Statistical Machine Learning

Summary

Fabian Damken

March 6, 2022

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Contents

List of Figures

List of Tables

List of Algorithms

1. Introduction

Most of the content in this summary, the ideas, the underlying structure and the image ideas are taken from the lecture "Statistical Machine Learning" by Prof. Jan Peters. It is really just a *summary* of the contents of the lecture.

1.1. Examples

1.2. Classification vs. Regression

1.3. Paradigm

1.4. Key Challenges

1.4.1. Generalization

2. Fundamentals: Linear Algebra

2.1. Vectors

A *vector* is an ordered list of numbers that can be interpreted as an *arrow* in multidimensional spaces:

$$\boldsymbol{v} \in \mathbb{R}^n \quad \rightarrow \quad v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

Scalar Multiplication Multiplying a vector by a scalar is defined as multiplying each component by that scalar (let $\boldsymbol{v} \in \mathbb{R}^n$, $\lambda \in \mathbb{R}$):

$$\lambda \boldsymbol{v} = \lambda \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} \lambda v_1 \\ \vdots \\ \lambda v_n \end{bmatrix}$$

Per component:

$$(\lambda v)_i = \lambda v_i$$

Scalar multiplication is a linear operation.

Addition Adding two vectors is defined by adding the component of both vectors (thus, both vectors must have the same size; let $v, w \in \mathbb{R}^n$):

$$\boldsymbol{v} + \boldsymbol{w} = \begin{bmatrix} v_1 + w_1 \\ \vdots \\ v_n + w_n \end{bmatrix}$$

Per component:

$$(v + w)_i = v_i + w_i$$

Vector addition is both associative and commutative.

Vector Transpose The *transposed* version $\boldsymbol{v}^T$ of a vector $\boldsymbol{v} \in \mathbb{R}^n$ is a vector that was flipped around its main axis (thus, the new vector is a row/column vector if the initial vector as a column/row vector):

$$\begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}^T = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}$$

$$\begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}^T = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

Transposing a vector twice returns the initial vector

$$\boldsymbol{v} = \left(\boldsymbol{v}^T\right)^T$$

Linear Combination A *linear combination* of multiple vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \in \mathbb{R}^m$ is the addition of the scaled versions of them (scaled by scalars $\lambda_1, \cdots, \lambda_n \in \mathbb{R}$):

$$u = \lambda_1 v_1 + \cdots + \lambda_n v_n$$

If in a group of vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \in \mathbb{R}^m$, no vector can be represented as a linear combination of the others, they are called *linearly independent*.

Inner and Outer Product and Length The *inner product* of two vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$ is the sum of the product of the components and is denoted by a single dot $(\boldsymbol{v} \cdot \boldsymbol{w})$:

$$\boldsymbol{v} \cdot \boldsymbol{w} = \boldsymbol{v}^T \boldsymbol{w} = (v_1 w_1) + \cdots + (v_n w_n)$$

Therefore, the inner product gives a scalar value. In Cartesian coordinates, this is also called the *scalar product*.

The length $\|\boldsymbol{v}\|$ of a vector $\boldsymbol{v} \in \mathbb{R}^n$ is given as the euclidean norm:

$$\|\boldsymbol{v}\| = \left(\boldsymbol{v} \cdot \boldsymbol{v}\right)^{\frac{1}{2}} = \sqrt{v_1^2 + \cdots + v_n^2}$$

The *outer product* of two vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$ is defined analogue to the inner product, but with the transpose switched and is denoted by a cross in a circle $(\boldsymbol{v} \otimes \boldsymbol{w})$:

$$\boldsymbol{v} \otimes \boldsymbol{w} = \boldsymbol{v} \otimes \boldsymbol{w}^T \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \otimes \begin{bmatrix} w_1 & \cdots & w_n \end{bmatrix} = \begin{bmatrix} v_1 w_1 & \cdots & v_1 w_n \\ \vdots & \ddots & \vdots \\ v_n w_1 & \cdots & v_n w_n \end{bmatrix}$$

This is equivalent to matrix multiplication with two vectors and thus produces a matrix $\boldsymbol{v} \otimes \boldsymbol{w} \in \mathbb{R}^{n \times n}$.

Angles between Vectors The *angle* θ between two vectors $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}$ is given by:

$$\cos\theta = \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{\|\boldsymbol{v}\| \, \|\boldsymbol{w}\|} \quad \Longleftrightarrow \quad \theta = \arccos \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{\|\boldsymbol{v}\| \, \|\boldsymbol{w}\|}$$

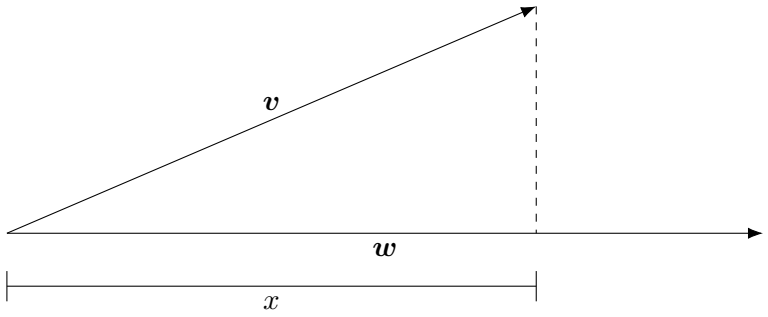

Figure 2.1.: Illustration of Vector Projection

Projections of Vectors A *projection* of a vector $\boldsymbol{v} \in \mathbb{R}^n$ onto a vector $\boldsymbol{w} \in \mathbb{R}^n$ results in a scalar value $x \in \mathbb{R}$ which equals the length of the adjacent w.r.t. to the angle between both vectors given that $\boldsymbol{v}$ is the hypotenuse and the third line is orthogonal to $\boldsymbol{w}$, see figure 2.1 for an illustration. Then this length is given as:

$$x = \|\boldsymbol{v}\| \cos\theta = \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{\|\boldsymbol{w}\|}$$

2.2. Matrices

A *matrix* is an ordered group of numbers that are ordered in two dimensions:

$$A \in \mathbb{R}^{n \times m} \quad \rightarrow \quad A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}$$

Scalar Multiplication Multiplying a matrix by a scalar is defined as multiplying each component by that scalar (let $A \in \mathbb{R}^{n \times m}$, $\lambda \in \mathbb{R}$):

$$\lambda A = \lambda \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} = \begin{bmatrix} \lambda a_{11} & \cdots & \lambda a_{1m} \\ \vdots & \ddots & \vdots \\ \lambda a_{n1} & \cdots & \lambda a_{nm} \end{bmatrix}$$

Per component:

$$(\lambda A)_{ij} = \lambda a_{ij}$$

Scalar multiplication is a linear operation.

Addition Adding two matrices is defined by adding the component of both matrices (thus, both matrices must have the same size; let $A, B \in \mathbb{R}^{n \times m}$):

$$A + B = \begin{bmatrix} a_{11} + b_{11} & \cdots & a_{1m} + b_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} + b_{n1} & \cdots & a_{nm} + b_{nm} \end{bmatrix}$$

Per component:

$$(A + B)_{ij} = a_{ij} + b_{ij}$$

Matrix addition is both associative and commutative.

Transpose The *transposed* version A^T of a matrix $A \in \mathbb{R}^{n \times m}$ is a matrix $A^T \in \mathbb{R}^{m \times n}$ that was flipped around its main axis:

$$\begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}^T = \begin{bmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1m} & \cdots & a_{nm} \end{bmatrix}$$

Transposing a matrix twice returns the initial matrix $A = \left(A^T\right)^T$

Matrix Multiplication The *matrix multiplication* of two matrices is only possible of the number of columns of the first matrix equals the number of rows of the second matrix (i.e. $A \in \mathbb{R}^{n \times m}$, $B \in \mathbb{R}^{m \times o}$). The resulting matrix then has the dimensions $AB \in \mathbb{R}^{n \times o}$. Matrix multiplication is defined as follows:

$$AB = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & \cdots & b_{1o} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mo} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + \cdots + a_{1m}b_{m1} & \cdots & a_{11}b_{1o} + \cdots + a_{1m}b_{mo} \\ \vdots & \ddots & \vdots \\ a_{n1}b_{11} + \cdots + a_{nm}b_{m1} & \cdots & a_{n1}b_{1o} + \cdots + a_{nm}b_{mo} \end{bmatrix}$$

Per component:

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}$$

Matrix multiplication is only associative and distributive w.r.t. matrix addition.

Inverse Let $I_n \in \mathbb{R}^{n \times n}$ be the identity matrix with all ones on the main diagonal and the rest zeros. If the dimension is clear, the n can be left out.

Using this definition, the *inverse* of a matrix $A \in \mathbb{R}^{n \times n}$ is defined as matrix $A^{-1} \in \mathbb{R}^{n \times n}$ that holds the following equation:

$$AA^{-1} = A^{-1}A = I_n$$

If such a matrix exists, the matrix A is called *regular* or *nonsingular*.

Pseudoinverse If a matrix $A \in \mathbb{R}^{n \times m}$ is not squared (i.e. $n \neq m$), there exists no inverse matrix. Instead, *Pseudoinverse Matrices* can be used to "invert" such a matrix. Left and right pseudoinverse are mutually exclusive, meaning that the one can only exist if the other does not (whilst neither have to exist).

The *left pseudoinverse* does only exist of the matrix has full column rank and is defined as:

$$A^{\#} = \left(A^T A\right)^{-1} A^T \quad \Longrightarrow \quad A^{\#} A = I_m$$

The *right pseudoinverse* does only exist if the matrix has full row rank and is defined as:

$$A^{\#} = J^T \left(J J^T\right)^{-1} \quad \Longrightarrow \quad A A^{\#} = I_n$$

Properties

Symmetricity A squared matrix $A \in \mathbb{R}^{n \times n}$ is *symmetric* iff $A^T = A$. This implies that:

- The inverse matrix A^{-1} is also symmetric.
- A can be decomposed into $A = QDQ^T$, where D is a diagonal matrix with all eigenvalues of A and Q is a matrix with all columns as the eigenvectors of A.

Definite Quadratic Form Let $A \in \mathbb{R}^{n \times n}$ be a squared matrix and let $\sigma(A)$ be its spectral. Then the matrix A is

$$\begin{cases} \text{positive definite} & \forall \lambda \in \sigma : \lambda > 0 & \Longleftrightarrow & \boldsymbol{x}^T A \boldsymbol{x} > 0 \\ \text{negative definite} & \forall \lambda \in \sigma : \lambda < 0 & \Longleftrightarrow & \boldsymbol{x}^T A \boldsymbol{x} < 0 \\ \text{positive semi-definite} & \forall \lambda \in \sigma : \lambda \geq 0 & \Longleftrightarrow & \boldsymbol{x}^T A \boldsymbol{x} \geq 0 \\ \text{positive semi-definite} & \forall \lambda \in \sigma : \lambda \leq 0 & \Longleftrightarrow & \boldsymbol{x}^T A \boldsymbol{x} \leq 0 \\ \text{indefinite} & \text{else} \end{cases}$$

for all vectors $\boldsymbol{x} \in \mathbb{R}^n$.

Regularity/Nonsingularity All of the following are equivalent w.r.t. a matrix $A \in \mathbb{R}^{n \times n}$:

- The matrix is regular.
- The matrix is nonsingular (or not singular).
- There exists a matrix $A^{-1} \in \mathbb{R}^{n \times n}$ with $A A^{-1} = A^{-1} A = I_n$.
- The determinant of the matrix is nonzero: $\det A \neq 0$.
- The matrix has full row rank.
- The matrix has full column rank.

2.3. Operations and Linear Transformations

Change of Basis

Linear Transformations

2.4. Eigenvalues and -vectors

Basis

Linear Transformations

2.5. Wrap-Up

- Vectors and matrices
- Operations on vectors and matrices
- Eigenvectors and -values
- Linear transformations

3. Fundamentals: Statistics

For a much more detailed introduction into the basic concepts (e.g. random variables), please lookup the summary of “Math 3: Stochastic and Statistics” (see https://www.dmken.com/cs), but notice that it is in German.

3.1. Random Variables

A *random variable* is a number that is determined by chance, draw according to a probability distribution.

3.2. Distributions

A probability distribution describes the probability that a random variable will equal a certain value (or lie in a certain range).

3.2.1. Uniform Distribution

All data/all values are equally likely within a bounded region R with size R.

$$p(x) = \frac{1}{R}$$

The distribution is plotted in figure 3.1.

3.2.2. Discrete Distributions

The random variables take *discrete values* (can be infinite, but countably infinity) and their probabilities sum up to 1:

$$\sum_i p(x_i) = 1$$

A discrete distribution is described by a *probability mass function* which is a normalized histogram.

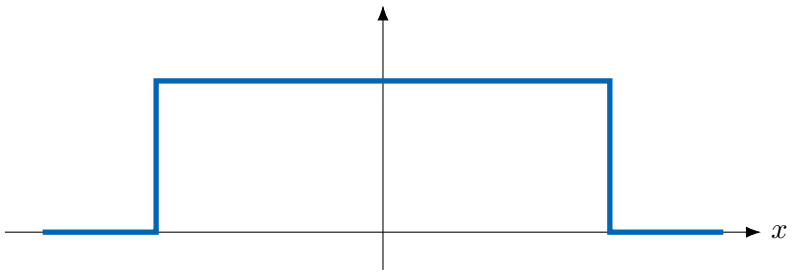

Figure 3.1.: Uniform Distribution

Bernoulli Distribution

A *Bernoulli random variable* only takes on two values, e.g. 0 or 1.

Parameters

μ The probability that the variable equals 1.

Properties

$$
\begin{aligned}
x &\in \{0, 1\} \\
p(x = 1 \mid \mu) &= \mu \\
\text{Bern}(x \mid \mu) &= y^x (1 - \mu)^{1-x} \\
\mathbb{E}(x) &= \mu \\
\text{Var}(x) &= \mu(1 - \mu)
\end{aligned}
$$

Binomial Distribution

Binomial variables are a sequence of N Bernoulli variables.

Parameters

μ The probability that one variable equals 1.

N The number of trials/samples.

Properties

$$
\begin{aligned}
\text{Bin}(m \mid N, \mu) &= \binom{N}{m} \mu^m (1 - \mu)^{N-m} \\
\mathbb{E}(m) &= N\mu \\
\text{Var}(m) &= N\mu(1 - \mu)
\end{aligned}
$$

See figure 3.2 for a visualization of $\text{Bin}(m \mid 10, 0.25)$.

Multinoulli Distribution

Multinoulli variables (also called *categorical variables*) are a generalization of Bernoulli variables where each variable can have multiple (namely K) outputs. The random variables is a vector with one-hot-encoding.

Parameters

$\boldsymbol{\mu}$ The entry μ_i defines the probability that the entry x_i equals 1.
All entries must be $\mu_i \geq 0$ and $\sum_{k=1}^{K} \mu_k = 1$.

K The number of classes/outcomes.

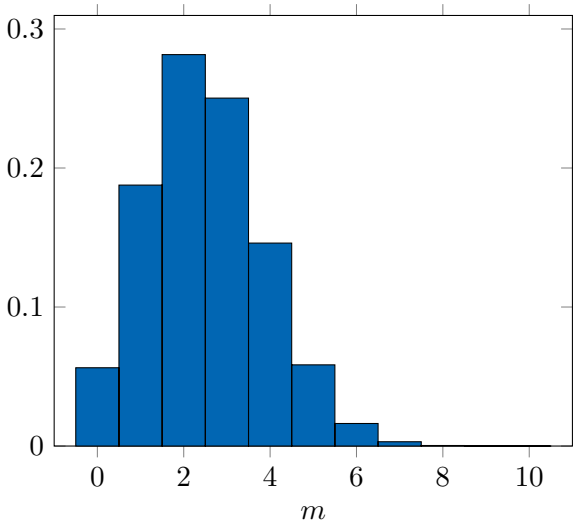

Figure 3.2.: Binomial Distribution $\text{Bin}(m \mid 10, 0.25)$

Properties

$$
\begin{aligned}
\boldsymbol{x} &= [0, 0, 1, 0, 0, 0]^T \\
p(x_i \mid \boldsymbol{\mu}) &= \mu_i \\
p(\boldsymbol{x} \mid \boldsymbol{\mu}) &= \prod_{k=1}^{K} \mu_k^{x_k} \\
\mathbb{E}(\boldsymbol{x} \mid \mu) &= \sum_{\boldsymbol{x}} p(\boldsymbol{x} \mid \boldsymbol{\mu}) \boldsymbol{x} = \boldsymbol{\mu}^T
\end{aligned}
$$

Multinomial Distribution

Multinomial variables are a sequence of N Multinoulli variables.

Parameters

$\boldsymbol{\mu}$ The entry μ_i defines the probability that, for one variable, the entry x_i equals 1. All entries must be $\mu_i \geq 0$ and $\sum_{k=1}^{K} \mu_k = 1$.

K The number of classes/outcomes.

N The number of trials/samples.

Properties

$$
\begin{aligned}
\text{Mult}(m_1, m_2, \cdots, m_K \mid \boldsymbol{\mu}, N) &= \binom{N}{m_1, m_2, \cdots, m_K} \prod_{k=1}^{K} \mu_k^{m_k} \\
\mathbb{E}(m_k) &= N\mu_k \\
\text{Var}(m_k) &= N\mu_k(1 - \mu_k) \\
\text{Cov}(m_j, m_k) &= -N\mu_j\mu_k
\end{aligned}
$$

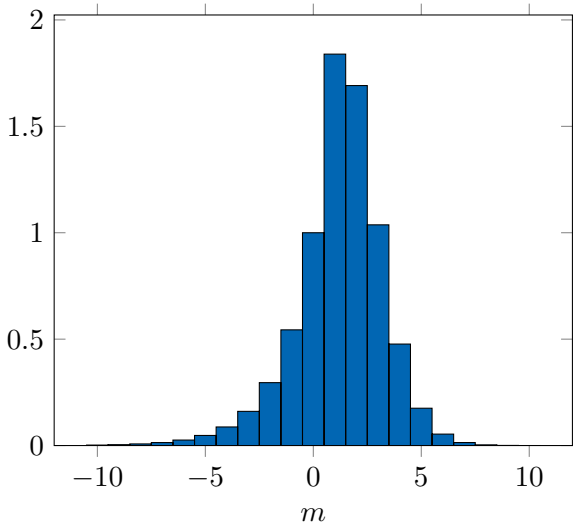

Figure 3.3.: Poisson Distribution $p(m \mid 5)$

Poisson Distribution

A *Poisson distribution* is a binomial distribution where the number of trials goes to infinity $N \to \infty$ and the success of each trial goes to zero $\mu \to 0$, s.t. $N\mu = \lambda$ is constant.

Parameters

λ Defines the expectation value and the variance at once.

Properties

$$p(m \mid \lambda) = \frac{\lambda^m}{m!} e^{-\lambda}$$
$$\mathbb{E}(m) = \lambda$$
$$\mathrm{Var}(m) = \lambda$$

See figure 3.3 for a visualization of $p(m \mid 5)$.

3.2.3. Continuous Distributions

The random variables take *discrete values* (infinite, can be uncountable) and their probability density function integrates to 1:

$$\int_{-\infty}^{+\infty} p(x)\,\mathrm{d}x = 1$$

A continuous distribution is described by a *probability density function* $p(x)$.

The probability that a random variable x falls into the interval (a, b) is

$$P(a < x < b) = \int_a^b p(x)\,\mathrm{d}x$$

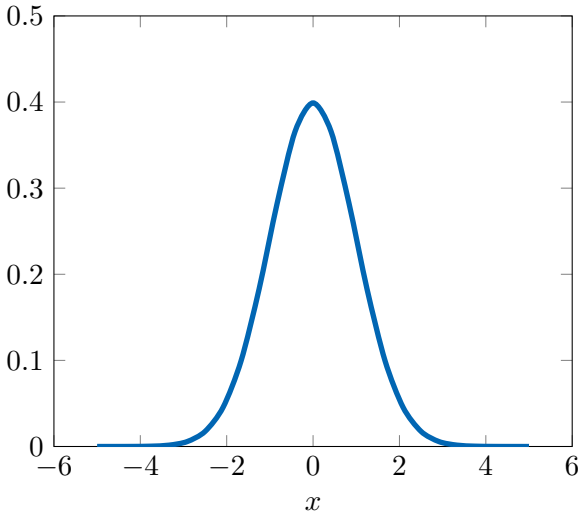

Figure 3.4.: Standard Gaussian Distribution $\mathcal{N}(x \mid 0, 1)$

Gaussian Distribution

Parameters

μ The expectation value.

σ^2 The variance.

Properties

$$p(x) = \mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}(x-\mu)^2 \right\}$$
$$\mathbb{E}(x) = \mu$$
$$\mathrm{Var}(x) = \sigma^2$$

A Gaussian distribution has more really useful properties:

- A Gaussian has soft tails, i.e. they fade away smoothly.
- Gaussians are often good models for data and provide analytical solutions.

See figure 3.4 for a visualization of $\mathcal{N}(x \mid 0, 1)$ (the standard Gaussian distribution).

3.2.4. Multivariate Gaussian Distribution

Gaussians can be applied to D-dimensional data $x_1, x_2, \cdots$ using multivariate Gaussian distributions.

Parameters

$\boldsymbol{\mu}$ A vector $\mu \in \mathbb{R}^D$ of the expectation values for each dimension.

Σ The *covariance matrix* containing the variance for each dimension on its main axis and the covariances on the other spaces. It is symmetric and defined as:

$$\Sigma = \begin{bmatrix} \mathrm{Var}(x_1) & \mathrm{Cov}(x_1, x_2) & \cdots & \mathrm{Cov}(x_1, x_D) \\ \mathrm{Cov}(x_2, x_1) & \mathrm{Var}(x_2) & \cdots & \mathrm{Cov}(x_2, x_D) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(x_D, x_1) & \mathrm{Cov}(x_D, x_2) & \cdots & \mathrm{Var}(x_D) \end{bmatrix}$$

Properties

$$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{2\pi}^D \sqrt{\det \Sigma}} \exp\left\{ -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \right\}$$
$$\mathbb{E}(\boldsymbol{x}) = \boldsymbol{\mu}$$
$$\mathrm{Var}(x_i) = \Sigma_{ii}$$

Geometry

Moments

3.2.5. Partitioned Gaussian Distributions

3.3. Central Limit Theorem

The distribution of the sum of N i.i.d. random variables becomes increasingly Gaussian as N increases. That is, with $N \to \infty$, it converges towards a Gaussian.

3.4. Probability Rules

Joint Distribution

$$p(x, y)$$

Marginal Distribution

$$p(y) = \int p(x, y)\, \mathrm{d}x$$

Conditional Distribution

$$p(y \mid x) = \frac{p(x, y)}{p(x)}$$

Probabilistic/Stochastic Independence

$$p(x, y) = p(x)p(y)$$

Chain Rule of Probabilities

$$\begin{aligned} p(x_1, \cdots, x_n) &= p(x_1 \,|\, x_2, \cdots, x_n) p(x_2, \cdots, x_n) \\ &= p(x_1 \,|\, x_2, \cdots, x_n) p(x_2 \,|\, x_3, \cdots, x_n) \cdots p(x_{n-1} \,|\, x_n) p(x_n) \end{aligned}$$

Bayes Rule

$$p(y \,|\, x) = \frac{p(x \,|\, y) p(y)}{p(x)}$$

- Posterior: $p(y \,|\, x)$
- Likelihood: $p(x \,|\, y)$
- Prior: $p(y)$
- Normalization Factor: $p(x) = \int p(x, y) \, \mathrm{d}y = \int p(x \,|\, y) p(y) \, \mathrm{d}y$

3.5. Expectation, Variance and Moments

3.5.1. Expectation

The *expectation* value of a random variable x with a distribution $p(x)$ is defined as:

$$\mathbb{E}_{x \sim p(x)}\big(f(x)\big) = \mathbb{E}_x\big(f(x)\big) = \mathbb{E}\big(f(x)\big) = \begin{cases} \sum_x f(x) \, p(x) & \text{for discrete distributions} \\ \int_x f(x) \, p(x) \, \mathrm{d}x & \text{for continious distributions} \end{cases}$$

This gives a similar formula for the *conditional expectation*:

$$\mathbb{E}_{x \sim p(x \,|\, y)}\big(f(x)\big) = \mathbb{E}_x\big(f(x)\big) = \mathbb{E}\big(f(x)\big) = \begin{cases} \sum_x f(x) \, p(x \,|\, y) & \text{for discrete distributions} \\ \int_x f(x) \, p(x \,|\, y) \, \mathrm{d}x & \text{for continious distributions} \end{cases}$$

With enough samples, the expectation value can be approximated using the arithmetic mean:

$$\mathbb{E}\big(f(x)\big) \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

Calculation Rules Let x, y be random variables and $\alpha \in \mathbb{R}$.

$$\begin{aligned} \mathbb{E}(\alpha x) &= \alpha \mathbb{E}(x) \\ \mathbb{E}(x + y) &= \mathbb{E}(x) + \mathbb{E}(y) \\ \mathbb{E}(xy) &= \mathbb{E}(x) \mathbb{E}(y) \end{aligned}$$

Equation 3.5.1 only holds if x and y are statistically independent.

