kernel, 35
General Representer Theorem, 38
Gini impurity , 92
Gradient, 12
Gradient Descent, 28

Hard-margin SVM, 19
Hessian Matrix, 12
Hierarchical clustering, 77
Hinge function, 29
Hyperplane, 9

K-means, 72
Kernel, 34
Kernel k-means, 75
Kernel PCA, 84
Kernel ridge regression, 68
Kernel trick, 34

Least-squares regression, 61
Leave-one-out cross-validation, 63
Linear Classifier, 16
Linear kernel, 35
Linear PCA, 84
Logistic function, 30
Logistic Regression, 30

Nearest centroid classifier, 16

Overfitting, 53

PCA, 82
Polynomial kernel, 35
Pooling, 48
Prediction function, 15
Pruning, 93

Random Forests, 94
Regularization, 56
Ridge regression, 62

Scalar Projection, 8
Scatter matrix, 83
Sigmoid function, 42
Signed Distance, 10
Simple linkage, 78
Soft-margin SVM, 20
Stochastic Gradient Descent, 30
Support vector, 20
Support vector regression, 69

Training, 15

Underfitting, 53