3.5.2. Variance and Covariance

The *variance* measures the spread of the variable in relation to its mean:

$$\mathrm{Var}(x) = \mathbb{E}\Big(\big(x - \mathbb{E}(x)\big)^2\Big) = \mathbb{E}\big(x^2\big) - \big(\mathbb{E}(x)\big)^2$$

The *covariance* measures the correlation between two variables (how much the variables change together):

$$\mathrm{Cov}(x, y) = \mathbb{E}_{x,y}(xy) - \mathbb{E}_x(x)\mathbb{E}_y(y)$$
$$\mathrm{Cov}(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}(\boldsymbol{x}\boldsymbol{y}^T) - \mathbb{E}_{\boldsymbol{x}}(\boldsymbol{x})\mathbb{E}_{\boldsymbol{y}}(\boldsymbol{y}^T)$$

This gives the following very important rule (with $\boldsymbol{\mu}$ and Σ from the Gaussian):

$$\mathbb{E}(\boldsymbol{x}\boldsymbol{x}^T) = \boldsymbol{\mu}\boldsymbol{\mu}^T + \Sigma$$

3.5.3. Moments

A *moment* is defined as

$$m_n = E\big(x^n\big)$$

The *central moment* is defined as

$$cm_n = \mathbb{E}\big((x - \mu)^n\big)$$

which leads to another definition of the variance, skewness and kurtosis:

$\boldsymbol{cm}_2$ Variance (measure of spreading)

$\boldsymbol{cm}_3$ Skewness (measure of asymmetry)

$\boldsymbol{cm}_4$ Kurtosis (measure of heavy/light tailed-ness)

3.6. Exponential Family

The *exponential family* is a large class of distributions that are analytically interesting, because taking the log of them simplifies them a lot. All distributions of this family are unimodal with the following general form:

$$p(\boldsymbol{x} \mid \boldsymbol{\eta}) = h(\boldsymbol{x})g(\boldsymbol{\eta}) \exp\left\{\boldsymbol{\eta}^T \boldsymbol{u}(\boldsymbol{x})\right\}$$

where $\boldsymbol{\eta}$ is the natural parameter and

$$g(\boldsymbol{\eta}) \int_{-\infty}^{+\infty} h(\boldsymbol{x}) \exp\left\{\boldsymbol{\eta}^T \boldsymbol{u}(\boldsymbol{x})\right\} = 1$$

holds. g can be interpreted as a normalization to make this property hold true.

3.6.1. Example: Bernoulli Distribution

The Bernoulli distribution is part of the exponential family and decomposes as

$$\begin{aligned}\text{Bern}(x \mid \mu) &= \mu^x (1-\mu)^{1-x} \\ &= \exp\left\{x \ln(\mu) + (1-x)\ln(1-\mu)\right\} \\ &= (1-y)\exp\left\{\ln\left(\frac{\mu}{1-\mu}\right)x\right\}\end{aligned}$$

with the logistic sigmoid

$$\sigma(\eta) = \frac{1}{1+\exp(-\eta)}$$

and

$$\eta = \ln\left(\frac{\mu}{1-\mu}\right)$$

we can write the Bernoulli distribution as

$$p(x \mid \mu) = \sigma(-\eta)\exp(\eta x)$$

which, in the exponential family form, gives:

$$h(x) = 1 \qquad g(\eta) = \sigma(-\eta) \qquad u(x) = x$$

3.6.2. Example: Gaussian Distribution

The Gaussian distribution is part of the exponential family and decomposes as

$$\begin{aligned}\mathcal{N}(x \mid \mu, \sigma^2) &= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \\ &= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left\{-\frac{1}{2\sigma^2}x^2 + \frac{\mu}{\sigma^2}x - \frac{\mu^2}{2\sigma^2}\right\} \\ &= h(x)g(\boldsymbol{\eta})\exp\left\{\boldsymbol{\eta}^T \boldsymbol{u}(x)\right\}\end{aligned}$$

with

$$\eta = \begin{bmatrix}-\frac{1}{2\sigma^2} & \frac{\mu}{\sigma^2}\end{bmatrix}^T \qquad h(x) = 1 \qquad g(\boldsymbol{\eta}) = \sqrt{-\frac{\eta_1}{\pi}}\exp\left\{\frac{\eta_2^2}{4\eta_1}\right\} \qquad \boldsymbol{u}(x) = \begin{bmatrix}x^2 \\ x\end{bmatrix}$$

3.7. Information Theory and Entropy

Information theory is about how to represent information compactly (as few bits as possible) and therefore about compression.

This raises three questions:

- How to measure complexity?
- How to measure the “distance” between probability distributions?
- How to reconstruct data?

3.7.1. Information and Entropy

- Information is hiding in data.
- E.g. in the English alphabet, every letter has a different probability p_i of occurring.
- A lower probability indicates that the data point contains more information.
- The *average information*, called *entropy* can be calculated as

$$H(p) = -\sum_i p_i \log_2(p_i)$$

3.7.2. Kullback-Leibler Divergence

The *Kullback-Leibler Divergence* is a similarity measurement between probability distributions, defined by

$$\begin{aligned}\mathrm{KL}(p \,\|\, q) &= -\int p(x) \ln\big(q(x)\big)\,\mathrm{d}x - \left(-\int p(x) \ln\big(p(x)\big)\,\mathrm{d}x\right) \\ &= -\int p(x) \ln\left(\frac{q(x)}{p(x)}\right)\mathrm{d}x\end{aligned}$$

The KL divergence represents the average additional number of bits required to specify a symbol x, if the underlying probability distribution is the estimated $q(x)$ and not the true one $p(x)$.

Some properties:

- $\mathrm{KL}(p \,\|\, q) \neq \mathrm{KL}(q \,\|\, p)$ not a distance
- $\mathrm{KL}(p \,\|\, q) \geq 0$ non-negative distance
- $\big(\forall x : p(x) = q(x)\big) \implies \mathrm{KL}(p \,\|\, q) = 0$

There exist other metrics than KL, but KL is deeply connected to maximum likelihood estimation.

3.8. Wrap-Up

- Random variables (both continuous and discrete)
- Probability distributions
- Basic rules of probability theory
- Expectation and variance
- Gaussian distribution and its importance
- Information and entropy

4. Fundamentals: Optimization

> "All learning problems are essentially optimization problems on data" (Christopher G. Atkeson, Professor at CMU)

All machine learning problems are optimization problems in the form

$$\begin{aligned} &\min_{\theta} J(\theta, \mathcal{D}) \\ \text{s.t.} \quad &f(\theta, \mathcal{D}) = 0 \\ &g(\theta, \mathcal{D}) \geq 0 \end{aligned}$$

with parameters θ to enable learning, a data set $\mathcal{D}$ to learn from, a cost function $J(\theta, \mathcal{D})$ to measure the performance and equality and inequality constraints $f(\theta, \mathcal{D}) = 0$, $g(\theta, \mathcal{D}) \geq 0$.

4.1. Convexity

A set $C \subseteq \mathbb{R}^n$ is *convex* iff for all $\boldsymbol{x}, \boldsymbol{y} \in C$ and for all $\alpha \in [0, 1]$ the following holds:

$$\alpha\boldsymbol{x} + (a - \alpha)\boldsymbol{y} \in C$$

Intuition: Every point on a line drawn between two arbitrary points in space lie in the set itself. The set has no "bays".

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* iff for all $\boldsymbol{x}, \boldsymbol{y} \in \text{Domain}(f)$ and for all $\alpha \in [0, 1]$ the following holds:

$$f\big(\alpha\boldsymbol{x} + (1 - \alpha)\boldsymbol{y}\big) \leq \alpha f(\boldsymbol{x}) + (1 - \alpha) f(\boldsymbol{y})$$

Intuition: The drawn line between two arbitrary points on the function do not cross the function (they may touch, so linear functions are also convex).

If f is differentiable, it is convex iff for all $\boldsymbol{x}, \boldsymbol{y} \in \text{Domain}(f)$ the following holds:

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}}(\boldsymbol{y} - \boldsymbol{x})$$

If f is twice differentiable, it is convex iff for all $\boldsymbol{x} \in \text{Domain}(f)$ the following holds:

$$\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x}) \preceq 0$$

> **Warning:** Differentiability is not a condition for convexity!

Problem	Example Cost Functions	Resulting Method
Classification	$\min_\theta \sum_{i=1}^n \ln\Big(1+\exp\big(-y_i x_i^T\theta\big)\Big)$	Logistic Regression
	$\min_{\theta_1,\theta_2} \sum_{i=1}^n \Big(y_i - g\Big(\theta_2^T g\big(\theta_1^T x_i\big)\Big)\Big)^2$	Neural Network Classification
	$\min_\theta \lVert\theta\rVert^2 + C\sum_{i=1}^n \xi_i$ s.t. $\xi_i - \big(1-y_i x_i^T\theta\big) \geq 0,\ \xi_i \geq 0$	Support Vector Machines
Regression	$\min_\theta \sum_{i=1}^n \big(y_i - \phi(x_i)^T\theta\big)^2$	Linear Regression
	$\min_{\theta_1,\theta_2,\theta_3} \sum_{i=1}^n \Big(y_i - \theta_3^T g\Big(\theta_2^T g\big(\theta_1^T x_i\big)\Big)\Big)^2$	Neural Network Regression
Density Estimation	$\min_\theta \sum_{i=1}^n \ln\big(p(x_i \mid \theta)\big)$	General Formulation
Clustering	$\min_{\mu_1,\cdots,\mu_k} \sum_{j=1}^k \sum_{i\in C_j} \lVert x_i - \mu_i\rVert^2$	

Table 4.1.: Common Cost Functions

4.2. Cost Functions

An ideal cost function is convex. But most of the time, they are not...

4.2.1. Common Cost Functions

Table 4.1 lists common cost functions for classification, regression, density estimation and clustering.

4.3. Constrained/Unconstrained Optimization

The general form of a constrained optimization problem is

$$\begin{aligned} &\max_{\boldsymbol{\theta}} \ J(\boldsymbol{\theta}) \\ \text{s.t.} \quad &\boldsymbol{f}(\boldsymbol{\theta}) = 0 \\ &\boldsymbol{g}(\boldsymbol{\theta}) \geq 0 \end{aligned}$$

with a cost function $J(\boldsymbol{\theta})$, some equality constraints $\boldsymbol{f}(\boldsymbol{\theta})$ and inequality constraints $\boldsymbol{g}(\boldsymbol{\theta})$.

4.4. Lagrange Multipliers

With a constrained optimization problem in the general form, the *Lagrangian* is defined as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\epsilon}) = J(\theta) + \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{\theta}) + \boldsymbol{\mu}^T\big(\boldsymbol{g}(\boldsymbol{\theta} + \boldsymbol{\epsilon}^2)\big)$$

The coefficients $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are called *Lagrangian Multipliers*, the variables $\boldsymbol{\epsilon}$ are called *slack variables* and are used to convert the inequality constraints into equality constraints.

To solve the optimization problem, take the derivatives w.r.t. θ, λ and μ and set them to zero:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L} = 0 \qquad \nabla_{\boldsymbol{\lambda}}\mathcal{L} = 0 \qquad \nabla_{\boldsymbol{\mu}}\mathcal{L} = 0$$

If this results in any $\epsilon_i = 0$, the inequality constraint is called *active* and the solution lies on the edge of that constraint. To check whether the result really is a minima/maxima, take the second derivative of the cost

function w.r.t. θ, $\nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, and check whether the resulting Hessian is positive or negative definite, yielding that the found solution is a minima or maxima, respectively.

4.4.1. Dual Formulation

Given the so-called *primal problem*

$$\begin{aligned} \min_{\boldsymbol{\theta}} \; & J(\boldsymbol{\theta}) \\ \text{s.t.} \quad & \boldsymbol{f}(\boldsymbol{\theta}) = 0 \\ & \boldsymbol{g}(\boldsymbol{\theta}) \geq 0 \end{aligned}$$

with the Lagrangian

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\epsilon}) = J(\theta) + \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{\theta}) + \boldsymbol{\mu}^T \big(\boldsymbol{g}(\boldsymbol{\theta} + \boldsymbol{\epsilon}^2)\big)$$

the *dual problem* is

$$\begin{aligned} \max_{\boldsymbol{\lambda}, \boldsymbol{\mu}} \hat{\mathcal{L}}(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\epsilon}) = \min_{\boldsymbol{\theta}} \; & \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\epsilon}^2) \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geq \mathbf{0} \\ & \boldsymbol{\mu} \geq \mathbf{0} \end{aligned}$$

- If $\boldsymbol{\lambda}^*$ is the solution for the dual problem, then $\hat{\mathcal{L}}(\boldsymbol{\lambda}^*)$ is a *lower bound* for the primal problem due to two concepts:
  - *Minimax inequality:* For any function with two arguments $\phi(x, y)$, the maximin is less or equal to the minimax:

    $$\max_y \min_x \; \phi(x, y) \leq \min_x \max_y \; \phi(x, y)$$

  - *Weak duality:* The primal values are always greater or equal to the dual values:

    $$\min_{\boldsymbol{\theta}} \max_{\substack{\boldsymbol{\lambda} \geq \mathbf{0} \\ \boldsymbol{\mu} \geq \mathbf{0}}} \; \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \geq \max_{\substack{\boldsymbol{\lambda} \geq \mathbf{0} \\ \boldsymbol{\mu} \geq \mathbf{0}}} \min_{\boldsymbol{\theta}} \; \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu})$$

- In machine learning, the dual is often far more useful than the primal.
- That is because $\hat{\mathcal{L}}$ is a concave function and easy to optimize, even if J and the constraints may be nonconvex.
- Given some $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, the dual is an unconstrained problem.

4.4.2. Example

Given the following optimization problem (in the real numbers):

$$\begin{aligned} \arg\max_{x,y} \; & J(x, y) = x + y \\ \text{s.t.} \quad & x^2 + y^2 - 1 = 0 \\ & 2 - x \geq 0 \end{aligned}$$

the Lagrangian is written as

$$\mathcal{L}(x, y, \lambda, \mu, \epsilon) = x + y + \lambda(x^2 + y^2 - 1) + \mu(2 - x + \epsilon^2)$$

Take the derivatives:

$$\begin{aligned}
\nabla_x \mathcal{L} &= 1 + 2\lambda x - \mu \\
\nabla_y \mathcal{L} &= 1 + 2\lambda y \\
\nabla_\lambda \mathcal{L} &= x^2 + y^2 - 1 \\
\nabla_\mu \mathcal{L} &= 2 - x + \epsilon^2 \\
\nabla_\epsilon \mathcal{L} &= 2\mu\epsilon
\end{aligned}$$

Settings them to zero gives the insight that either $\mu = 0$ or $\epsilon = 0$ must be true. This must be done per case.

Case 1: $\mu = 0$ This yields the following equation system:

$$\begin{aligned}
0 &= 1 + 2\lambda x \\
0 &= 1 + 2\lambda y \\
0 &= x^2 + y^2 - 1 \\
0 &= 2 - x + \epsilon^2
\end{aligned}$$

with the solution $x = y = \pm\frac{1}{\sqrt{2}}$ and $\epsilon^2 = \frac{1}{\sqrt{2}} - 2$, so the inequality constraint is not active as the solution fulfills the equation $x < 2$.

Case 2: $\epsilon = 0$ This yields the following equation system:

$$\begin{aligned}
0 &= 1 + 2\lambda x - \mu \\
0 &= 1 + 2\lambda y \\
0 &= x^2 + y^2 - 1 \\
0 &= 2 - x
\end{aligned}$$

Which yields the following solutions:

$$\begin{aligned}
x_1 &= 2 & x_2 &= 2 \\
y_1 &= -i\sqrt{3} & y_2 &= i\sqrt{3}
\end{aligned}$$

As $\epsilon = 0$, the solution must lie on the edge of the inequality constraint, thus $x = 2$. As only real solutions where wanted, this solution can be discarded.

4.5. Numerical Optimization

For a lot of optimization problems, the solution cannot be computed analytically, so these have to be approximated using numerical optimization.

The performance of numerical methods can be measured with the following questions:

- Does the algorithm converge to the optimal solution?

- How many steps does it take to converge?
- Is the convergence smooth or bumpy?
- Does it work for all types of functions or just a special type (e.g. convex)?
- ...

Thus boils down to the following metrics that have to be taken into account:

- Number of iterations required
- Cost per iteration
- Memory footprint
- Region of convergence
- Is the cost function noisy?

The basic idea behind numerical optimization is to find a $\delta\theta$ with

$$J(\theta + \alpha\delta\theta) < J(\theta)$$

and to apply iterative updates rules like

$$\theta_{n+1} = \theta_n + \alpha\delta\theta$$

The key question is: How to find a good direction $\delta\theta$?

4.5.1. Learning Rate

There are two basic methods for finding the learning rate α:

- **Line Search**
  The learning rate is searched for each step with
  $$\alpha_n = \arg\min_{\alpha} J(\theta_n + \alpha\delta\theta_n)$$
- **Constant Learning Rate**
  The learning rate $\alpha = \text{const}$ is just fixed and not dynamically determined.
- **Adaptive Learning Rate**
  The learning rate α is changed in each step according to some rules. Note that line search is kind of an adaptive learning rate that does not take previous learning rates into account. See 13.6.2 for more information about adaptive learning rates.

4.5.2. Test Functions

For testing the performance of the methods, well-known functions with interesting properties are used.

Quadratic Function

$$J(\boldsymbol{\theta}) = (\theta_1 - 5)^2 + (\theta_1 - 5)(\theta_2 - 5) + (\theta_2 - 5)^2$$

The quadratic function is plotted in 4.1.

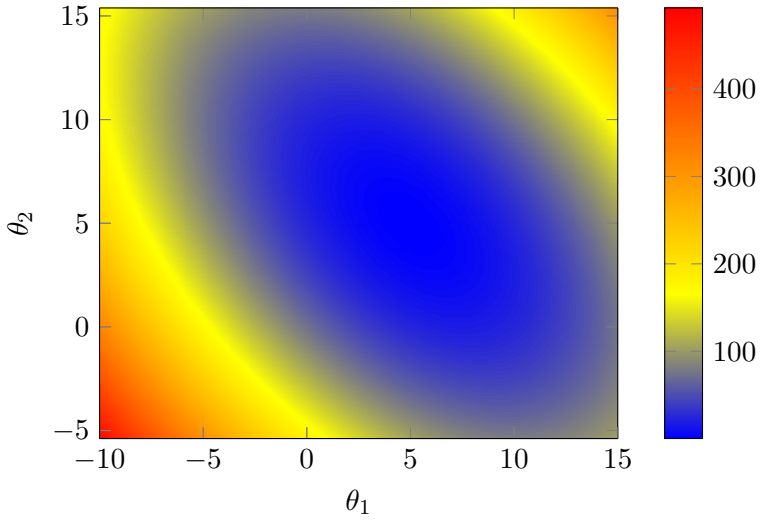

Figure 4.1.: Quadratic Function

Rosenbrock Function

$$J(\boldsymbol{\theta}) = \left(\theta_2 - \theta_1^2\right)^2 + 0.01(1 - \theta_1)^2$$

The Rosenbrock function is plotted in 4.2.

4.5.3. Axial Iteration

Alternate minimization for each axis.

4.5.4. Steepest Descent

- Also called *gradient descent*.
- Move in the direction of the gradient $\nabla J(\boldsymbol{\theta})$.
- The gradient is perpendicular to the contour lines and the next gradient is always orthogonal to the previous step direction after line minimization.
- As the gradient points into the direction of the maximum, the gradient has to be added for maximization and subtracted for minimization (with a positive step size).
- Problem: The gradient walks down in a zig-zag line that is very inefficient.

Algorithm 1 shows gradient descent in its basic version with a fixed learning rate α and the initialization vector $\mathbf{0}$. The algorithm terminates after n iterations. For maximization, the minus in line 3 has to be changed to a plus.

Test Functions

The plot of gradient descent working on the Rosenbrock function is plotted in figure 4.3, on the Quadratic function in figure 4.4.

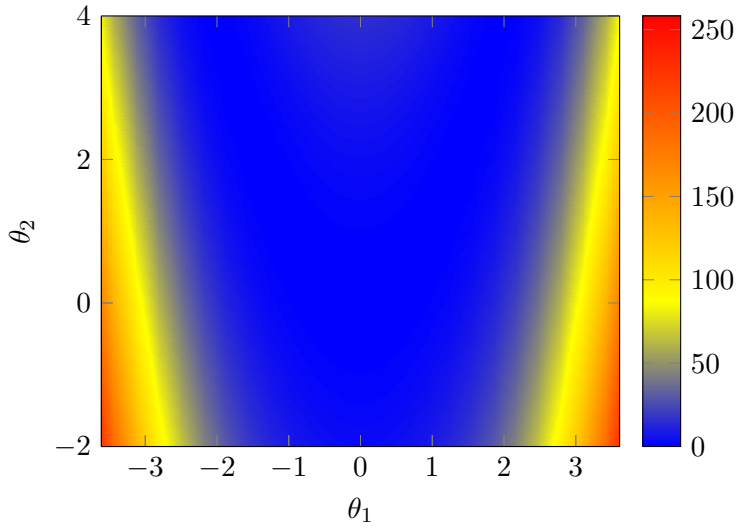

Figure 4.2.: Rosenbrock Function

Algorithm 1: Steepest Descent (Minimization)

$\boldsymbol{\theta}^{(1)} \leftarrow \mathbf{0}$
for $i = 1, \cdots, n$ **do**
  $\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
return $\boldsymbol{\theta}^{(n+1)}$

4.5.5. Newtons Method

Newtons method uses the first-order Taylor approximation

$$J(\boldsymbol{\theta} + \delta\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta} + \frac{1}{2}\delta\boldsymbol{\theta}^T \nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \delta\boldsymbol{\theta}$$
$$= c + \boldsymbol{g}^T \delta\boldsymbol{\theta} + \frac{1}{2}\delta\boldsymbol{\theta}^T H \delta\boldsymbol{\theta} =: \tilde{J}(\delta\boldsymbol{\theta})$$

where $\boldsymbol{g}$ is the Jacobian and H is the Hessian.

Minimizing this approximation yields the solution

$$\delta\boldsymbol{\theta} = -H^{-1}\boldsymbol{g}$$

- Has quadratic convergence and finds the optimal solution for quadratic functions in one step (in the case of a learning rate $\alpha = 1$).
- If the Hessian is positive definite, $\delta\boldsymbol{\theta}$ is guaranteed to point downhill.
- If the Hessian just equals the identity matrix $H = I$, this method is equal to steepest descent.
- Problem: Computing the Hessian at every iteration is extremely expensive and often not feasible (the inversion can be removed by transforming it into a linear equation system).

Algorithm 2 shows Newtons method for minimization.

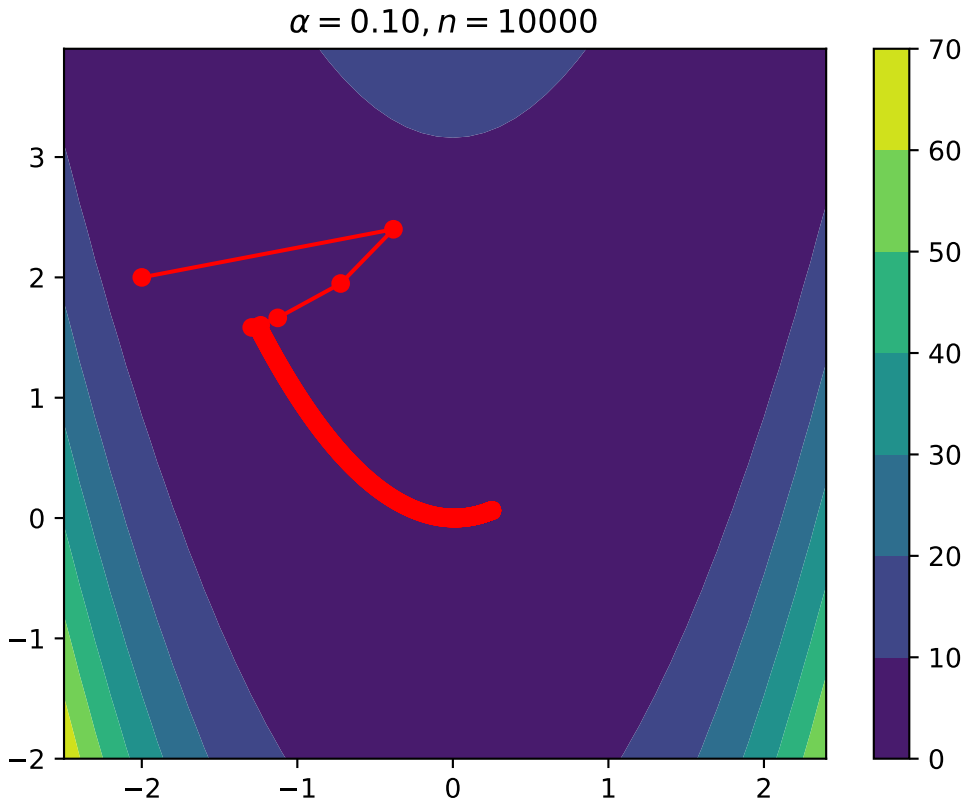

Figure 4.3.: Steepest Descent on Rosenbrock

Algorithm 2: Newtons Method (Minimization)

$\boldsymbol{\theta}^{(1)} \leftarrow \mathbf{0}$
for $i = 1, \cdots, n$ **do**
  $\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \alpha H^{-1}\left(\boldsymbol{\theta}^{(i)}\right)\boldsymbol{g}\left(\boldsymbol{\theta}^{(i)}\right)$
return $\boldsymbol{\theta}^{(n+1)}$

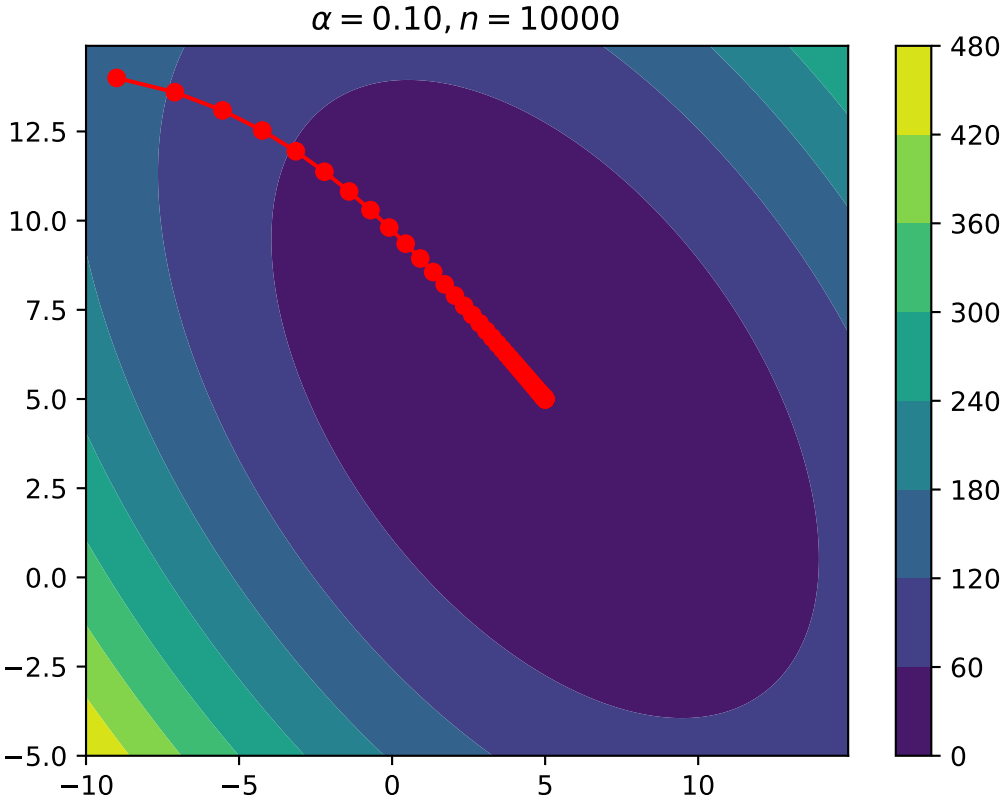

Figure 4.4.: Steepest Descent on Quadratic

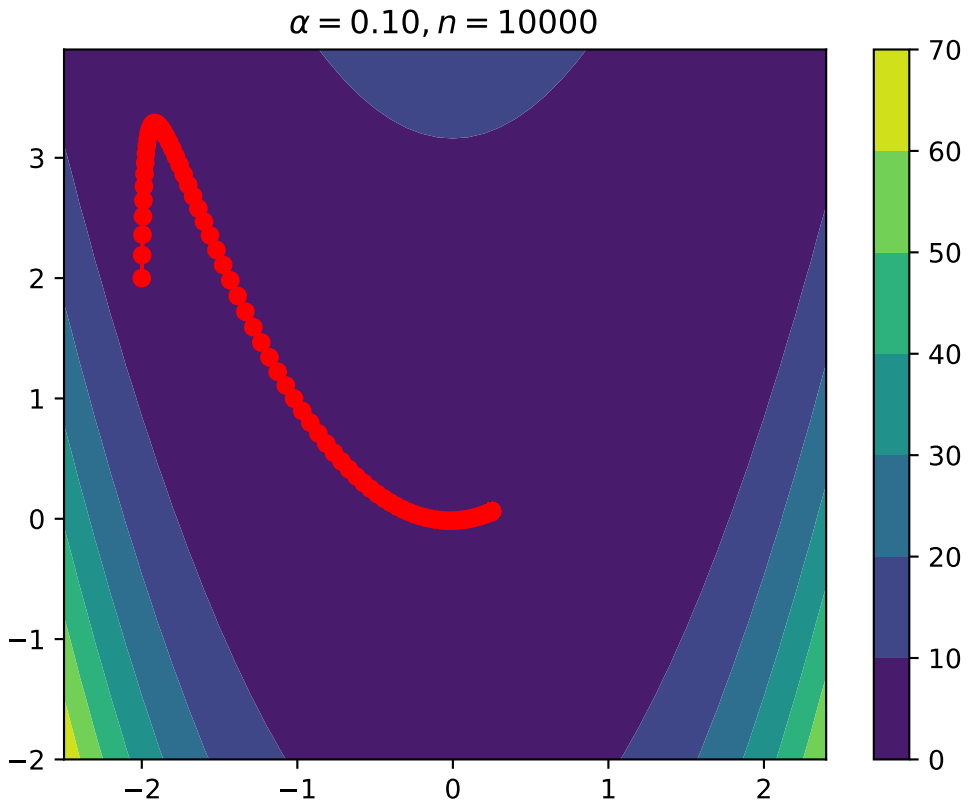

Figure 4.5.: Newtons Method on Rosenbrock

Test Functions

The plot of newtons method working on the Rosenbrock function is plotted in figure 4.5, on the Quadratic function in figure 4.6.

4.5.6. Quasi-Newton Method (BFGS)

- Approximate the Hessian using
  - Hessians change slowly,
  - Hessians are symmetric and
  - the derivatives interpolate.

This gives the following optimization problem:

$$\begin{aligned} & \min \|H - H_n\| \\ \text{s.t.} \quad & H = H^T \\ & H\left(\boldsymbol{\theta}^{(n+1)} - \boldsymbol{\theta}^{(n)}\right) = g\left(\boldsymbol{\theta}^{(n)}\right) - \boldsymbol{g}\left(\boldsymbol{\theta}^{(n)}\right) \end{aligned}$$

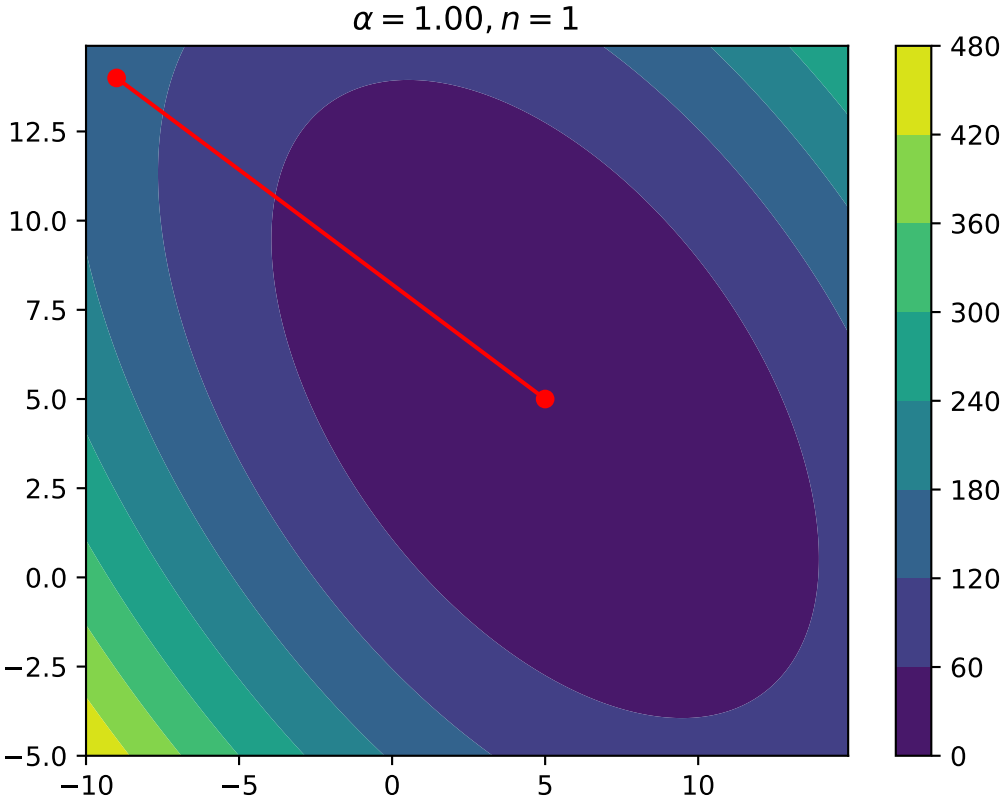

Figure 4.6.: Newtons Method on Quadratic

Using this, the Hessian can be computer iteratively:

$$
\begin{aligned}
H_{n+1}^{-1} &= \left(I - \frac{\boldsymbol{s}_n \boldsymbol{y}_n^T}{\boldsymbol{y}_n \cdot \boldsymbol{s}_n}\right) H_n^{-1} \left(I - \frac{\boldsymbol{y}_n \boldsymbol{s}_n^T}{\boldsymbol{y}_n \cdot \boldsymbol{s}_n}\right) + \frac{\boldsymbol{s}_n \boldsymbol{s}_n^T}{\boldsymbol{y}_n \cdot \boldsymbol{s}_n} \\
\boldsymbol{y}_n &= \boldsymbol{g}\big(\boldsymbol{\theta}^{(n+1)}\big) - \boldsymbol{g}\big(\boldsymbol{\theta}\big) \\
\boldsymbol{s}_n &= \boldsymbol{\theta}^{(n+1)} - \boldsymbol{\theta}^{(n)}
\end{aligned}
$$

- The first step in the algorithm can be slightly off due to initialization errors.
- For great dimensions, BFGS is preferred over the others as it does not require to compute the Hessian.

Algorithm 3 shows BFGS for minimization.

Algorithm 3: Quasi-Newton-Method, BFGS (Minimization)

$\boldsymbol{\theta}^{(1)} \leftarrow \mathbf{0}$
$H_1^{-1} \leftarrow 0$
for $i = 1, \cdots, n$ **do**
  $\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \alpha H_i^{-1}\big(\boldsymbol{\theta}^{(i)}\big)\boldsymbol{g}\big(\boldsymbol{\theta}^{(i)}\big)$
  $\boldsymbol{y} \leftarrow \boldsymbol{g}\big(\boldsymbol{\theta}^{(n+1)}\big) - \boldsymbol{g}\big(\boldsymbol{\theta}\big)$
  $\boldsymbol{s} \leftarrow \boldsymbol{\theta}^{(n+1)} - \boldsymbol{\theta}^{(n)}$
  $H_{i+1}^{-1} = \left(I - \frac{\boldsymbol{s}\boldsymbol{y}^T}{\boldsymbol{y}\cdot\boldsymbol{s}}\right) H_i^{-1} \left(I - \frac{\boldsymbol{y}\boldsymbol{s}^T}{\boldsymbol{y}\cdot\boldsymbol{s}}\right) + \frac{\boldsymbol{s}\boldsymbol{s}^T}{\boldsymbol{y}\cdot\boldsymbol{s}}$
return $\boldsymbol{\theta}^{(n+1)}$

Test Functions

The plot of BFGS working on the Rosenbrock function is plotted in figure 4.7, on the Quadratic function in figure 4.8.

4.5.7. Conjugate Gradient (CG)

- *Conjugate gradient* choose the descent direction $\delta\boldsymbol{\theta}$ such that it is guaranteed to reach the minimum in a finite number of steps.
- Each $\delta\boldsymbol{\theta}$ is chosen to conjugate all previous search directions w.r.t. the Hessian.
- The resulting search directions are mutually linearly independent.
- This avoid undoing previously done work.
- An N-dimensional quadratic function can be minimized in at most N CG steps.
- Also avoids computing the Hessian!
- $\delta\boldsymbol{\theta}^{(n)}$ is calculated using only $\delta\boldsymbol{\theta}^{(n-1)}$, $\nabla_{\boldsymbol{\theta}}\big(\boldsymbol{\theta}^{(n)}\big)$ and $\nabla_{\boldsymbol{\theta}}\big(\boldsymbol{\theta}^{(n-1)}\big)$:

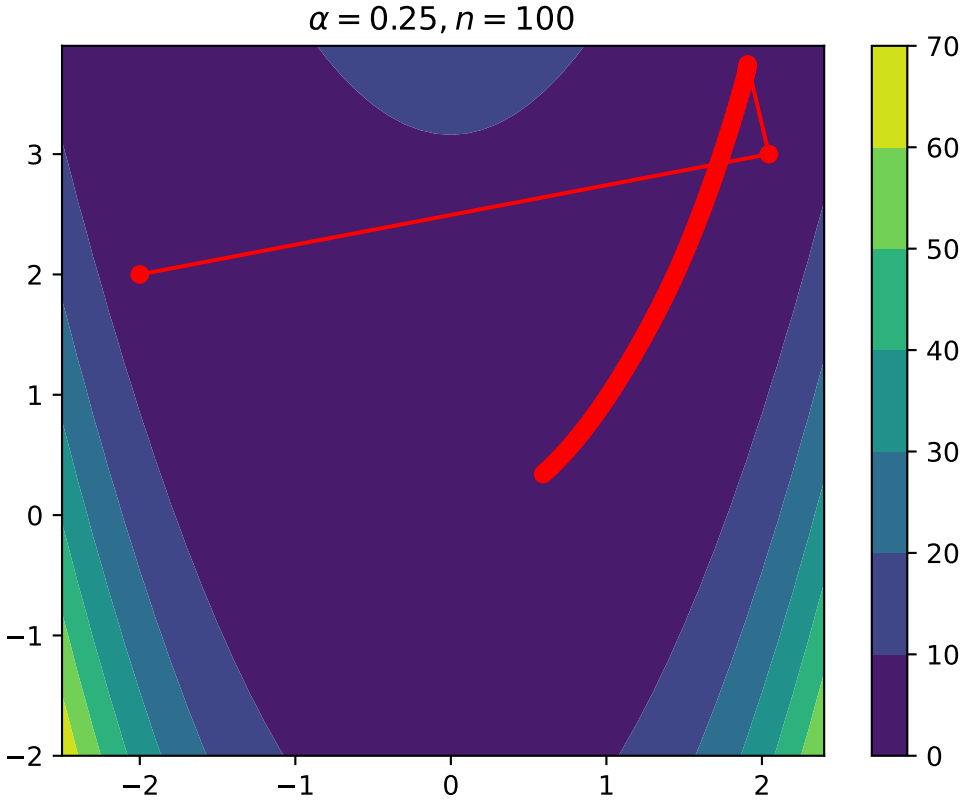

Figure 4.7.: BFGS on Rosenbrock

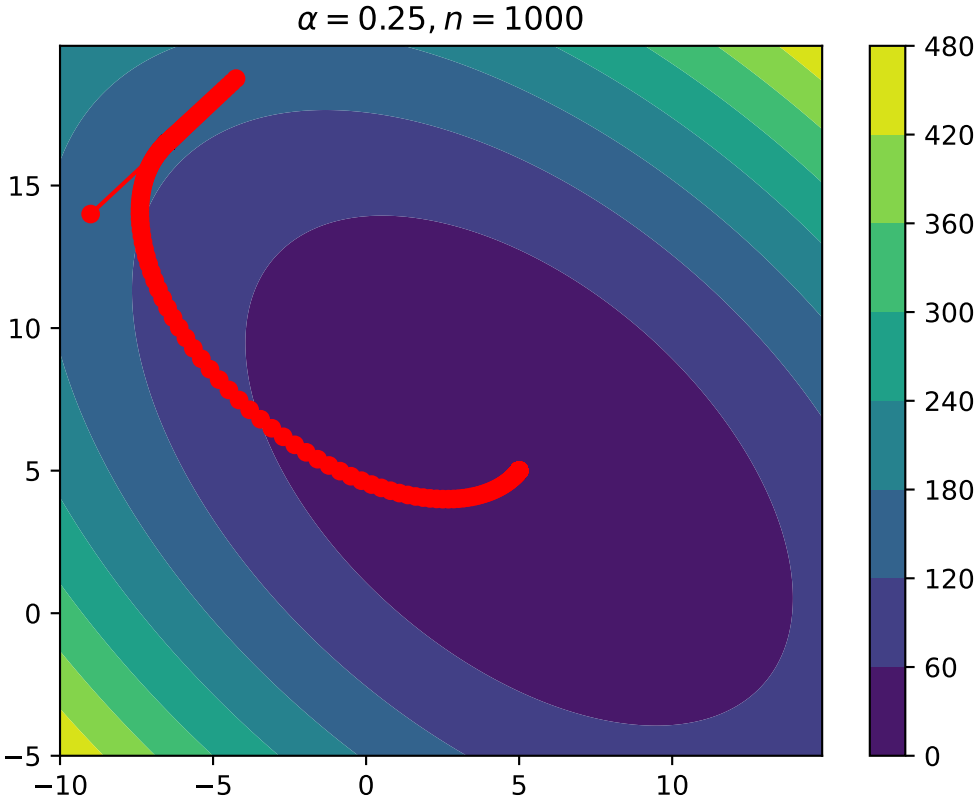

Figure 4.8.: BFGS on Quadratic

$$\delta\boldsymbol{\theta}^{(n)} = \nabla_{\boldsymbol{\theta}} J\big(\boldsymbol{\theta}^{(n)}\big) + \frac{\Big|\nabla_{\boldsymbol{\theta}} J\big(\boldsymbol{\theta}^{(n)}\big)\Big|^2}{\Big|\nabla_{\boldsymbol{\theta}} J\big(\boldsymbol{\theta}^{(n-1)}\big)\Big|^2}\delta\boldsymbol{\theta}^{(n-1)}$$

Algorithm 4 shows CG for minimization.

Algorithm 4: Conjugate Gradients (Minimization)

$\boldsymbol{\theta}^{(0)} \leftarrow \mathbf{0}$
$\boldsymbol{\theta}^{(1)} \leftarrow \mathbf{0}$
$\delta\boldsymbol{\theta}^{(1)} \leftarrow \mathbf{0}$
for $i = 1, \cdots, n$ **do**
  $\delta\boldsymbol{\theta}^{(i+1)} \leftarrow \nabla_{\boldsymbol{\theta}} J\big(\boldsymbol{\theta}^{(i)}\big) + \frac{\Big|\nabla_{\boldsymbol{\theta}} J\big(\boldsymbol{\theta}^{(i)}\big)\Big|^2}{\Big|\nabla_{\boldsymbol{\theta}} J\big(\boldsymbol{\theta}^{(i-1)}\big)\Big|^2}\delta\boldsymbol{\theta}^{(i-1)}$
  $\boldsymbol{\theta}^{(i+1)} \leftarrow \boldsymbol{\theta}^{(i)} - \alpha\,\delta\boldsymbol{\theta}^{(i+1)}$
return $\boldsymbol{\theta}^{(n+1)}$

Test Functions

The plot of CG working on the Rosenbrock function is plotted in figure 4.9, on the Quadratic function in figure 4.10.

4.5.8. Conjugate Gradients vs. BFGS

- BFGS is more costly per iteration than CG.
- BFGS converges in fewer steps.
- BFGS has less tendency to get stuck.
- BFGS requires algorithmic “hacks” to achieve a significant descent per iteration.
- Which one is better depends on the problem.

4.6. Wrap-Up

- Relation between machine learning and optimization
- Properties of good cost functions
- Convex sets and functions
- Importance of convex functions in machine learning
- Constrained and unconstrained optimization problems
- Lagrangian formulation
- Different numerical methods

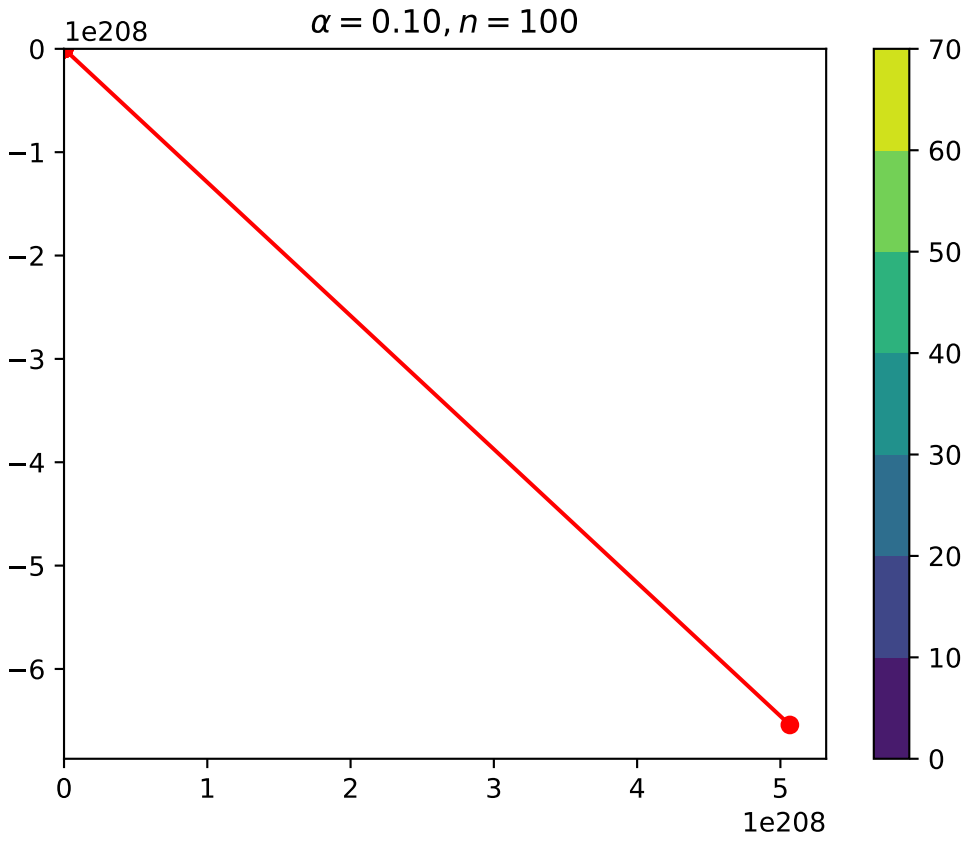

Figure 4.9.: CG on Rosenbrock

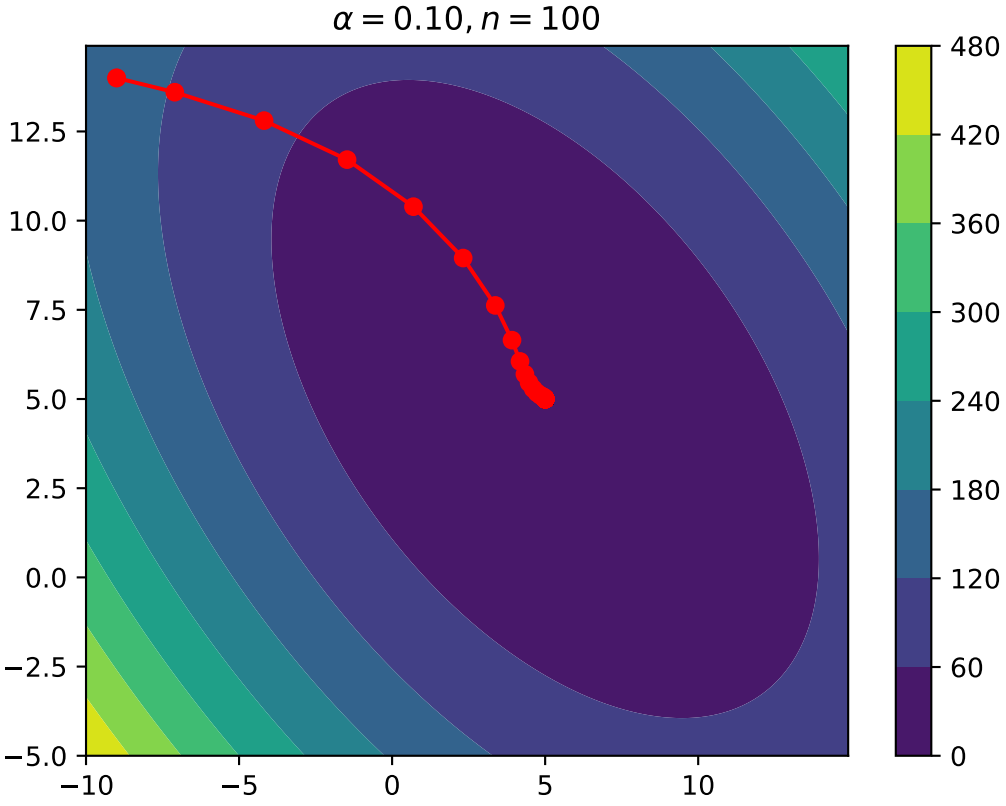

Figure 4.10.: CG on Quadratic

5. Bayesian Decision Theory

Bayesian decision theory is a statistical approach to make optimal decisions.

- All data in machine learning is generated by a stochastic process that is governed by the rules of probability.
- The data is understood as a set of samples from some underlying probability distribution.

5.1. Character Recognition

Goal: Classify a new letter so that the probability of a wrong classification is minimized where the only possibilities are a and b.

5.1.1. Class Conditional Probabilities

The *class conditional probability* (likelihood) $p(\boldsymbol{x} \mid C_k)$ is the probability of making an observation $\boldsymbol{x}$ knowing that it comes from some specific class C_k. $\boldsymbol{x}$ is often the *feature vector*, e.g. the number of black pixels, the height of black pixels, etc.

Let x be the height of black pixels and therefore a scalar value $x \; in \mathbb{R}$.

- This yields some useful decision theory: Given some x, decide for class a if $p(x \mid a) \geq p(x \mid b)$.
- But: If $p(x \mid a) = p(x \mid b)$, this yields no solution and class priors have to be taken into account.

Example Figure 5.1 shows some example conditional properties.

- For $x = 5$ or $x = 11$, the decision is clear (choose a or b respectively).
- But for $x = 8$, its completely unclear.

5.1.2. Class Priors

A *class prior* is a a-priori probability of a letter to occur (e.g. in the English alphabet, the probability of observing the letter e is much higher than observing a y).

- All class priors have to sum up to one (it must be anything).

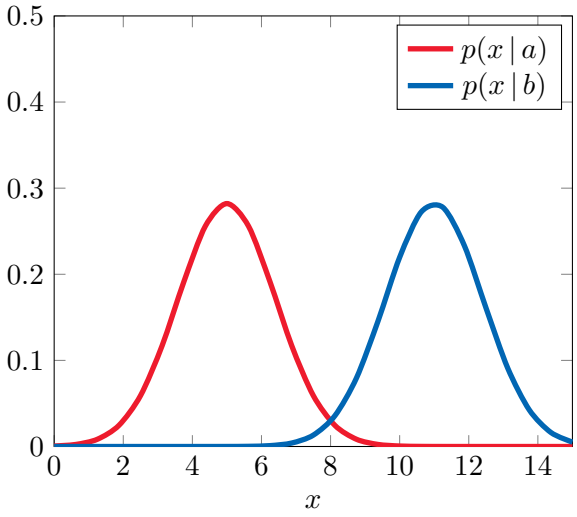

Figure 5.1.: Class Conditional Probabilities

Example In the text aaaaaaaabbabbaaaaaaa, the class priors are:

$$
\begin{aligned}
C_1 &= a \\
C_2 &= b \\
p(a) &= \frac{16}{20} = 80\% \\
p(b) &= \frac{4}{20} = 20\% \\
\sum_k p(C_k) &= p(a) + p(b) = 1
\end{aligned}
$$

By using Bayes theorem, the posterior can be calculated and the likelihood can be scaled by the prior to give a better view on the problem. Scaling by the normalization factor visualized the decision boundary.

Figure 5.2 shows these plots.

5.2. Bayesian Decision Theory

With the class conditional probability $p(X_k \mid \boldsymbol{x})$ and the prior $p(C_k)$, the class posterior can be calculated as

$$
p(C_k \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid C_K)p(C_k)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x} \mid C_K)p(C_k)}{\sum_j p(\boldsymbol{x} \mid C_j)p(C_j)}
$$

5.3. Bayesian Probabilities

- With *Bayesian probabilities*, probability is not just interpreted as a frequency of certain events, but as a degree of belief in an outcome.
- This allows to assert a prior belief in a data point coming from a certain class.

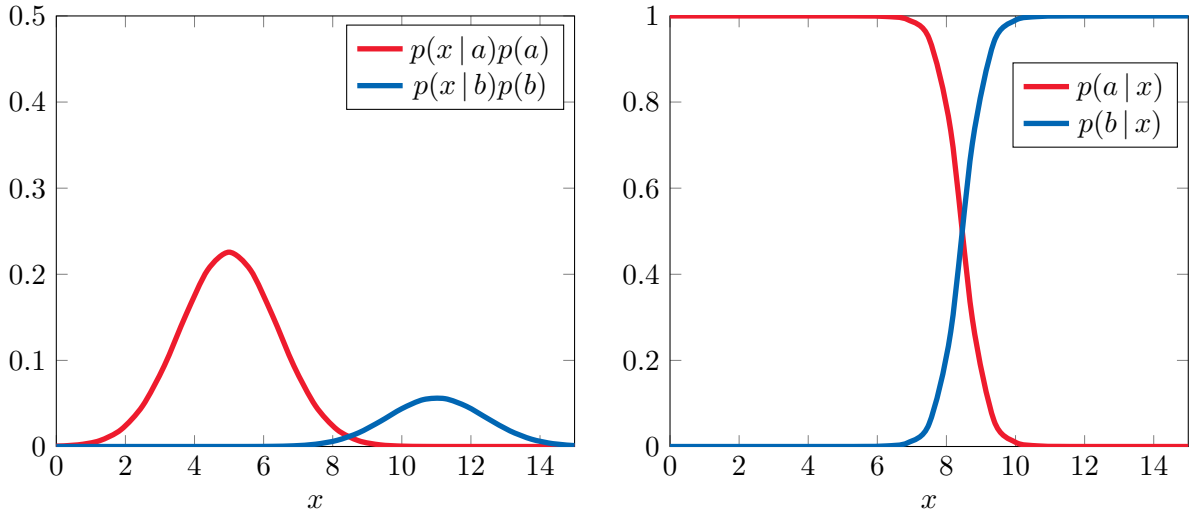

Figure 5.2.: Class Priors

5.4. Misclassification Rate

The goal of Bayesian decision theory is to minimize the *misclassification rate*, the probability of making a wrong decision:

$$p(\text{error}) = p(x \in R_1, C_2) + p(x \in R_2, C_1) = \int_{R_1} p(x \mid C_2)p(C_2)\,\mathrm{d}x + \int_{R_2} p(x \mid C_1)p(C_1)\,\mathrm{d}x$$

5.5. Decision Rule, Optimal Classifier and Decision Boundary

The basic *decision rule* is to decide for C_1 iff

$$p(C_1 \mid x) > p(C_2 \mid x) \quad \Longleftrightarrow \quad \frac{p(x \mid C_1)}{p(x \mid C_2)} > \frac{p(C_2)}{p(C_1)}$$

A classifier that obeys this rule is called *Bayesian optimal classifier*.

The *decision boundary* is the point where $\frac{p(x \mid C_1)}{p(x \mid C_2)} = \frac{p(C_2)}{p(C_1)}$. This line (or curve) can then be drawn into some graph and is the point where the classifier "switches" to the other class. This is most of the time only used for understanding what the classifier does than for real application (however, understanding what happens is really important).

5.5.1. Multiple Classes

Decide for class k iff it has the highest a-posteriori probability ($\forall j \neq k$)

$$p(C_k \mid x) > p(C_j \mid x) \quad \Longleftrightarrow \quad \frac{p(x \mid C_K)}{p(x \mid C_j)} > \frac{p(C_j)}{p(C_k)}$$

This yields more decision regions and multiple decision boundaries.

5.5.2. High Dimensional Features

For a lot of problems, the feature vector must have more than one entry, so $\boldsymbol{x} \in \mathbb{R}^n$ with $n \geq 2$. The derived decision boundaries still apply, but multivariate class conditional densities $p(\boldsymbol{x} \mid C_k)$ have to be taken into account.

5.6. Dummy Classes

In some applications, a *dummy class* "don't know" or "don't care" must be present (also called *reject option*).

5.7. Risk Minimization

- Minimizing the misclassification rate may not alway be enough, as not every misclassification may be equally bad.
- The key idea is to construct a *loss function* (or *cost function*) that expresses which misclassifications are really bad and which are not so bad.
- This loss function is called $\lambda(\alpha_i \mid C_j)$, where C_j is the actual class and α_i is the decision. Let $\lambda_{ij} \coloneqq \lambda(\alpha_i \mid C_j)$.
- The expected loss of making a decision α_i (the *overall risk*) then calculates as:

$$R(\alpha_i \mid x) = \mathbb{E}_{C_k \sim p(C_k \mid x)}\big(\lambda(\alpha_i \mid C_k)\big) = \sum_j \lambda(\alpha_i \mid C_j) p(C_j \mid x)$$

- So instead of minimizing the misclassification rate, minimize the overall risk.

5.7.1. Decision Rule

Decide for class C_1 iff

$$R(\alpha_2 \mid x) > R(\alpha_1 \mid x) \quad \iff \quad \frac{p(x \mid X_1)}{p(x \mid X_2)} > \frac{\lambda_{12} - \lambda_{22}}{\lambda_{21} - \lambda_{11}} \frac{p(C_2)}{p(C_1)}$$

This rule can be generalized for multiple classes and high dimensional features just like before.

Applying a 0-1 loss function

$$\lambda(\alpha_i \mid C_j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases}$$

yields the decision rule without an explicit loss function as seen in section 5.5.

5.8. Wrap-Up

- Class-conditional probabilities, class priors and class posteriors
- Bayesian decision theory
- Usage of Bayes theorem for classification

- Misclassification rate
- Bayes optimal classifier
- Generalization of decisions for more than two classes
- Risk and relation to misclassification

6. Probability Density Estimation

Probability density estimation (PDE) is about to estimate/learn the class conditional probability density $p(x \mid C_k)$.

- In supervised learning, both the input data points and their true labels/classes are known.
- The density is estimated separately for each class C_k.
- Let $p(x) \coloneqq p(x \mid C_k)$ for simplicity.
- There exist three models for PDE:
  - Parametric Models
    A "small" number of parameters completely define the probability density.
  - Non-Parametric Models
    No explicit parameters are used, but every known data point is used as a parameter (so, non-parametric models have as much parameters as there is data).
  - Mixture Models
    Combination of both.

6.1. Parametric Models

A simple case for a *parametric model* is the Gaussian distribution

$$p(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}(x-\mu)^2 \right\}$$

that is governed by two parameters: mean μ and variance σ^2. If both are known, the probability distribution is fully described.

The notation to say "variable x is defined by the parametric model $p(x \mid \theta)$ with parameters θ", write $x \sim p(x \mid \theta)$. For a Gaussian, the parameters are $\theta = (\mu, \sigma^2)$.

- *Learning* means to estimate the parameters θ given some training data $X = \{x_1, x_2, \cdots\}$.
- The *likelihood* of θ (the probability that the data X was generated from a probability density function with parameters θ) is defined as

$$L(\theta) = p(X \mid \theta) \overset{i.i.d.\text{ if}}{=} \prod_{i=1}^{N} p(x_i \mid \theta)$$

6.1.1. Maximum Likelihood

Assume that all data is i.i.d.!

- The parameters θ can be estimated by maximizing the likelihood.
- If a model has more than one parameter, maximize it w.r.t. each parameter once to get multiple estimators for the different parameters.
- Instead of maximizing the normal likelihood $L(\theta)$, it is mostly better to maximize the log-likelihood $\mathcal{L}(\theta) = \ln L(\theta)$ because:
  - it removes the product and turns it into a sum and
  - for members of the exponential family, removes the exponential part and splits the products into additions.

  This is possible because the logarithm is strictly increasing. In fact, every strictly increasing function can be used, but the logarithm is most of the time the best decision.

$$\mathcal{L}(\theta) = \ln\big(L(\theta)\big) = \ln p(X \,|\, \theta) = \sum_{i=1}^{N} \ln p(x_n \,|\, \theta)$$

- Then maximize the log-likelihood by taking the derivative w.r.t. θ (or the parameter to be estimated) and set them to zero.

Example Given the Gaussian probability density

$$p(x \,|\, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}(x-\mu)^2 \right\}$$

the log-likelihood computes as

$$\begin{aligned}\mathcal{L} &= \ln \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{1}{2\sigma^2}(x_i-\mu)^2 \right\} \\ &= \sum_{i=1}^{N} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(x_i-\mu)^2\end{aligned}$$

To estimate μ, take the derivative and set it so zero (this maximizes the likelihood w.r.t. the mean):

$$\begin{aligned} \nabla_\mu \mathcal{L} &= \frac{1}{\sigma^2}\sum_{i=1}^{N}(x_i-\mu) = -N\frac{1}{\sigma^2}\mu + \frac{1}{\sigma^2}\sum_{i=1}^{N} x_i \\ \implies \quad 0 &= -\frac{1}{\sigma^2}N\mu + \frac{1}{\sigma^2}\sum_{i=1}^{N} x_i \\ \iff \quad \mu &= \frac{1}{N}\sum_{i=1}^{N} x_i \end{aligned}$$

this yields the arithmetic mean as an estimator for the expectation value.

To estimate σ^2, take the derivative and set it so zero:

$$\begin{aligned}
\nabla_\sigma \mathcal{L} &= \sum_{i=1}^{N} -\frac{1}{\sigma} + \frac{1}{\sigma^3}(x_i - \mu)^2 = -\frac{1}{\sigma}N + \frac{1}{\sigma^3}\sum_{i=1}^{N}(x_i - \mu)^2 \\
\implies \quad 0 &= -\frac{1}{\sigma}N + \frac{1}{\sigma^3}\sum_{i=1}^{N}(x_i - \mu)^2 \\
\iff \quad \sigma^2 &= \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2
\end{aligned}$$

this yields an estimator for σ^2. But this estimator is biased, so the maximum likelihood estimation not always yields an unbiased estimator.

6.1.2. Degenerate Case

If only one data point is available ($N = 1$, $X = \{x_1\}$), the resulting Gaussian stretches infinitely to the top on one point, so is the likelihood.

To still get a useful estimate, a prior has to be put on the mean. This leads to Bayesian estimation.

Explanation A probability density function $p(x)$ such as the Gaussian has to integrate to 1:

$$\int_{-\infty}^{+\infty} p(x)\,\mathrm{d}x = 1$$

If the Gaussian has no variance $\sigma^2 = 0$, it does only have positive values on one exact point ($p(\mu) > 0$). As the integral can be thought of as calculating the "area under the curve", the function has to be somewhat 2-dimensional to have an area. In small Δx, the integral can be approximated with a square

$$\int_{x}^{x+\Delta x} p(x)\,\mathrm{d}x \approx p(x)\Delta x$$

By just looking at the square "withing" the mean $p(\mu)\Delta x$, the Δx shrinks to 0 as the variance is zero.

But the density function has to integrate to 1!

This way, we get:

$$\begin{aligned}
1 = \int_{-\infty}^{+\infty} p(x)\,\mathrm{d}x = \int_{-\infty}^{\mu} p(x)\,\mathrm{d}x + \int_{\mu}^{\mu} p(x)\,\mathrm{d}x + \int_{\mu}^{+\infty} p(x)\,\mathrm{d}x = \int_{\mu}^{\mu} p(x)\,\mathrm{d}x = \lim_{\Delta x \to 0} p(\mu)\Delta x \\
\implies p(\mu) \to \infty
\end{aligned}$$

to match the requirement.

This can also be calculated with $\lim_{x \to 0} \frac{1}{x} \to \infty$.

6.1.3. Bayesian Estimation

- In Bayesian estimation/learning of parametric distributions, it is assumed that parameters are not fixed, but are random variables too.
- This allows the usage of prior knowledge about the parameters.

The dependence on a prior can be formulated as a *conditional probability* $p(x \mid X)$:

$$p(x \mid X) = \int p(x, \theta \mid X) \, \mathrm{d}\theta \qquad p(x, \theta \mid X) = p(x \mid \theta, X) p(\theta \mid X)$$

As $p(x)$ is fully determined by θ (it is a *sufficient statistic*), $p(x \mid \theta, X) = p(x \mid \theta)$ holds. This way, the above equation can be simplified:

$$p(x \mid X) = \int p(x \mid \theta) p(\theta \mid X) \, \mathrm{d}\theta$$

The probability $p(\theta \mid X)$ makes it explicit how the parameters depend on the data and can be calculated using Bayes theorem

$$p(\theta \mid X) = \frac{p(X \mid \theta) p(\theta)}{p(X)}$$

with the prior $p(\theta)$.

If $p(\theta \mid X)$ is small for most θ, but large for a specific $\hat{\theta}$, the probability density $p(x \mid X)$ can be estimated as

$$p(x \mid X) \approx p(x \mid \theta)$$

this is called *Bayes point*. The more uncertain the estimator is about $\hat{\theta}$, the more the density is averaged across multiple θ.

Problem: Most of the time it is impossible to integrate over θ (or just do so numerically). Analytical solutions are rare.

Gaussian Bayesian Estimation

For a Gaussian distribution, there exists a closed form solution to estimate the density

$$p(\mu \mid X) = \frac{p(X \mid \mu) p(\mu)}{p(X)}$$

when the variance of the data distribution is known and fixed with prior $p(\mu) = \mu_\prime, \sigma_\prime^\in$.

Then, with the sample mean $\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i$, the parameters of the distribution $p(\mu \mid X) \sim \mathcal{N}(\mu_N, \sigma_N^2 =$ can be estimated as

$$\mu_n = \frac{N\sigma_0^2 + \sigma^2 \mu_0}{N\sigma_0^2 + \sigma^2} \qquad \frac{1}{\sigma_N^2} = \frac{N}{\sigma^2} + \frac{1}{\sigma_0^2}$$

where σ^2 is the variance of the data distribution, (μ_0, σ_0^2) are the parameters of the prior and (μ_N, σ_n^2) are the parameters to be estimated.

Conjugate Priors

- *Conjugate priors* are prior distributions that do not change the distribution family of the posterior distribution family, i.e. they both lie in the same distribution family.
- Gaussians are conjugate to themselves, which yields elegant closed form solutions.
- In general, this is not the case which makes everything more complicated.

6.2. Non-Parametric Models

- Non-parametric models are useful if the underlying probability density distribution family is unknown.
- They are directly estimated from data, without an explicit parametric model.
- Every data point is a parameter, so non-parametric models have an uncertain and possibly infinite number of parameters.
- The biggest problem with most estimation models is the "too smooth vs. not smooth enough" problem.
- Note: All of the following examples use a dataset of 10000 data points that was generated by a mixture of two Gaussians $(5, 10)$ and $\mathcal{N}(10, 5)$, both equally weighted. which is plotted in red as the actual distribution.

6.2.1. Histograms

- *Histograms* discretize the continuous feature space into discrete bins of data.
- They can be used for nearly every problem and can approximate any probability density arbitrarily well with the right data set.
- But it is a brute-force method.
- In high dimensional feature spaces, histograms become impractical because of the exponential increase of bins. They require exponentially much data. This is known as the *curse of dimensionality*.
- The size of the bins is somewhat arbitrary.

Formally The probability that a data point $\boldsymbol{x}$ falls into Region R is measured as

$$P(\boldsymbol{x} \in R) = \int_R p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}$$

If R is sufficiently small with volume V, $p(\boldsymbol{x})$ is almost constant:

$$P(\boldsymbol{x} \in R) \approx p(\boldsymbol{x})V$$

If R is sufficiently large with volume V:

$$P(\boldsymbol{x} \in R) = \frac{K}{N} \quad \Longrightarrow \quad p(\boldsymbol{x}) \approx \frac{K}{NV}$$

where N is the total number of data points and K is the number of points that fall into region R.

Example Figure 6.1 shows three histograms that are not smooth enough (bin size 0.5), just right (bin size 3) and too smooth (bin size 20).

6.2.2. Kernel Density Estimation (KDE)

Kernel density estimation (KDE) is a variation of "histograms" where V gets fixed and K is determined (i.e. count the data points that fall in a fixed hypercube).

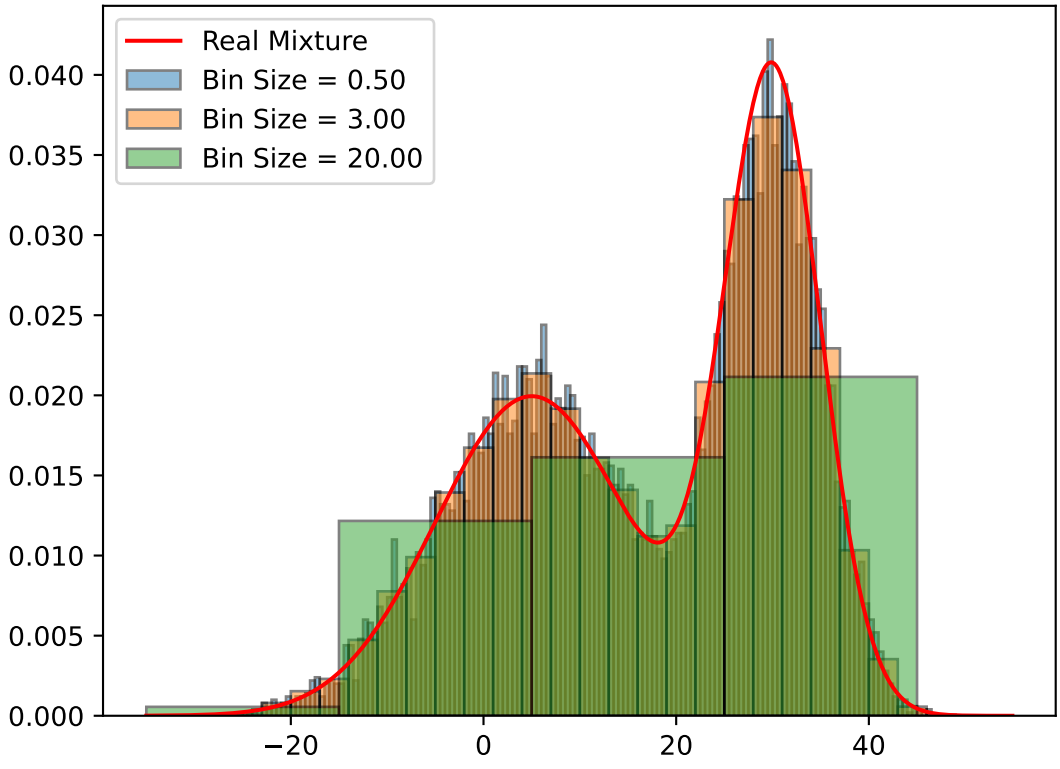

Figure 6.1.: Histogram

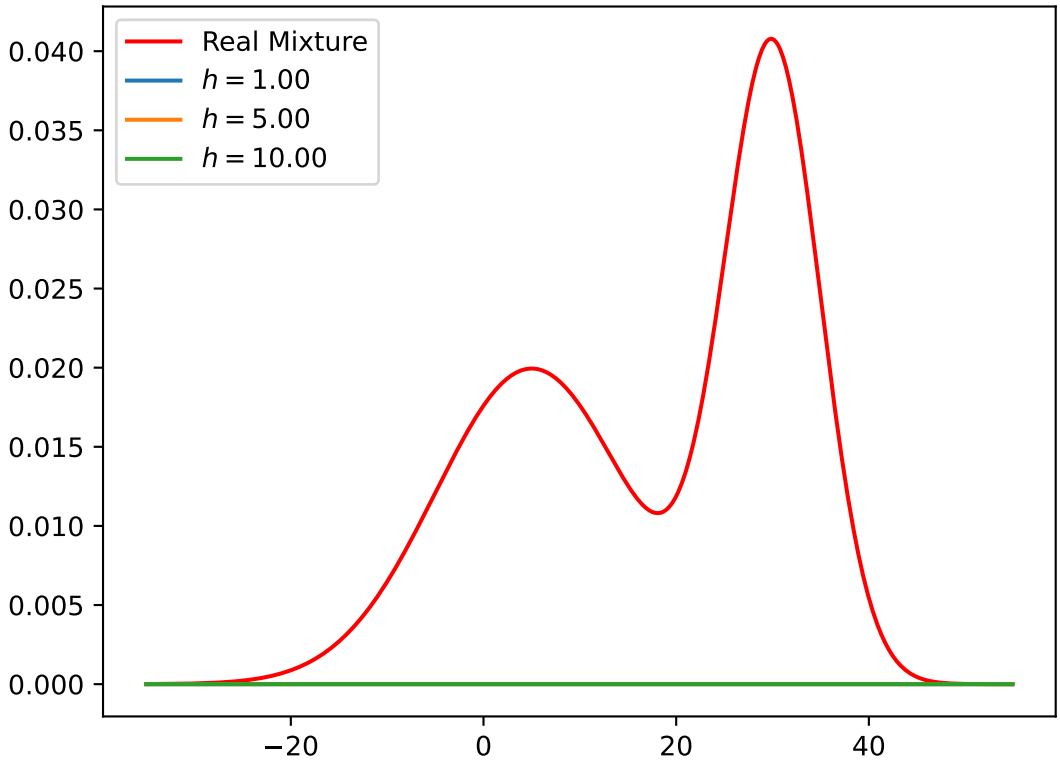

Figure 6.2.: Kernel Density Estimation (Parzen Window)

Parzen Window

These hypercube is called *Parzen window* in d dimensions with edge length h. It has the following equations:

$$H(\boldsymbol{u}) = \begin{cases} 1 & |u_j| \leq \frac{h}{2}, j = 1, \cdots, d \\ 0 & \text{otherwise} \end{cases}$$

$$V = \int H(\boldsymbol{u}) \, \mathrm{d}\boldsymbol{u} = h^d$$

$$K(\boldsymbol{x}) = \sum_{i=1}^{N} H\left(\boldsymbol{x} - \boldsymbol{x}^{(i)}\right)$$

$$\implies \qquad p(\boldsymbol{x}) \approx \frac{K(\boldsymbol{x})}{NV} = \frac{1}{Nh^d} \sum_{i=1}^{N} H\left(\boldsymbol{x} - \boldsymbol{x}^{(i)}\right)$$

Example Figure 6.2 shows the estimated density distribution using kernel density estimation with a Parzen window.

Gaussian Kernel

The Gaussian kernel uses a "soft" window in d dimensions with parameter h and gives smoother results than the Parzen window. Problem: Has infinite support and requires a lot of computation. It has the following equations:

$$H(\boldsymbol{u}) = \frac{1}{\left(\sqrt{2\pi h^2}\right)^d} \exp\left\{ -\frac{\|\boldsymbol{u}\|^2}{2h^2} \right\}$$

$$V = \int H(\boldsymbol{u}) \, \mathrm{d}\boldsymbol{u} = 1$$

$$K(\boldsymbol{x}) = \sum_{i=1}^{N} H\left(\boldsymbol{x} - \boldsymbol{x}^{(i)}\right)$$

$$\implies \qquad p(\boldsymbol{x}) \approx \frac{K(\boldsymbol{x})}{NV} = \frac{1}{N\left(\sqrt{2\pi h^2}\right)^d} \sum_{i=1}^{N} \exp\left\{ -\frac{\|\boldsymbol{x} - \boldsymbol{x}^{(i)}\|^2}{2h^2} \right\}$$

Example Figure 6.3 shows the estimated density distribution using kernel density estimation with a Gaussian kernel. The parameter $h = 5$ seems to fit the density the best at the first view while not being too noisy.

Arbitrary Kernel

An arbitrary kernel has the following form and the kernel function $k(\boldsymbol{u})$ (which must have the properties $k(\boldsymbol{u}) \geq 0$ and $\int k(\boldsymbol{u}) \, \mathrm{d}\boldsymbol{u} = 1$):

$$V = h^d$$

$$K(\boldsymbol{x}) = \sum_{i=1}^{N} k\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}^{(i)}\|^2}{h} \right)$$

$$\implies \qquad p(\boldsymbol{x}) \approx \frac{K(\boldsymbol{x})}{NV} = \frac{1}{Nh^d} \sum_{i=1}^{N} k\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}^{(i)}\|^2}{h} \right)$$

Common Kernel Comparison

All kernel methods have one problem in common: The kernel bandwidth h has to be selected properly.

Parzen Window

$$k(u) = \begin{cases} 1 & |u| \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

- Not very smooth results.

Gaussian Kernel

$$k(u) = \frac{1}{\sqrt{2\pi}} \exp\left\{ -\frac{1}{2} u^2 \right\}$$

- Problem: Kernel has infinite support and requires a lot of computation.
- But gives much smoother results than the Parzen window.

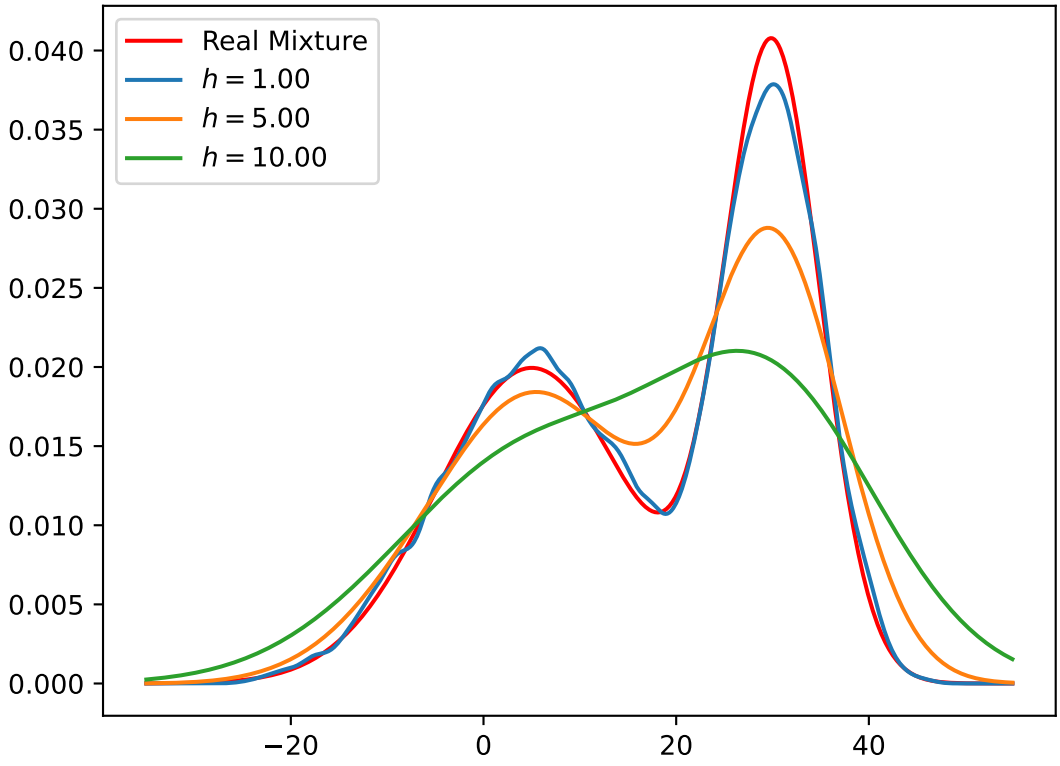

Figure 6.3.: Kernel Density Estimation (Gaussian Kernel)

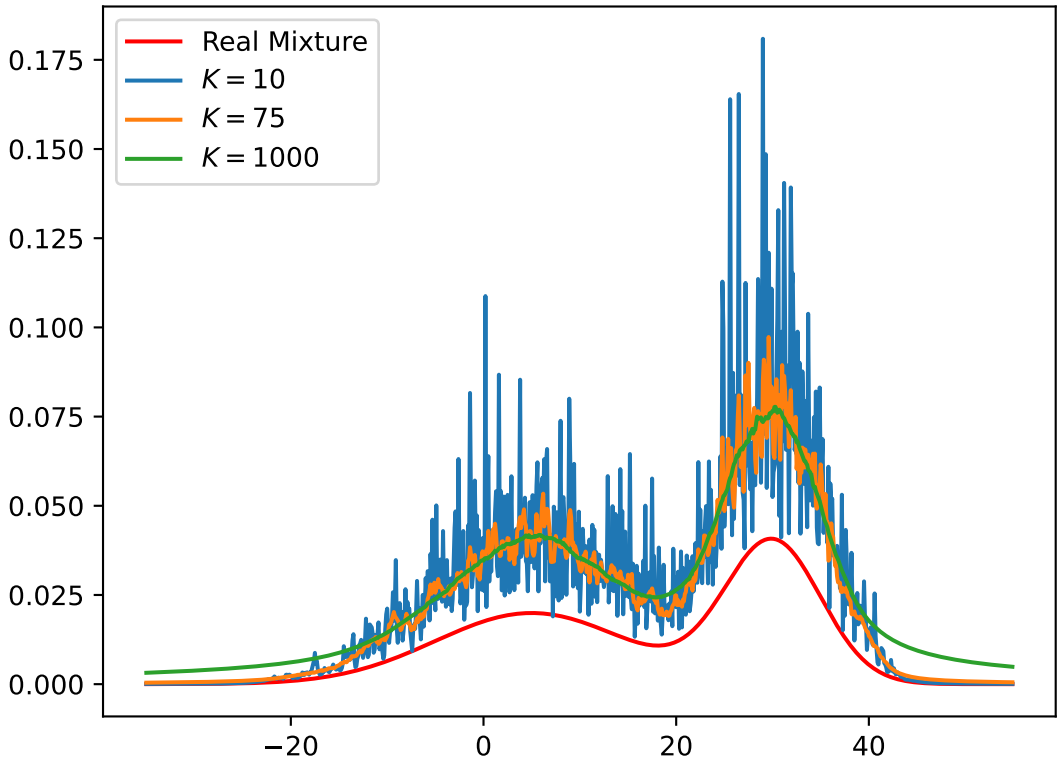

Figure 6.4.: K-Nearest Neighbors

Epanechnikov Kernel

$$k(u) = \max\left\{0, \quad \frac{3}{4}(1-u)^2\right\}$$

- Smoother and has finite support.

6.2.3. K-Nearest Neighbors (KNN)

Kernel density estimation (KDE) is a variation of "histograms" where K gets fixed and V is determined (i.e. increase the size of a sphere until K data points fall into it). In general, KNN produces a pretty noisy density estimation and is very sensitive to small changes.

Example Figure 6.4 shows the estimated density distribution using K-nearest neighbors. The parameter $K = 75$ seems to fit the density the best at the first view while not being too noisy (of course it is still pretty noise, because it is KNN).

Classification

Assume a data set with N points, where N_j is the number of points in class C_j and $\sum_j N_j = N$. To classify a new point x, draw a sphere around the point that contains K points (regardless which class they belong to). Let V be the volume of the sphere that contains K_j points of class C_j.

With Bayesian classification

$$P(C_j \mid x) = \frac{P(x \mid C_j)P(C_j)}{P(X)}$$

this yields the solution

$$P(X) \approx \frac{K}{NV} \qquad P(x \mid C_j) \approx \frac{K_j}{N_j V} \qquad P(C_j) \approx \frac{N_j}{N} \implies \qquad P(C_j \mid x) \approx \frac{K_j}{N_j V}\frac{N_j}{N}\frac{NV}{K} = \frac{K_j}{K}$$

This, with KNN, the posterior probability can be computed without the knowledge about how many data points are available and without an explicit influence of the sphere size.

6.3. Mixture Models

Mixture models combine parametric and non-parametric models.

The probability density $p(x)$ of a mixture model can be described as

$$p(x) = \sum_{j=1}^{M} p(x \mid j)p(j)$$

where M is the number of mixture components and $p(j)$ is the probability (or *weight*) of mixture component j. These probabilities have to sum up to one.

6.3.1. Mixture of Gaussians

A *mixture of Gaussians* (MoG) is one of the basic mixture models. It has the following form (where $p(x \mid j)$ is just another notation for $p(x \mid C_j)$, similar for other probability densities):

$$\begin{aligned} p(x) &= \sum_{j=1}^{M} p(x \mid j)p(j) \\ p(x \mid j) &= \mathcal{N}(x \mid \mu_j, \sigma_j^2) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left\{ -\frac{1}{2\sigma_j^2}(x-\mu_j)^2 \right\} \\ p(j) &= \pi_j, \quad 0 \le \pi_j \le 1, \quad \sum_{j=1}^{M} \pi_j = 1 \end{aligned}$$

with the mixture parameters $\theta = \{\mu_1, \sigma_1^2, \pi_1, \cdots, \mu_M, \sigma_M^2, \pi_M\}$.

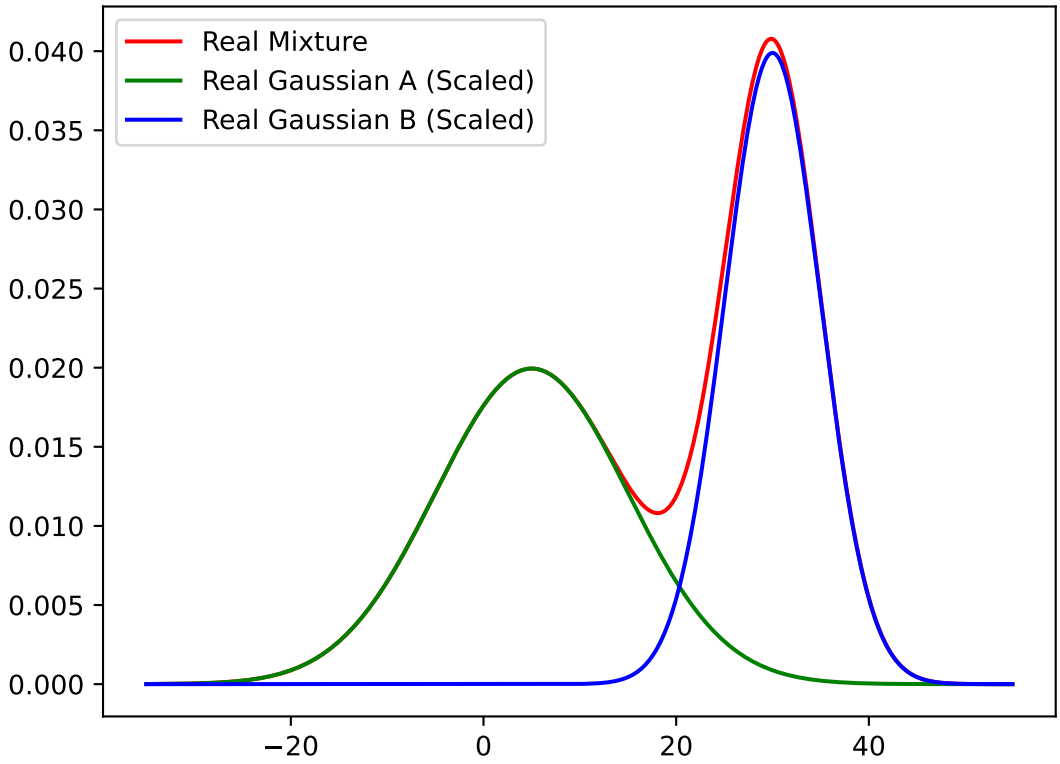

Figure 6.5.: Mixture of Gaussians

Figure 6.5 shows the mixture of Gaussians that was used for the examples in the previous section that has the following properties:

$$
\begin{aligned}
p(x \mid 1) &= \mathcal{N}(x \mid 5, 10) \\
p(x \mid 2) &= \mathcal{N}(x \mid 30, 5) \\
\pi_1 = \pi_2 &= \frac{1}{2} \\
\implies \qquad p(x \mid 5, 10, 0.5, 30, 5, 0.5) &= \frac{1}{2}\mathcal{N}(x \mid 5, 10) + \frac{1}{2}\mathcal{N}(x \mid 30, 5)
\end{aligned}
$$

This example will be used for all further examples with a dataset with 10000 data points. The real mixture distribution is plotted in red.

Maximum Likelihood Estimation

Applying MLE to a mixture of Gaussians

$$\mathcal{L} = \ln L(\theta) = \sum_{i=1}^{N} \ln p(x_i \mid \theta)$$

$$\implies \quad \nabla_\theta \mathcal{L} = 0$$

$$\implies \quad \mu_j = \frac{\sum_{j=1}^{N} p(j \mid x_j) x_j}{\sum_{j=1}^{N} p(j \mid x_j)}$$

gives a circular dependency through all estimators. Therefore, no analytical solution exist!

Gradient Ascent

- *Gradient ascent* can be used to maximize the log-likelihood numerically.
- But it typically has a complex (nonlinear, circular) gradient.
- So the optimization of one Gaussian depends on all other components.
- Hard to compute!

Different Strategy Split the data set into *observes* and *unobserved* (*latent*) variables. Typically, x is observed and $p(j \mid x)$ is unobserved (the component that has generated an x is latent).

- If both the observed and the latent dataset are known (the *complete* dataset), the maximum likelihood solution can be computed via

$$\mu_j = \frac{\sum_{j=1}^{N} p(j \mid x_j) x_j}{\sum_{j=1}^{N} p(j \mid x_j)}$$

- If the distributions are known, the unobserved data can be inferred using Bayes decision rule.
- But if neither the latent dataset nor the distribution is known, an estimation of j is needed. This can be done using clustering.

6.3.2. Estimation using Clustering

Hard Assignments

- Every points gets assigned a mixture label.
- No points gets "multiple" labels with probabilities. It is a 0-1 labeling (*hard assignments*).
- The mixture components are then estimated using only this data.

Gaussians If a guess about the distribution is available, but the unobserved data is not, the probabilities can be calculated for each mixture component:

$$p(j \mid x) = \frac{p(x \mid j)\pi_j}{\sum_{j=1}^{M} p(x \mid j)\pi_j}$$

Expectation Maximization (EM)

Let be the observed data and let be the latent data, so the complete data is $Z = (X, Y)$.

- The *expectation maximization* (EM) algorithm is used to perform maximum likelihood estimation even if the data is incomplete.
- Idea: Estimate the latent variables and use the estimations to estimate the distribution parameters.
- In case of Gaussian mixtures, associate every data point to one of the mixture components.

Properties and Definitions With the observed data $X = \{x_1, \cdots, x_N\}$, the unobserved data $Y = \{y_1, \cdots, y_N\}$ and the joint density

$$p(Z) = p(X, Y) = p(Y \mid X)p(X)$$

with parameters

$$p(Z \mid \theta) = p(X, Y \mid \theta) = p(Y \mid X, \theta)p(X \mid \theta)$$

the incomplete and complete likelihood can be defined as:

- Incomplete Likelihood

$$L(\theta \mid X) = p(X \mid \theta) = \prod_{i=1}^{N} p(x_i \mid \theta)$$

- Complete Likelihood

$$L(\theta \mid Z) = p(Z \mid \theta) = p(Y \mid X, \theta)p(X \mid \theta) = \prod_{i=1}^{N} p(y_i \mid x_i, \theta)p(x_i \mid \theta)$$

Algorithm Y is not known, but the current guess $\theta^{(i-1)}$ of the parameters θ can be used to predict Y. Formally, this is to compute the expected value of the complete log-likelihood given the data X and the current estimation $\theta^{(i-1)}$:

$$Q(\theta, \theta^{(i-1)}) := \mathbb{E}_Y\big(\ln p(X, Y \mid \theta) \mid X, \theta^{(i-1)}\big) = \int p(y \mid X, \theta^{(i-1)}) \ln p(X, y \mid \theta) \, \mathrm{d}y$$

Repetition: X and $\theta^{(i-1)}$ are fixed while Y and θ are (random) variables.

The algorithm then contains two steps:

E-Step (Expectation) Compute $p(y \mid X, \theta^{(i-1)})$.

M-Step (Maximization) Maximize the expected value of the log-likelihood to get the next estimation $\theta^{(i)}$

$$\theta^{(i)} = \arg\max_{\theta} Q(\theta, \theta^{(i-1)})$$

Formal Properties

- The expected log-likelihood of the i-th iteration is at least as good as that of the $i-1$-th iteration:

$$Q(\theta^{(i)}, \theta^{(i-1)}) \geq Q(\theta^{(i-1)}, \theta^{(i-1)})$$

- If this expectation is maximized w.r.t. $\theta^{(i)}$, then the following holds:

$$L(\theta^{(i)} \mid X) \geq L(\theta^{(i-1)} \mid X)$$

- Thus, the incomplete log-likelihood increases in every iteration or at least stays the same.
- The incomplete log-likelihood is optimized (locally).
- In practice, the results depend highly on the initialization. A good initialization is crucial or EM might get stuck in local optima.

Gaussian Mixtures

- In the special case of Gaussian mixtures, there exists a closed form solution.
- Also estimate the variance and the prior distribution over the mixture components.
- Algorithm 5 shows the EM algorithm for univariate Gaussian mixture models, where M is the number of Gaussians and N is the amount of data.
- Figure 6.6 shows the algorithm in action and shows the progress between the first and last iteration.

Algorithm 5: EM for Univariate Gaussian

Initialize $\mu_1, \sigma_1, \pi_1, \cdots, \mu_M, \sigma_M, \pi_M$
for $i = 1, \cdots, n$ **do**
  $\alpha_{kj} \leftarrow p(j \mid x_k) = \frac{\mathcal{N}(x_k \mid \mu_j, \sigma_j^2)\pi_j}{\sum_{i=1}^{M} \mathcal{N}(x_k \mid \mu_i, \sigma_i^2)\pi_i}$
  $N_j \leftarrow \sum_{i=1}^{N} \alpha_{ij}$
  $\mu_j^{\text{new}} \leftarrow \frac{1}{N_j} \sum_{i=1}^{N} \alpha_{ij} x_i$
  $\sigma_j^{\text{new}} \leftarrow \sqrt{\frac{1}{N_j} \sum_{i=1}^{N} \alpha_{ij} \left(x_i - \mu_j^{\text{new}}\right)}$
  $\pi_j^{\text{new}} \leftarrow \frac{N_j}{N}$
return $\mu_1, \sigma_1, \pi_1, \cdots, \mu_M, \sigma_M, \pi_M$

Derivation the Gaussian EM Algorithm The "EM algorithm" itself is not really an algorithm, but instead a method to derive an EM algorithm for a probability distribution. This section covers the derivation of the EM algorithm for univariate Gaussian mixtures. The observed data is $X = \{x_1, \cdots, x_M\}$ and the latent data is $Y = \{y_1, \cdots, y_M\}$ where y_i denotes the mixture component a data point x_i belongs to.

With discrete y_i, the equation for $Q(\theta, \theta^{(i-1)})$ simplifies:

$$Q(\theta, \theta^{(i-1)}) = \sum_{j=1}^{M} p(y_j \mid X, \theta^{(i-1)}) \ln p(X, y_j \mid \theta) = \sum_{j=1}^{M} \sum_{n=1}^{N} \underbrace{p(y_j \mid x_n, \theta^{(i-1)})}_{:=\alpha_{nj}} \ln p(x_n, y_j \mid \theta)$$

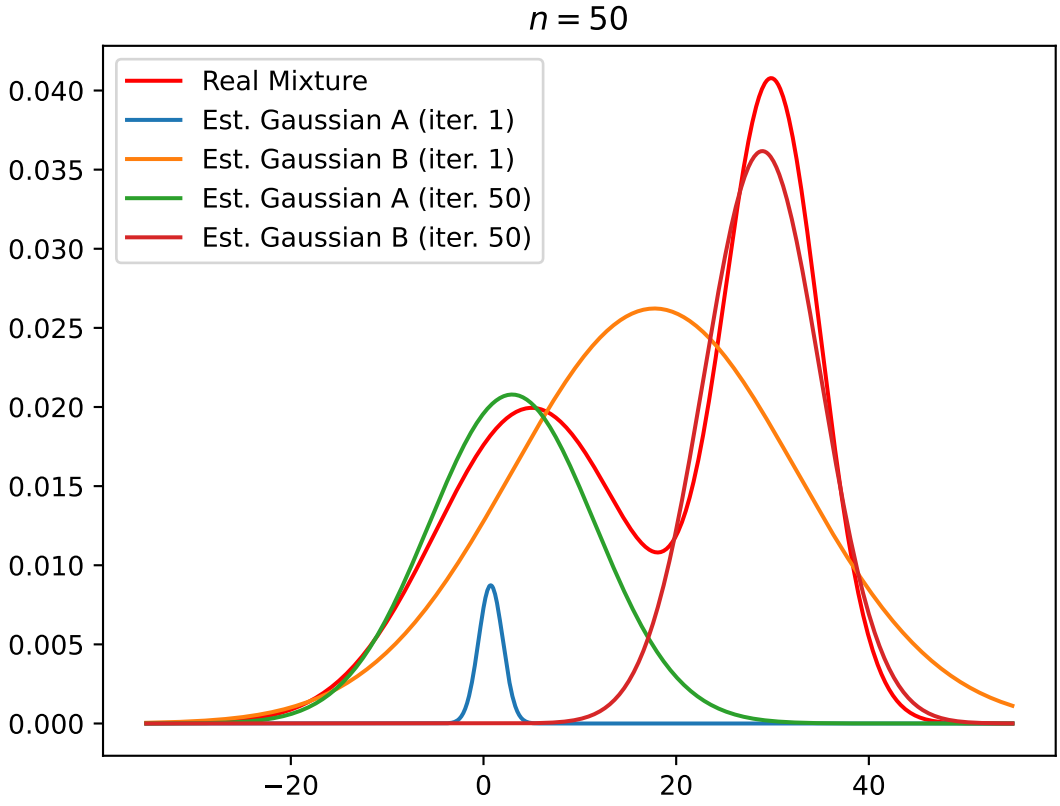

Figure 6.6.: EM for Univariate Gaussian

Now calculate α_{nj}:

$$\begin{aligned}
\alpha_{nj} &= p(y_j \mid x_n, \theta^{(i-1)}) \\
&= \frac{p(x_n, \theta^{(i-1)} \mid y_j) p(x_n, \theta^{(i-1)})}{p(y_j)} \\
&\overset{\dagger}{=} \frac{p(x_n \mid \theta^{(i-1)}) p(\theta^{(i-1)})}{p(y_j)} \\
&= \frac{p(x_n \mid \mu_j, \sigma_j^2)\pi_j}{\sum_{i=1}^{M} p(x_n \mid \mu_i, \sigma_i^2)\pi_i}
\end{aligned}$$

Step † is possible because $\theta^{(i-1)}$ is a sufficient statistic and therefore fully determines the probability density.

This yields the formula for the E-Step.

To get the formula for the M-Step, insert α_{nj} into $Q(\theta^{\text{new}}, \theta^{(i-1)})$, simplify, take the derivatives w.r.t. θ^{new} and set them to zero. Let $N_j \coloneqq \sum_{n=1}^{N} \alpha_{nj}$.

$$\begin{aligned}
& & Q(\theta^{\text{new}}, \theta^{(i-1)}) &= \sum_{j=1}^{N}\sum_{n=1}^{N} \alpha_{nj} \ln p\big(x_n, y_j \mid \theta^{\text{new}}\big) \\
\implies & & Q_j &= \sum_{n=1}^{N} \alpha_{nj} \ln p\big(x_n, y_j \mid \mu_j^{\text{new}}, \sigma_j^{\text{new}}\big) \\
& & &= \sum_{n=1}^{N} \alpha_{nj} \ln \frac{1}{\sqrt{2\pi\big(\sigma_j^{\text{new}}\big)^2}} - \alpha_{nj} \frac{1}{2\big(\sigma_j^{\text{new}}\big)^2}\big(x_n - \mu_j^{\text{new}}\big)^2 \\
\implies & & \nabla_{\mu_j^{\text{new}}} Q_j &= -N_j \frac{1}{\big(\sigma_j^{\text{new}}\big)^2}\mu_j + \frac{1}{\big(\sigma_j^{\text{new}}\big)^2}\sum_{n=1}^{N} \alpha_{nj} x_n \\
\implies & & N_j \mu_j^{\text{new}} &= \sum_{n=1}^{N} \alpha_{nj} x_n \\
\iff & & \mu_j^{\text{new}} &= \frac{1}{N_j}\sum_{n=1}^{N} \alpha_{nj} x_n \\
\implies & & \nabla_{\sigma_j^{\text{new}}} Q_j &= \sum_{n=1}^{N} -\alpha_{nj}\frac{1}{\sigma_j^{\text{new}}} + \alpha_{nj}\frac{1}{\big(\sigma_j^{\text{new}}\big)^3}\big(x_n - \mu_j^{\text{new}}\big)^2 \\
& & &= -\frac{1}{\sigma_j^{\text{new}}} N_j + \frac{1}{\big(\sigma_j^{\text{new}}\big)^3}\sum_{n=1}^{N} \alpha_{nj}\big(x_n - \mu_j^{\text{new}}\big)^2 \\
\implies & & \big(\sigma_j^{\text{new}}\big)^2 N_j &= \sum_{n=1}^{N} \alpha_{nj}\big(x_n - \mu_j^{\text{new}}\big)^2 \\
\iff & & \big(\sigma_j^{\text{new}}\big)^2 &= \frac{1}{N_j}\sum_{n=1}^{N} \alpha_{nj}\big(x_n - \mu_j^{\text{new}}\big)^2 \\
\iff & & \sigma_j^{\text{new}} &= \sqrt{\frac{1}{N_j}\sum_{n=1}^{N} \alpha_{nj}\big(x_n - \mu_j^{\text{new}}\big)^2}
\end{aligned}$$

This yields the formula for the M-Step (with the prior calculation $\pi_j = \frac{N_j}{N}$ as of the definition).

The same technique can be applied to derive EM algorithms for other distributions (e.g. multivariate Gaussians). Notice that it might not yield a closed form solution, especially for distributions that are not part of the exponential family (they are kind of "easy" because the exponential disappears with the logarithm),

6.3.3. Mixture Components

- The biggest problem with mixture models is: How many mixture components are needed? More lead to a better likelihood, but are not always better because of overfitting.
- There exist some heuristics for automatic selection:
  - Find a K that maximizes the *Akaike information criterion* $\ln p(X \mid \theta_{\text{ML}}) - K$ where K is the number of parameters.
  - Or find a K that maximizes the *Bayesian information criterion* $\ln p(X \mid \theta_{\text{ML}}) - \frac{1}{2} K \ln N$ where N is the number of data points.
- Mixture models are much more general than just mixture of Gaussians, the components can even lie in different distribution families.

6.4. Wrap-Up

- Difference between parametric and non-parametric models
- The likelihood function and how to derive maximum likelihood estimators
- Bayesian estimation
- Different non-parametric models (histogram, KDE, KNN)
- Mixture models
- EM-algorithm

7. Clustering

Clustering is about to find meaningful groups of data points and find the group assignment. It is a type of unsupervised learning as no labeled data is needed. Clustering can be split into two different basic types:

- Agglomerative Clustering
  - Each data point is made a different cluster.
  - While the clustering is not satisfactory, the two clusters with the smallest inter-cluster distance are merged.
- Divisive Clustering
  - All data points lie in a single cluster.
  - While the clustering is not satisfactory, split the cluster that yields the two components with the largest inter-cluster distance.

Note: All of the following examples use a dataset of 1000 data points that was generated by a mixture of the two-dimensional multivariate Gaussians, plotted in figure 7.1.

7.1. Mean Shift Clustering

Mean shift clustering is a agglomerative clustering method for finding the modes (maxima) in a cloud of data points where the points are most dense using kernel density estimation and local search.

- The search path starts at different points and "climbs up the hills" (mean shift clustering is a hill climbing algorithm).
- Paths that converge at the same point get the same label.

The grand scheme is to start with a kernel density estimate

$$\hat{f}(\boldsymbol{x}) = \frac{1}{Nh^d} \sum_{i=1}^{N} k\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{h^2} \right)$$

and then derive the mean shift procedure by taking the gradient of the kernel density estimate to calculate the mean shift $\boldsymbol{m}_{h,g}(\boldsymbol{x})$:

$$\boldsymbol{m}_{h,g}(\boldsymbol{x}) = \frac{\sum_{i=1}^{N} g\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{h^2} \right) \boldsymbol{x}_i}{\sum_{i=1}^{N} g\left(\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{h^2} \right)} - \boldsymbol{x}$$

where $g(\boldsymbol{u}) = -k'(\boldsymbol{u})$ and move into the direction $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{m}_{h,g}(\boldsymbol{x})$. Repeat this until convergence and repeat this for each data point. All points that converge to the same data point lie in one cluster.

Algorithm 6 shows the mean shift algorithm in its basic form with a data set $X = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ and learning rate α.

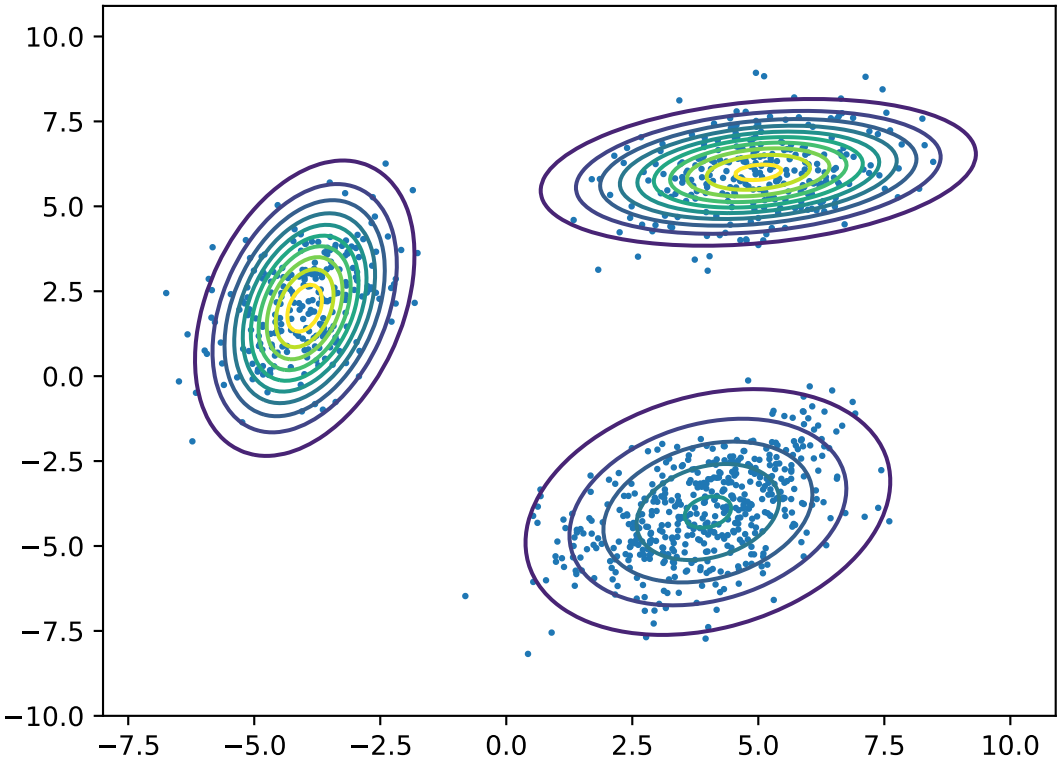

Figure 7.1.: Cluster Gaussians

Algorithm 6: Mean Shift Clustering

for $k = 1, \cdots, n$ **do**
  for $j = 1, \cdots, N$ **do**
    $\boldsymbol{m} \leftarrow \frac{\sum_{i=1}^{N} g\left(\frac{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|^2}{h^2}\right) \boldsymbol{x}_i}{\sum_{i=1}^{N} g\left(\frac{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|^2}{h^2}\right)} - \boldsymbol{x}_j$
    $\boldsymbol{x}_j \leftarrow \boldsymbol{x}_j + \alpha \boldsymbol{m}$

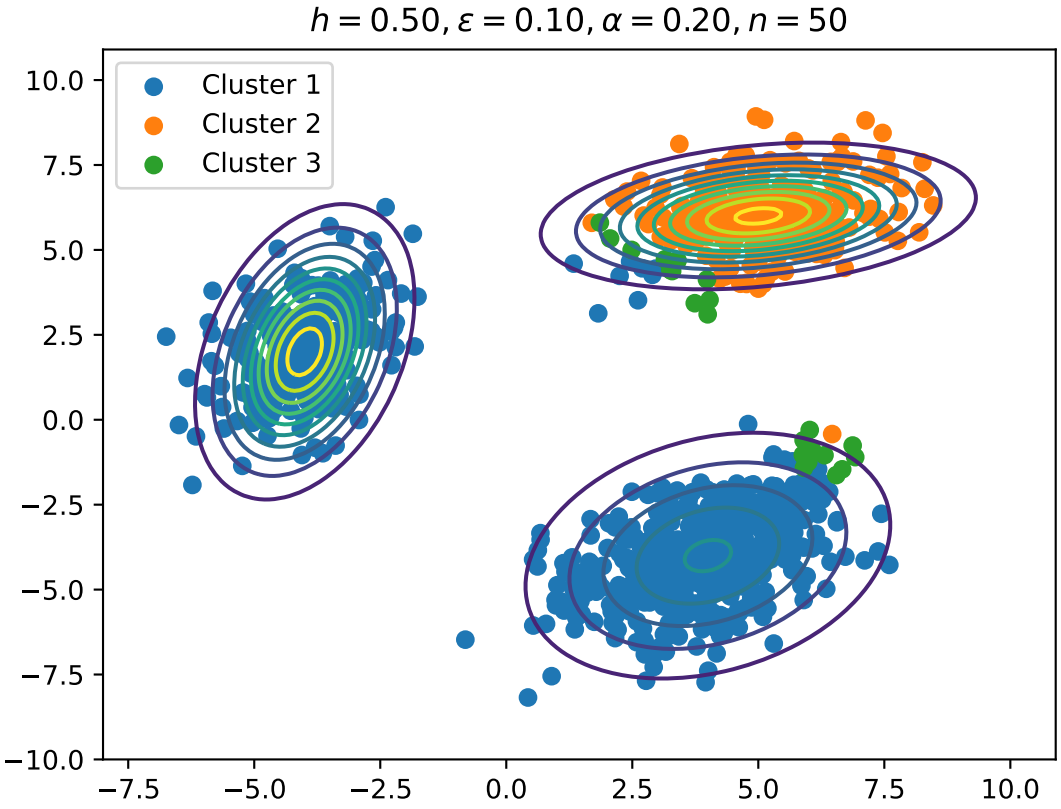

Figure 7.2.: Mean Shift Clustering

Example Figure 7.2 shows the mean shift clustering algorithm on a mixture of three bivariate Gaussian distributions, figure 7.3 shows the way each data point goes.

7.2. Wrap-Up

- Different algorithms for clustering

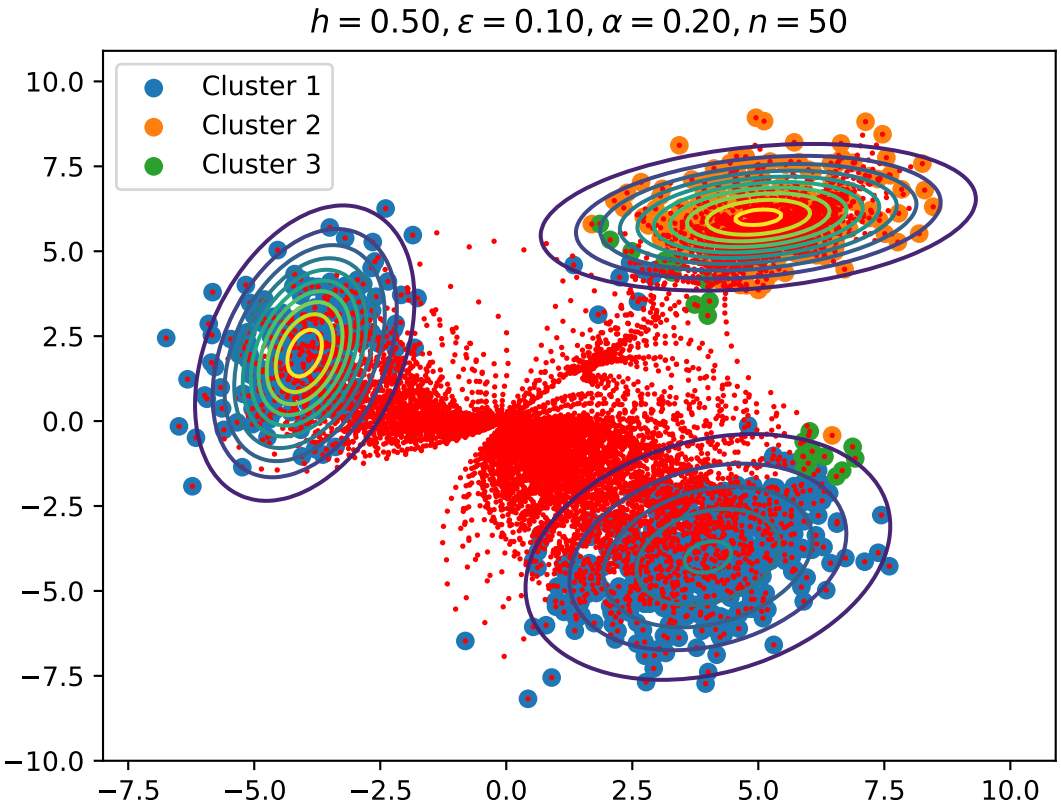

Figure 7.3.: Mean Shift Clustering (Way)

8. Evaluation

The *performance* for parameter estimation and classification has to be measured in order to compare them and to detect under-/overfitting.

8.1. Test Error vs. Training Error

- The training error might be really low where the test error is really high. This is an indicator for overfitting.
- If both errors are large, the model seems to underfit.
- The model selection has to be done carefully!

8.2. Bias and Variance

The *bias* of an estimator $\hat{\theta}$ is the expected derivation of the true parameter θ (with a data set X):

$$\text{Bias}\left(\hat{\theta}\right) = \mathbb{E}_X\left(\hat{\theta}(X) - \theta\right)$$

If the expected value of an estimation differs from the true value, the estimator is called *biased*. If not, is is called *unbiased*. The *variance* is the expected squared error between the estimator and the mean estimator:

$$\text{Var}\left(\hat{\theta}\right) := \mathbb{E}_X\left(\left(\hat{\theta}(X) - \mathbb{E}_X\left(\hat{\theta}(X)\right)\right)^2\right)$$

8.2.1. MVUE and BLUE

- An estimator with zero bias and minimum variance is called a *minimum variance unbiased estimator* (MVUE).
- A MVUE that is linear in its features is called *best linear unbiased estimator* (BLUE).

8.2.2. Bias-Variance Tradeoff

- In practice, an unbiased estimator with a small variance is wanted. But mostly, this is not possible.
- The bias represents the structural error whereas the variance represents the estimation error (finite data sets will always have variance).
- The expected total error is proportional to $\text{Bias}^2 + \text{Variance}$. Typically not both can be minimized.

- The learning algorithm has to find the right tradeoff between bias and variance (simple enough to prevent overfitting and yet expressive enough to represent the important parts of the data).
- To ensure this, the algorithm has to be evaluated on the test data (see section 8.3).

8.2.3. Example: MLE of a Gaussian

The following to sections will cover the calculation of the bias of the maximum likelihood estimators for a Gaussian distribution for μ (called $\hat{\mu}$) and σ^2 (called $\hat{\sigma}^2$).

Mean (μ) The estimator is given as

$$\hat{\mu}(X) = \frac{1}{N}\sum_{i=1}^{N} x_i$$

So the bias can be calculated as:

$$\begin{aligned}
\text{Bias}\left(\hat{\mu}(X) - \mu\right) &= \mathbb{E}_X\left(\hat{\mu}(X) - \mu\right) \\
&= \mathbb{E}_X\left(\frac{1}{N}\sum_{i=1}^{N} x_i\right) - \mu \\
&= \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_X\left(x_i\right) - \mu \\
&= \frac{1}{N}\left(\sum_{i=1}^{N} \mu\right) - \mu \\
&= \mu - \mu \\
&= 0
\end{aligned}$$

So the MLE of the mean of a Gaussian is unbiased.

Variance (σ^2) The estimator is given as

$$\hat{\sigma}^2(X) = \frac{1}{N}\sum_{i=1}^{N} \left(x_i - \hat{\mu}\right)^2$$

So the bias can be calculated as:

$$\begin{aligned}
\text{Bias}\left(\hat{\sigma}^2(X) - \sigma^2\right) &= \mathbb{E}_X\left(\hat{\sigma}^2(X) - \sigma^2\right) \\
&= \mathbb{E}_X\left(\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \hat{\mu}\right)^2\right) - \sigma^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_X\left(\left(x_i - \hat{\mu}\right)^2\right) - \sigma^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_X\left(x_i^2 - 2x_i\hat{\mu} + \hat{\mu}^2\right) - \sigma^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_X\left(x_i^2 - \hat{\mu}^2\right) - \sigma^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_X\left(x_i^2\right) - \mathbb{E}_X\left(\hat{\mu}^2\right) - \sigma^2 \\
&= \mathbb{E}_X\left(x^2\right) - \mathbb{E}_X\left(\hat{\mu}^2\right) - \sigma^2
\end{aligned}$$

With $\sigma^2 = \mathbb{E}(x^2) - \mathbb{E}(x)^2$ and $\hat{\sigma}^2 = \mathbb{E}(\hat{\mu}^2) - \mathbb{E}(\hat{\mu})^2$ and $\mathbb{E}(x) = \mathbb{E}(\hat{\mu})$:

$$\begin{aligned}
&= \left(\sigma^2 + \mathbb{E}(x)^2\right) - \left(\hat{\sigma}^2 + \mathbb{E}(\hat{\mu})^2\right) - \sigma^2 \\
&= \left(\sigma^2 + \mu^2\right) - \left(\hat{\sigma}^2 + \mu^2\right) - \sigma^2 \\
&= -\hat{\sigma}^2 \\
&= -\operatorname{Var}\left(\frac{1}{N}\sum_{i=1}^{N}x_i\right) \\
&= -\frac{1}{N^2}\operatorname{Var}\left(\sum_{i=1}^{N}x_i\right) \\
&= -\frac{1}{N^2}\sum_{i=1}^{N}\operatorname{Var}(x_i) \\
&= -\frac{1}{N}\operatorname{Var}(x) \\
&= -\frac{1}{N}\sigma^2
\end{aligned}$$

So the MLE of the variance of a Gaussian is biased (slightly below the actual variance).

8.2.4. Example: Regression

8.3. Model Selection and Occam's Razor

- The essence of Occams's Razor is: Always choose the simplest model that matches the data. Simplest means the model with the smallest complexity (e.g. the polynomial with the lowest degree).

- Model selection is a complex task.
- The whole data set has to be split into multiple data sets to avoid overfitting and to get better estimation for the prediction error:
  1. *Training Set* Fit parameters.
  2. *Validation Set* Choose the model class or single parameters.
  3. *Test Set* Estimate the prediction error of the trained model.

8.3.1. Cross Validation

- During *cross validation*, the whole data set $\mathcal{D}$ is split into K data sets $\mathcal{D}_\kappa$ and $K-1$ sets are used training and one data set is used for validation.
- This yields the following computations (where $\mathcal{M}_j$ is a model):

$$\boldsymbol{\theta}_k(\mathcal{M}_j) = \arg\min_{\boldsymbol{\theta}\in\mathcal{M}_j} \sum_{\kappa\neq k} \sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_\kappa} L_{f_{\boldsymbol{\theta}}}(\boldsymbol{x}_i, y_i)$$
$$L_k(\mathcal{M}_j) = \sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_\kappa} L_{f_{\boldsymbol{\theta}}}(\boldsymbol{x}_i, y_i)$$

- There exist multiple variations of cross validation:
  - *Exhaustive cross validation* Try all partitioning possibilities. $\implies$ Computationally expensive.
  - *Bootstrap cross validation* Randomly sample non-overlapping training/validation sets.

8.3.2. K-Fold Cross Validation

- Randomly partition the data set into K data sets, select one for validation and repeat this K times, each with a different validation set.
- Compute the validation loss in each iteration and choose the model with the lowest average validation loss:

$$\mathcal{M}^* = \arg\min_{\mathcal{M}} \frac{1}{K}\sum_{k=1}^{K} L_k(\mathcal{M})$$

- $L_k(\mathcal{M})$ is computed as defined in 8.3.1.
- *Leave-one-out cross-validation (LOOCV):* K is set to $K = N-1$, which yields a validation set size of 1.

8.3.3. Machine Learning Cycle

Figure 8.1 shows the general cycle of machine learning that can/must be repeated multiple times in order to get a good model.

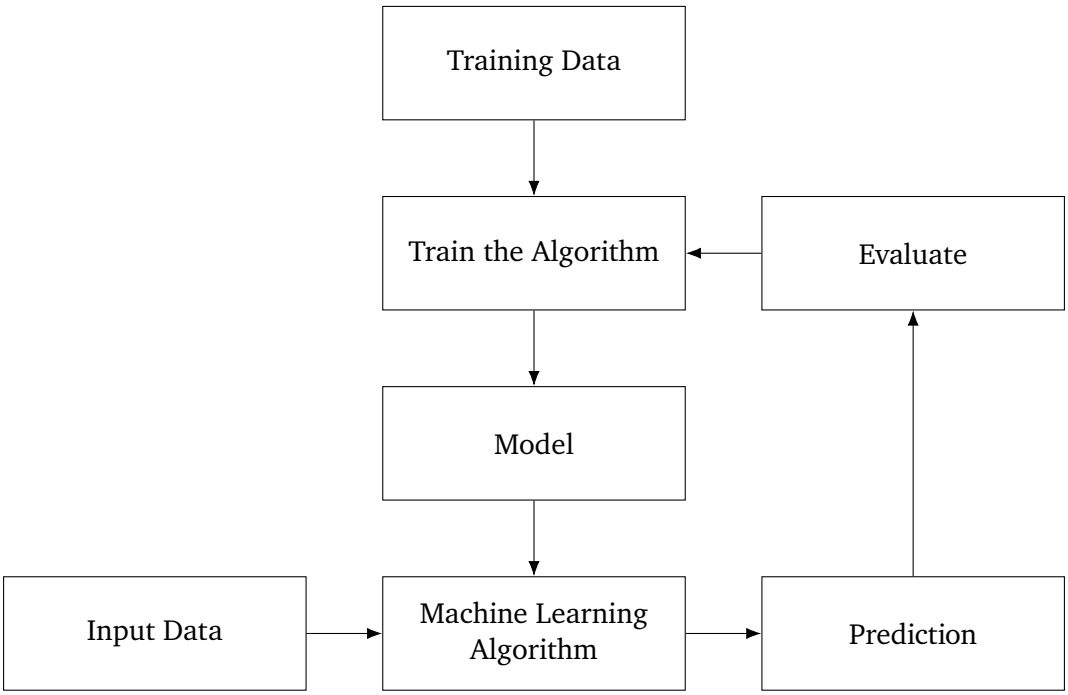

Figure 8.1.: Machine Learning Cycle

8.4. Wrap-Up

- Bias and variance of an estimator
- Bias-Variance tradeoff
- MVUE and BLUE
- Difference between unbiased and biased estimators
- Mimic test data evaluation using cross-validation

9. Regression

Regression is about to learn a mapping $f : I \to O$, $y = f(x; \theta)$ from input I to output O with the parameters $\theta \in \Theta$. The parameters are what needs to be "learned" and f represents the model that is trained. In regression, in output space O is continuous, e.g. $O = \mathbb{R}$ or $O = \mathbb{R}^2$ etc.

In general, the training data is given as pairs of in- and output values $\boldsymbol{x}_i, y_i$. This chapter will only cover the case $y_i \in \mathbb{R}$, but in general y_i can have multiple dimensions. Let $X \coloneqq \{ \boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \}$ and $Y \coloneqq \{ y_1, \cdots, y_n \}$ be the sets of the training input/output values.

Note: All of the following examples are based in the true function $f(x) = \sigma(x)\sin(x)$ with the sigmoid function $\sigma(x)$ and 50 sampled data points with a noise of $\mathcal{N}(0, 1)$. The true function is shown in figure 9.1.

9.1. Linear Regression

In *linear regression*, the function f to train is a linear function (called *regressor*)

$$y = \boldsymbol{x}^T \boldsymbol{w} + w_0$$

9.1.1. Least Squares Regression

- The linear $y_i = \boldsymbol{x}_i^T \boldsymbol{w} + w_0$ gives n linear equation, one for each training data pair.
- With $\hat{\boldsymbol{x}}_i \coloneqq \begin{bmatrix} \boldsymbol{x}_i \\ 1 \end{bmatrix}$ and $\hat{\boldsymbol{w}} \coloneqq \begin{bmatrix} \boldsymbol{w} \\ w_0 \end{bmatrix}$ the regressor can be written as $y_i = \hat{\boldsymbol{x}}_i \hat{\boldsymbol{w}}$.
- Using the matrices $\hat{X} = \left[\hat{\boldsymbol{x}}_i, \cdots, \hat{\boldsymbol{x}}_n \right]$ and the vector $\boldsymbol{y} = [y_1, \cdots, y_n]$ the complete problem can be summarized into one matrix-vector equation:

$$\hat{X}^T \hat{\boldsymbol{w}} = \boldsymbol{y}$$

This is an overdetermined linear equation system that therefore will most likely not yield a solution. So instead use least squares optimization and solve the (unbounded) optimization problem

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \left\| \hat{X} \boldsymbol{w} - \boldsymbol{y} \right\|^2$$

which yields the solution

$$\hat{\boldsymbol{w}} = \left(\hat{X} \hat{X}^T \right)^{-1} \hat{X} \boldsymbol{y}$$

using the left pseudo-inverse of $\hat{X}$.

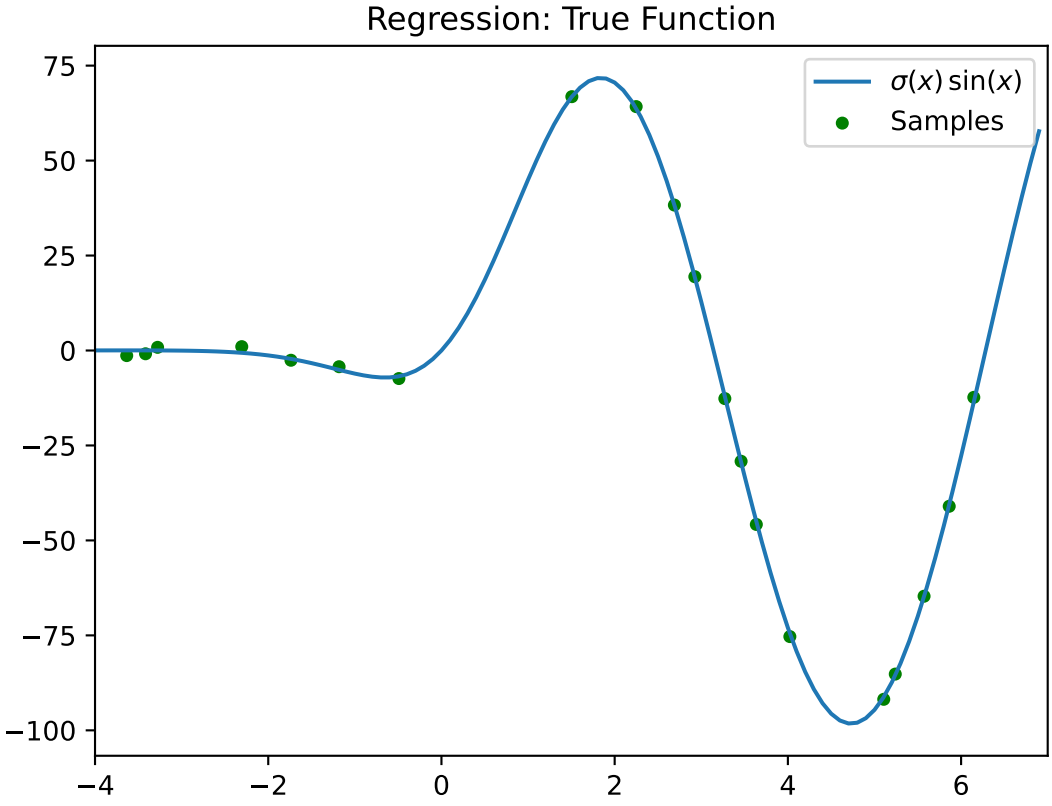

Figure 9.1.: Regression: True Function

Problems

- LSR depends on the inversion of a $D \times D$ matrix, where D is the dimension.
- Naive matrix version takes $\mathcal{O}(D^3)$ and is numerically instable.
- As D grows, other methods like gradient descent have to be taken into account.
- LSR indirectly assumes that the targets are Gaussians!

Regularized Least Squares Regression To regularize LSR, a *regularization term* λ can be added to the estimator, yielding the following minimization objective:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \frac{1}{2}\left\|\hat{X}^T\boldsymbol{w} - \boldsymbol{y}\right\|^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

Solving this yields the following solution:

$$\hat{\boldsymbol{w}} = \left(\hat{X}\hat{X}^T + \lambda I\right)^{-1}\hat{X}\boldsymbol{y}$$

> **Warning:** The regularization term assumes that both the noise and the targets are Gaussian distributed!

9.2. Generalized Linear Regression

Generalized linear regression and can learn arbitrary polynomials that are nonlinear w.r.t. to input variables $\boldsymbol{x}$. The regressor has the following general form:

$$y(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) = \sum_{i=0}^{M} w_i\phi_i(\boldsymbol{x})$$

with the *basis functions* $\phi_i(\cdot)$. Also, $\phi_0(\boldsymbol{x}) = 1$ is assumed for every $\boldsymbol{x}$.

- With basis functions like $\phi(x) = \begin{bmatrix}1 & x & x^2 & x^3\end{bmatrix}$, the regressor is nonlinear w.r.t. $\boldsymbol{x}$.
- But the model is still linear w.r.t. to the parameters $\boldsymbol{w}$, so the learning methods for linear regression can still be applied for polynomial regression (with small adjustments).
- Note that higher polynomials can easily lead to massive overfitting!

Assuming a Gaussian distribution and using $\boldsymbol{y} = \begin{bmatrix}y_1 \\ \vdots \\ y_n\end{bmatrix}$ and $\Phi = \begin{bmatrix}\phi(\boldsymbol{x}_1) & \cdots & \phi(\boldsymbol{x}_n)\end{bmatrix}$, the least squares solution for generalized linear regression is

$$\hat{\boldsymbol{w}} = \left(\Phi\Phi^T\right)^{-1}\Phi\boldsymbol{y}$$

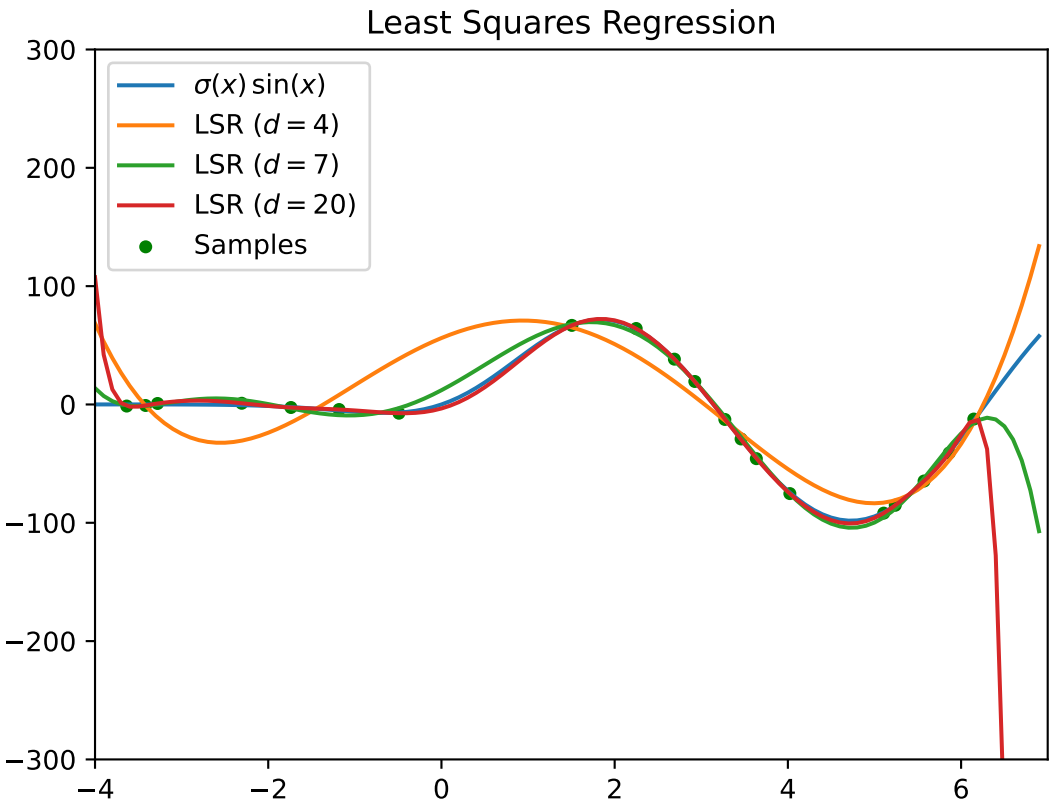

Figure 9.2.: Regression: Least Squares, Underfitting

Example This example used a nonlinear polynomial transformation $\phi_d(\cdot)$ that contains all polynomials up to the d-th degree, i.e. for $d = 2$:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$$

Figure 9.2 shows the results from three different polynomial degrees:

- $d = 4$ which underfits,
- $d = 7$ which fits just right and
- $d = 20$ which massively overfits.

9.3. Maximum Likelihood Approach

9.3.1. Probabilistic Regression

For probabilistic regression, two assumptions have to be made:

1. The target function values are generated by adding noise ϵ to the function estimate:

$$y = f(\boldsymbol{x}, \boldsymbol{w}) + \epsilon$$

2. The noise is a random variable that is Gaussian distributed (with the precision β and variance β^{-1}):

$$\epsilon \sim \mathcal{N}\left(0, \beta^{-1}\right) \quad \Longrightarrow \quad p\big(y \,|\, \boldsymbol{x}, \boldsymbol{w}, \beta\big) = \mathcal{N}\Big(y \,|\, f\big(\boldsymbol{x}, \boldsymbol{w}\big), \beta^{-1}\Big)$$

So y is now a random variable that is Gaussian distributed with the underlying probability distribution $p\big(y \,|\, \boldsymbol{x}, \boldsymbol{w}, \beta\big)$!

9.3.2. Maximum Likelihood Regression

Probabilistic regression gives the possibility to use well-known procedures like MLE for regression, called *maximum likelihood regression*.

Conditional Likelihood With the input data points $X = \big[\, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \,\big] \in \mathbb{R}^{D \times n}$ (dimensions D) and the output data points $\boldsymbol{Y} = [\, y_1, \cdots, y_n \,]$, the *conditional likelihood* can be formulated as (with the general linear regressor $\boldsymbol{w}^T \phi(\boldsymbol{x})$):

$$p\big(\boldsymbol{y} \,|\, X, \boldsymbol{w}, \beta\big) \overset{\text{i.i.d.}}{=} \prod_{i=1}^{n} \mathcal{N}\Big(y_i \,|\, f\big(\boldsymbol{x}_i, \boldsymbol{w}\big), \beta^{-1}\Big) \overset{\text{linear model}}{=} \prod_{i=1}^{n} \mathcal{N}\Big(y_i \,|\, \boldsymbol{w}^T \phi(\boldsymbol{x}_i), \beta^{-1}\Big)$$

The maximum likelihood is approach is to maximize the conditional w.r.t. $\boldsymbol{w}$ and β.

Maximization Using $\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$, $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$ and $\Phi = \big[\phi\big(\boldsymbol{x}_1\big) \quad \cdots \quad \phi\big(\boldsymbol{x}_n\big)\big]$, $\boldsymbol{w}$ and β can be estimated with the standard ML-estimation (take the derivative of the log-likelihood, set it to zero and solve for $\boldsymbol{w}$ or β, respectively). This yields the following estimators:

$$\begin{aligned} \boldsymbol{w}_{\text{ML}} &= \big(\Phi \Phi^T\big)^{-1} \Phi \boldsymbol{y} \\ \beta_{\text{ML}}^{-1} &= \frac{1}{n} \sum_{i=1}^{n} \Big(y_i - \boldsymbol{w}_{\text{ML}}^T \phi\big(\boldsymbol{x}_i\big)\Big)^2 \end{aligned}$$

Properties

- Maximum likelihood yields the same solution as squared errors, so LSR indirectly assumes that the targets are Gaussian distributed (not distribution free).
- But MLR can also estimate β, identifying how certain the estimation is about the result (bigger $\beta \rightarrow$ more certain as the variance β^{-1} gets less).

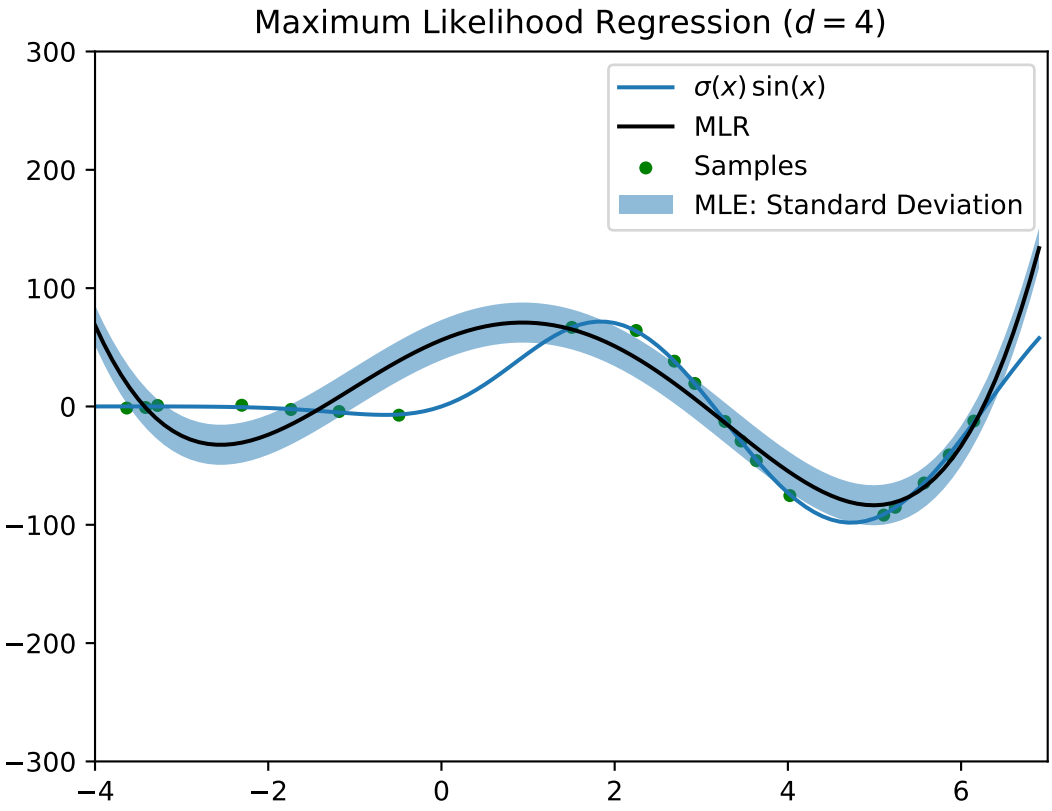

Figure 9.3.: Regression: Maximum Likelihood, Underfitting

Example This example used a nonlinear polynomial transformation $\phi_d(\cdot)$ that contains all polynomials up to the d-th degree, i.e. for $d = 2$:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$$

The following figures show the results for maximum likelihood regression, together with the standard deviation estimated with β_{ML}^{-1}:

- Figure 9.3 used $d = 4$ which slightly underfits,
- Figure 9.4 used $d = 7$ which seems to fit very good and finally
- Figure 9.5 used $d = 20$ which overfits.

Of course the estimation for the curve itself is the same as with least squares, as the equation for computing it is identical.

9.3.3. Loss Functions

- MLR yields a probability distribution $p(y \mid \boldsymbol{x}, \boldsymbol{w}, \beta)$ for the function value y.

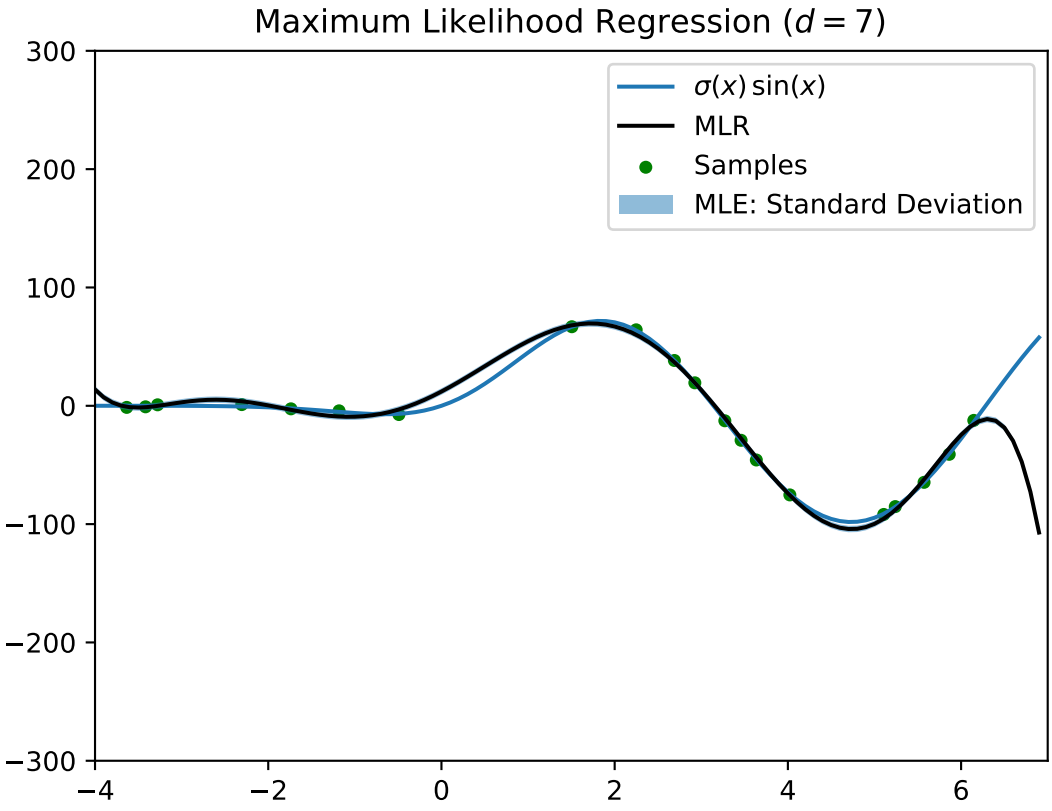

Figure 9.4.: Regression: Maximum Likelihood, Just Right

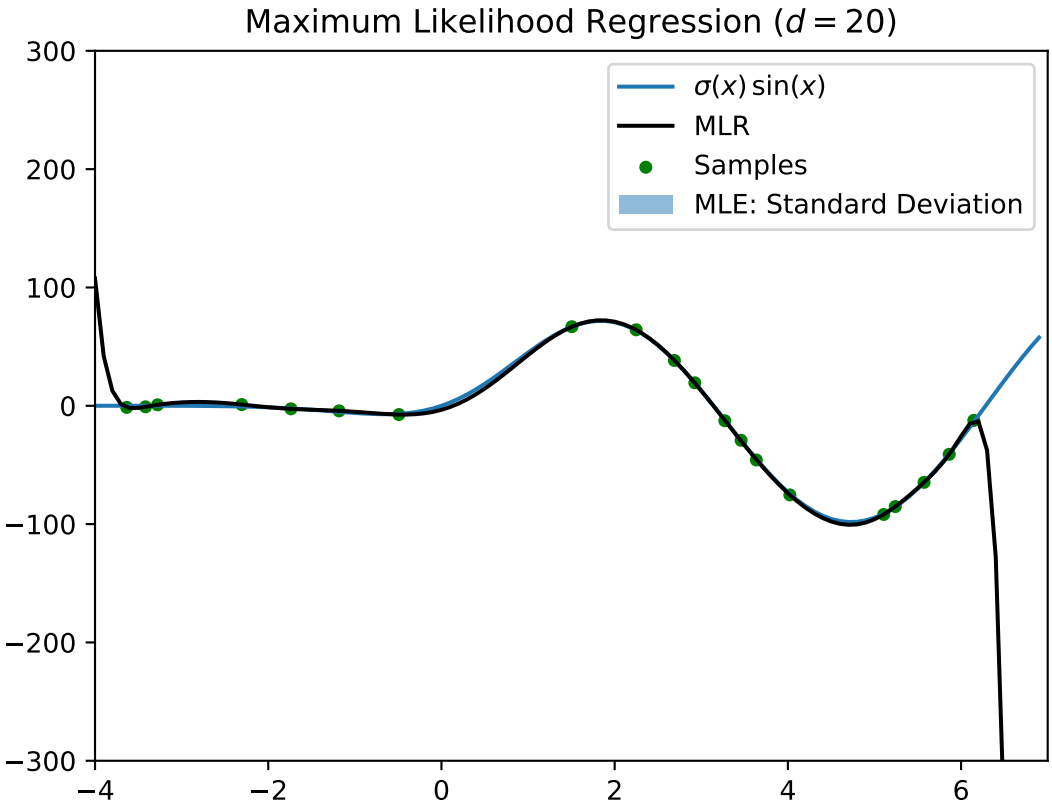

Figure 9.5.: Regression: Maximum Likelihood, Overfitting

- To actually estimate the function value y_t for a new data points $\boldsymbol{x}_t$, a *loss function*

$$L : \mathbb{R} \times R \to \mathbb{R}^+ : \Big(y_t, f\big(\boldsymbol{x}_t\big)\Big) \mapsto L\Big(y_t, f\big(\boldsymbol{x}_t\big)\Big)$$

  is needed.

- Then, the expected loss has to be minimized (w.r.t. $f(\boldsymbol{x})$):

$$\mathbb{E}_{\boldsymbol{x},y\sim p(\boldsymbol{x},y)}(L) = \iint L\Big(y, f\big(\boldsymbol{x}\big)\Big) p\big(\boldsymbol{x}, y\big)\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}y$$

The simplest case is the squared loss function $L\big(y, f(\boldsymbol{x})\big) = \big(y - f(\boldsymbol{x})\big)^2$ which yields the solution

$$f(\boldsymbol{x}) = \mathbb{E}_{y\sim p(y\,|\,\boldsymbol{x})}(y) = \mathbb{E}(y\,|\,\boldsymbol{x})$$

So, under the squared error, the *optimal regression function* is just the mean of the posterior distribution $p(y\,|\,\boldsymbol{x})$ (also called *mean prediction*).

For the generalized linear regression function, this is just the value of the function:

$$f(\boldsymbol{x}) = \mathbb{E}(y\,|\,x) = \boldsymbol{w}^T\phi(\boldsymbol{x})$$

9.4. Bayesian Linear Regression

- In *Bayesian linear regression*, a prior is placed on the parameters $\boldsymbol{w}$ to tame the instabilities and to reduce overfitting.
- As in all Bayesian interpretations, it is based on Bayes rule.
- Also, Bayesian linear regression no more produces a single value for $\boldsymbol{w}$, but rather a probability distribution over the parameters.
- Idea: Put a Gaussian prior on $\boldsymbol{w}$ with a spherical covariance matrix with precision α (variance α^{-1})

$$\boldsymbol{w} \sim p(\boldsymbol{w}\,|\,\alpha) = \mathcal{N}\left(\boldsymbol{w}\,|\,\boldsymbol{0}, \alpha^{-1}I\right)$$

  the mean can be set to a different value, but zero makes the following computations easier.

- Using this prior, the posterior distribution becomes:

$$p(\boldsymbol{w}\,|\,X, y, \alpha, \beta) \propto p(\boldsymbol{y}\,|\,X, \boldsymbol{w}, \beta)\, p(\boldsymbol{w}\,|\,\alpha) = p(\boldsymbol{y}\,|\,X, \boldsymbol{w}, \beta)\, \mathcal{N}\left(\boldsymbol{w}\,|\,\boldsymbol{0}, \alpha^{-1}I\right)$$

9.4.1. Maximum A-Posteriori (MAP)

Using *maximum a-posteriori estimation* (take the derivative of the log-posteriori, set it to zero and solve for the parameter of interest), $\boldsymbol{w}$ can be estimated as

$$\boldsymbol{w}_{\text{MAP}} = \left(\Phi\Phi^T + \frac{\alpha}{\beta}I\right)^{-1}\Phi\boldsymbol{y}$$

The prior has the effect that it regularizes the pseudo-inverse via the parameter $\frac{\alpha}{\beta}$. This is also called *ridge regression*.

Sometimes, only the ridge parameter $\lambda = \frac{\alpha}{\beta}$ is given, as known of the regularized least squares regression.

9.4.2. Full Bayesian Regression

- The actual value of $\boldsymbol{w}$ is not really a point of interest, as the goal is to predict a function value rather than a parameter value.
- The idea of full Bayesian regression is to remove $\boldsymbol{w}$ by marginalizing over it:

$$p(y_t \mid \boldsymbol{x}_t, X, \boldsymbol{y}) = \int p(y_t, \boldsymbol{w} \mid \boldsymbol{x}_t, X, \boldsymbol{y}) \, \mathrm{d}\boldsymbol{w} = \int \underbrace{p(y_t \mid \boldsymbol{w}, \boldsymbol{x}_t)}_{\text{regression model}} \underbrace{p(\boldsymbol{w} \mid X, \boldsymbol{y})}_{\text{posterior}} \mathrm{d}\boldsymbol{w}$$

  where y_t is the predicted value, $\boldsymbol{x}_t$ is the test input, X are the training data points and $\boldsymbol{y}$ are the training function values.
- The marginalized probability distribution $p(y_t \mid \boldsymbol{x}_t, X, \boldsymbol{y})$ is called the *predictive distribution*.
- For Gaussian distributions, this is solvable in a closed form, leading to *Gaussian processes*.

Gaussians For Gaussians, the predictive distribution is given as

$$p(y_t \mid \boldsymbol{x}_t, X, \boldsymbol{y}) = \mathcal{N}\left(\boldsymbol{y}_t \mid \mu(\boldsymbol{x}_t), \sigma^2(\boldsymbol{x}_t)\right)$$

with the parameters

$$\mu(\boldsymbol{x}_t) = \phi^T(\boldsymbol{x}_t) \left(\frac{\alpha}{\beta} I + \Phi\Phi^T\right)^{-1} \Phi^T \boldsymbol{y}$$
$$\sigma^2(\boldsymbol{x}_t) = \frac{1}{\beta}\phi^T(\boldsymbol{x}_t) \left(\alpha I + \beta\Phi\Phi^T\right)^{-1} \phi(\boldsymbol{x}_t)$$

So the mean and variance are state dependent!

This leads to Gaussian processes (see section 9.6) getting more certain about the estimate as more data points are taken into account.

Example This example used a nonlinear polynomial transformation $\phi_d(\cdot)$ that contains all polynomials up to the d-th degree, i.e. for $d = 2$:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix}$$

The following figures show the results for full Bayesian regression with the noise parameter $\alpha^{-1} = 1$ and the prior $\beta 0.01$, together with the standard deviation.

- Figure 9.6 used $n = 2$ samples does not fit the function very well and has a very high variance,
- Figure 9.7 used $n = 8$ samples, fitting the function better with still a high variance and
- Figure 9.8 used $n = 20$ (all) samples fitting the function really well with high variance on both sides.

This is expected as regression gets more accurate the more data points are used. Note that in this example, the polynomial degree stayed fixed!

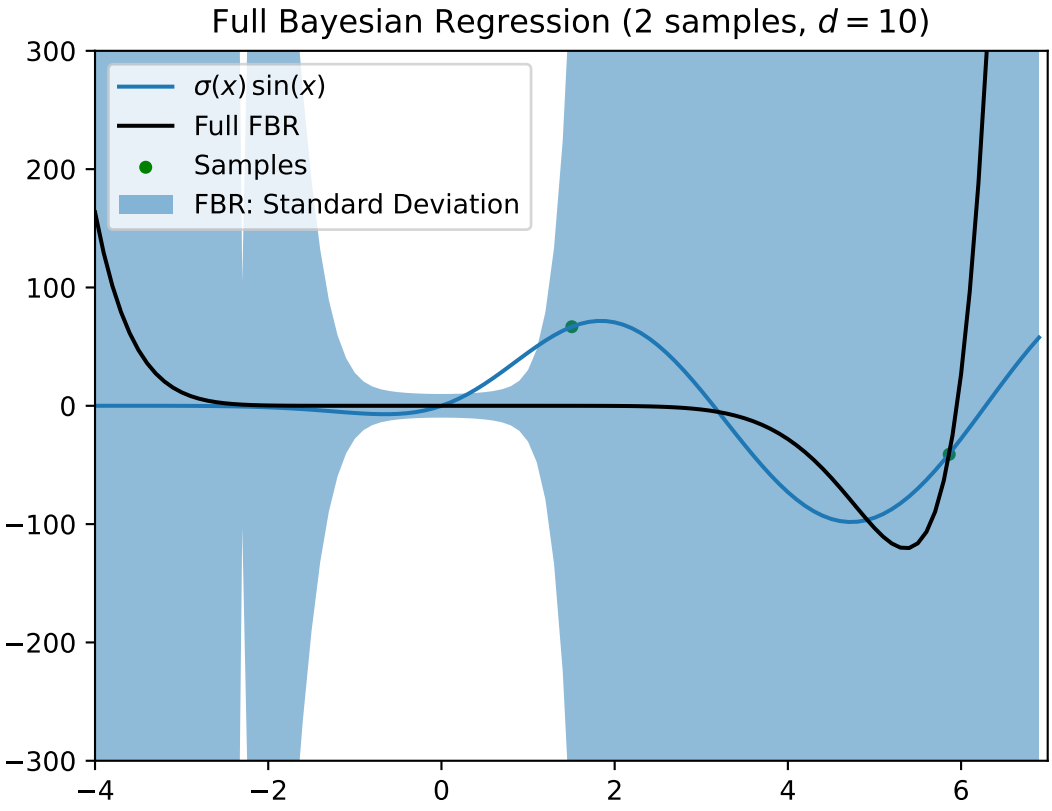

Figure 9.6.: Regression: Full Bayesian Regression (2 Samples)

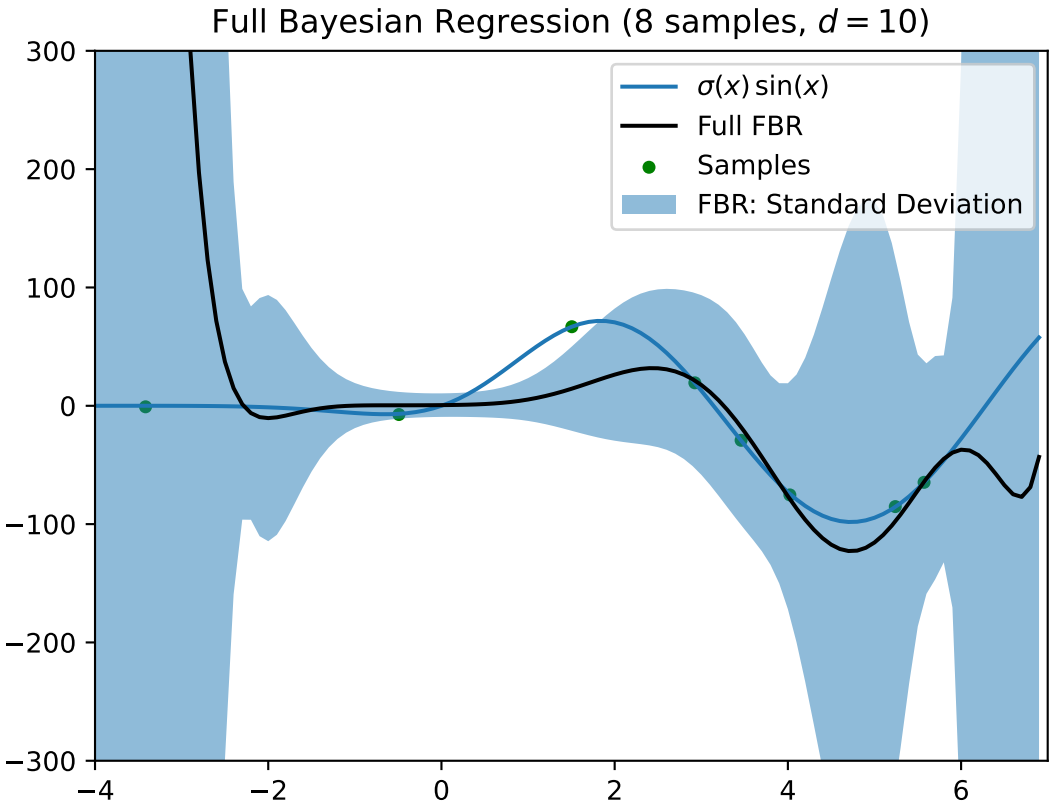

Figure 9.7.: Regression: Full Bayesian Regression (8 Samples)

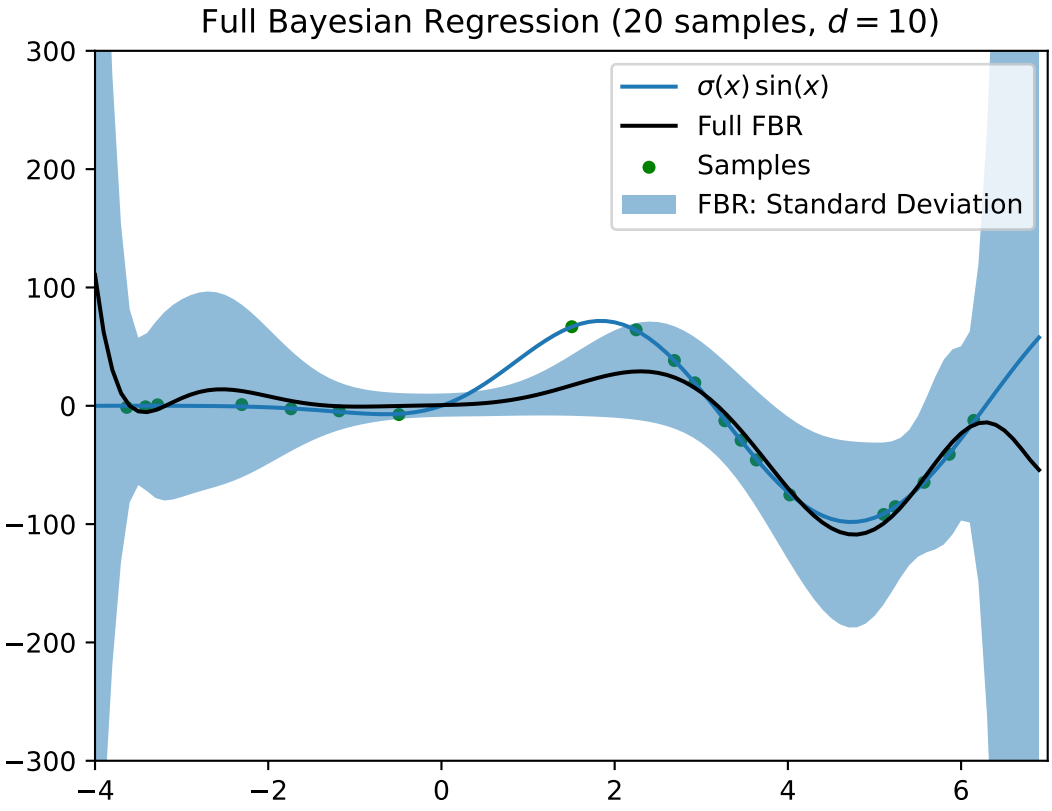

Figure 9.8.: Regression: Full Bayesian Regression (All Samples)

9.5. Kernel Regression

A *kernel* is an inner product of feature vectors (or feature transformations)

$$K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{y})$$

which is symmetric ($K(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{y}, \boldsymbol{x})$).

Example kernels:

- Stationary kernels:
  $$K(\boldsymbol{x}, \boldsymbol{y}) = \hat{K}(\boldsymbol{x} - \boldsymbol{y})$$
- Linear kernel: $K(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y}$
- Homogeneous kernels:
  $$K(\boldsymbol{x}, \boldsymbol{y}) = \hat{K}\big(|\boldsymbol{x} - \boldsymbol{y}|\big)$$

Working with kernels instead of handcrafted features has a lot of advantages:

- Can work entirely in the features space with the help of kernels.
- Regression can even consider infinite feature spaces (e.g. with the Gaussian RBF kernel).
- Many algorithms can be derived from the dual representation.
- Many old problems of RBFs (how many kernels, which metric, etc.) can be solved in a principled way.

But: Kernel regression requires the inversion of a $N \times N$ matrix, where N is the number of samples. This can be very costly!

Example Figure 9.9 shows kernel regression with an RBF kernel with the bandwidth $\sigma^2 = 0.01$, performing not so good. Figure 9.10 also uses an RBF kernel but with the bandwidth $\sigma^2 = 1$, performing much better. Both are using a ridge parameter $\lambda 0.01$.

9.5.1. Dual Representation of Regression

The primal formulation for (regularized) regression is

$$J(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} \big(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i\big)^2 + \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{w}$$

taking the gradient w.r.t. $\boldsymbol{w}$, setting it to zero and solving for $\boldsymbol{w}$:

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_{i=1}^{N} \big(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i\big)\phi(\boldsymbol{x}_i) + \lambda \boldsymbol{w} \stackrel{!}{=} 0$$

$$\implies \qquad \boldsymbol{w} = -\frac{1}{\lambda} \sum_{i=1}^{N} \big(\boldsymbol{w}^T \phi(\boldsymbol{x}_i) - y_i\big)\phi(\boldsymbol{x}_i) = \sum_{i=1}^{N} a_i \phi(\boldsymbol{x}_i) = \Phi^T \boldsymbol{a}$$

with $\Phi = \begin{bmatrix} \phi(\boldsymbol{x}_1)^T & \cdots & \phi(\boldsymbol{x}_N)^T \end{bmatrix} \in \mathbb{R}^{N \times D}$. Thus, $\boldsymbol{w}$ is a linear combination of the features $\phi(\boldsymbol{x}_i)$! The dual representation then focuses on solving for $\boldsymbol{a}$, not $\boldsymbol{w}$.

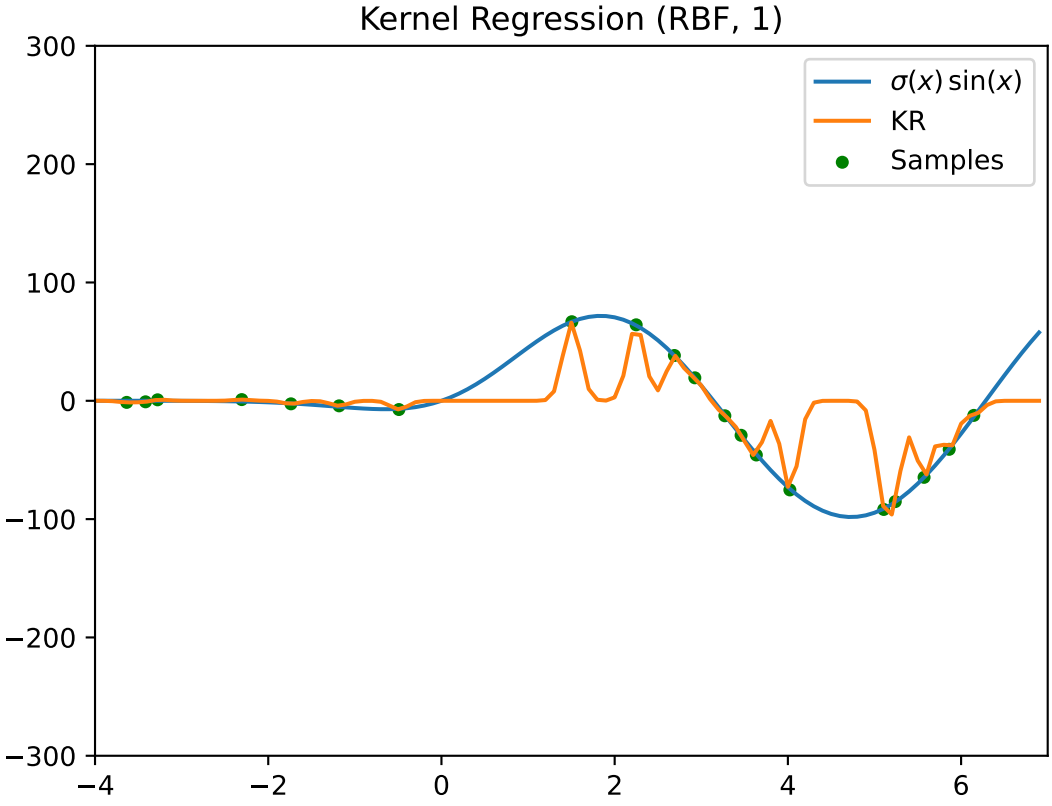

Figure 9.9.: Regression: Kernel Regression (RBF, $\sigma^2 = 0.01$)

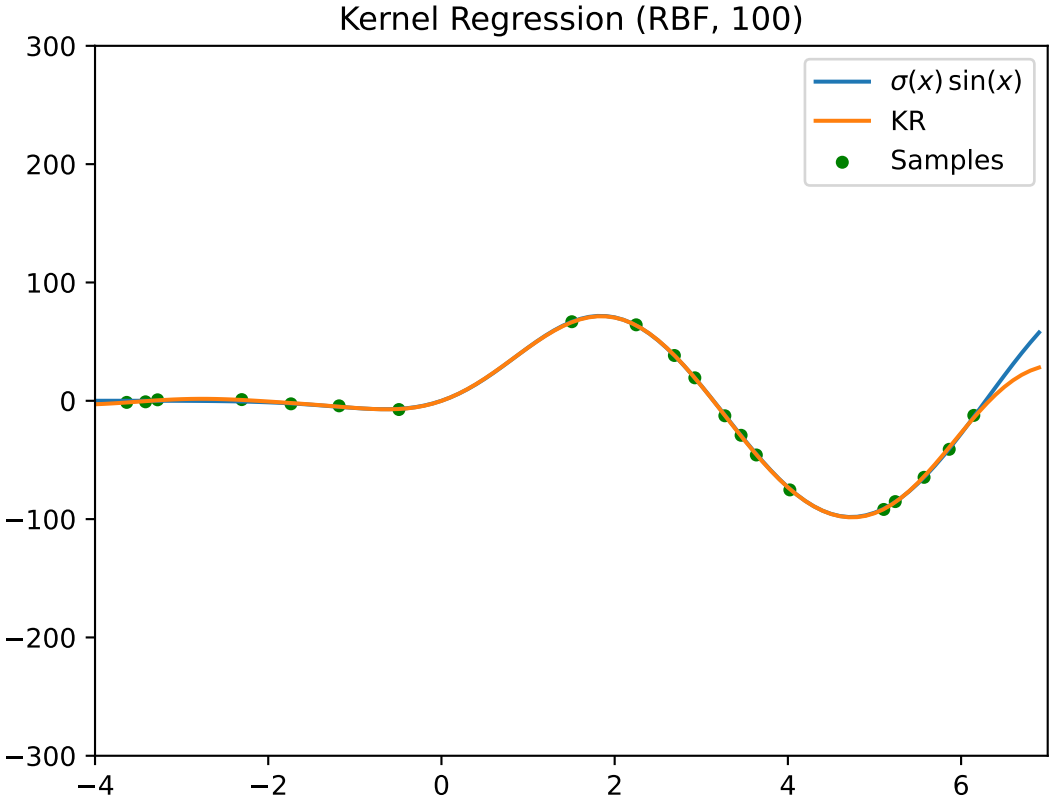

Figure 9.10.: Regression: Kernel Regression (RBF, $\sigma^2 = 1$)

Inserting this into the cost function yields the dual formulation:

$$
\begin{aligned}
& J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) - y_i\right)^2 + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w} \\
\implies \quad & \tilde{J}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(\boldsymbol{a}^T\Phi\phi(\boldsymbol{x}_i) - y_i\right)^2 + \frac{\lambda}{2}\boldsymbol{a}^T\Phi\Phi^T\boldsymbol{a} \\
& = \frac{1}{2}\sum_{i=1}^{N}\left(\boldsymbol{a}^T\Phi\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T\Phi^T\boldsymbol{a} - 2\boldsymbol{a}^T\Phi\phi(\boldsymbol{x}_i)y_i + y_i^2\right) + \frac{\lambda}{2}\boldsymbol{a}^T\Phi\Phi^T\boldsymbol{a} \\
& = \frac{1}{2}\sum_{i=1}^{N}\boldsymbol{a}^T\Phi\phi(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i)^T\Phi^T\boldsymbol{a} - \sum_{i=1}^{N}\boldsymbol{a}^T\Phi\phi(\boldsymbol{x}_i)y_i + \frac{1}{2}\sum_{i=1}^{N}y_i^2 + \frac{\lambda}{2}\boldsymbol{a}^T\Phi\Phi^T\boldsymbol{a} \\
& = \frac{1}{2}\boldsymbol{a}^T\Phi\Phi^T\Phi\Phi^T\boldsymbol{a} - \boldsymbol{a}^T\Phi\Phi^T\boldsymbol{y} + \frac{1}{2}\boldsymbol{y}^T\boldsymbol{y} + \frac{\lambda}{2}\boldsymbol{a}^T\Phi\Phi^T\boldsymbol{a}
\end{aligned}
$$

Let $\hat{K} \coloneqq \Phi\Phi^T$ be the *Gram matrix* with $\hat{K} \coloneqq K(\boldsymbol{x}_i, \boldsymbol{x}_j)$:

$$
= \frac{1}{2}\boldsymbol{a}^T\hat{K}\hat{K}\boldsymbol{a} - \boldsymbol{a}^T\hat{K}\boldsymbol{y} + \frac{1}{2}\boldsymbol{y}^T\boldsymbol{y} + \frac{\lambda}{2}\boldsymbol{a}^T\hat{K}\boldsymbol{a}
$$

Now solve the dual problem for $\boldsymbol{a}$ by taking the derivative and set it to zero:

$$
\begin{aligned}
& \tilde{L}(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\hat{K}\hat{K}\boldsymbol{a} - \boldsymbol{a}^T\hat{K}\boldsymbol{y} + \frac{1}{2}\boldsymbol{y}^T\boldsymbol{y} + \frac{\lambda}{2}\boldsymbol{a}^T\hat{K}\boldsymbol{a} \\
\implies \quad & \frac{\partial\tilde{L}(\boldsymbol{a})}{\partial\boldsymbol{a}} = \hat{K}\hat{K}\boldsymbol{a} - \hat{K}\boldsymbol{y} + \frac{\lambda}{2}\hat{K}\boldsymbol{a} = \hat{K}(\hat{K}\boldsymbol{a} - \boldsymbol{y} + \lambda\boldsymbol{y}) \stackrel{!}{=} 0 \\
\implies \quad & \boldsymbol{a} = (\hat{K} + \lambda I)^{-1}\boldsymbol{y}
\end{aligned}
$$

As of the definition of the Gram matrix $\hat{K}$, is is positive semi-definite, thus $\hat{K}^{-1}$ exists.

A prediction can then be computed as

$$
y(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b = \boldsymbol{a}^T\Phi\phi(\boldsymbol{x}) = \hat{k}(\boldsymbol{x})^T(\hat{K} + \lambda I)^{-1}\boldsymbol{y}
$$

with $\hat{k}(\boldsymbol{x}) = \begin{bmatrix}K(\boldsymbol{x}, \boldsymbol{x}_1) & \cdots & K(\boldsymbol{x}, \boldsymbol{x}_N)\end{bmatrix}^T$.

So all computations can be expressed in terms of the kernel function!

9.5.2. Useful Kernels

Polynomial Kernels of Degree d

$$
K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y})^d
$$

Gaussian Kernel Also known as *Radial Basis Function* (RBF).

$$
K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left\{-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\right\}
$$

This kernel has a feature space with infinity number of radial functions!

The parameter σ^2 is the *bandwidth* of the kernel. This is basically the inverse of the "variance" of the kernel and a measure of similarity. If it is small, far away points will be considered similar while if it is big, nearby points will be considered more similar.

9.6. Gaussian Processes Regression

A *Gaussian process* (GP) is a probability distribution over functions $y(\boldsymbol{x})$ such that any finite set of function values evaluates at some input is jointly Gaussian distributed. A Gaussian process is fully specified by the 2nd order statistics (mean and covariance).

- The *prior mean function* is the expected function before observing any data,
- The *covariance function* encodes some structural assumptions (e.g. smoothness) (e.g. multivariate Gaussian kernel).

Thus, a GP is fully defined by

$$\begin{aligned} \mathbb{E}(\boldsymbol{y}) &= \mathbb{E}(\Phi \boldsymbol{w}) = \Phi \mathbb{E}(\boldsymbol{w}) = 0 \\ \mathbb{E}\big(y(\boldsymbol{x}_1) y(\boldsymbol{x}_j)\big) &= K(\boldsymbol{x}, \boldsymbol{y}) \end{aligned}$$

9.6.1. Regression

Assume the generative model to have some noise ϵ that is Gaussian distributed with $\epsilon \sim \mathcal{N}(0, \beta^{-1})$:

$$t_i = y(\boldsymbol{x}_i) + \epsilon$$

This makes y a random variable that is also Gaussian distributed with

$$p(t_i \,|\, y_i) = \mathcal{N}(t_i \,|\, y_i, \beta^{-1})$$

The kernel function that determines K is typically chosen to express the property that, for similar points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the corresponding values $y(\boldsymbol{x}_i)$ and $y(\boldsymbol{x}_j)$ will be more strongly correlated that for dissimilar points. The definition of similarity highly depends on the application.

9.6.2. Function Value Prediction

- Prior over functions (GP): $p(y)$
- Likelihood (measurement/noise model): $p(t \,|\, y)$
- Posterior over function via Bias theorem:

$$p(y \,|\, t) = \frac{p(t \,|\, y)\, p(y)}{p(t)}$$

Given a training set $\boldsymbol{t}_n = \begin{bmatrix} t_1 & \cdots & t_n \end{bmatrix}^T$ with corresponding $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$, the goal is to predict a new t_{n+1} for x_{n+1}.

Approach: Evaluate the predictive distribution

$$p(t_{n+1} \,|\, x_{n+1}, t_{1:n}, x_{1:n})$$

Remember that GP assumes that $p(t_1, \cdots, t_n, t_{n+1})$ is jointly Gaussian distributed, so the predictive distribution is also Gaussian distributed.

Assume that $\boldsymbol{x}$ is Gaussian distributed and it can be partitioned into two disjoint subsets $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$, the distribution can be rewritten in term of the mean and the covariance matrix of $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$:

$$
\begin{aligned}
p(\boldsymbol{x}) &= \mathcal{N}\left(\boldsymbol{x} \,|\, \boldsymbol{\mu}, \Sigma\right) \\
\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_a \\ \boldsymbol{x}_b \end{bmatrix} \quad \boldsymbol{\mu} &= \begin{bmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{bmatrix} \quad \Sigma = \begin{bmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{bmatrix}
\end{aligned}
$$

The conditional distribution is also Gaussian:

$$
\begin{aligned}
p(\boldsymbol{x}_a \,|\, \boldsymbol{x}_b) &= \mathcal{N}\left(\boldsymbol{x}_a \,|\, \boldsymbol{\mu}_{a\,|\,b}, \Sigma_{a\,|\,b}\right) \\
\boldsymbol{\mu}_{a\,|\,b} &= \boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\boldsymbol{x}_b - \boldsymbol{\mu}_b) \\
\Sigma_{a\,|\,b} &= \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}
\end{aligned}
$$

Thus, the predictive distribution can be expressed as:

$$
\begin{aligned}
p(\boldsymbol{t}_{n+1}) &= \mathcal{N}\left(\boldsymbol{t}_{n+1} \,|\, 0, C_{n+1}\right) \\
C_{n+1} &= \begin{bmatrix} C_n & \boldsymbol{k} \\ \boldsymbol{K} & c \end{bmatrix} \\
\boldsymbol{k} &= \begin{bmatrix} K(\boldsymbol{x}_1, \boldsymbol{x}_{n+1}) \\ \vdots \\ K(\boldsymbol{x}_n, \boldsymbol{x}_{n+1}) \end{bmatrix} \\
c &= K(\boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+1}) + \beta^{-1}
\end{aligned}
$$

Yielding the following prediction equations:

$$
\begin{aligned}
m(\boldsymbol{x}_{n+1}) &= \boldsymbol{K}^T C_N^{-1} \boldsymbol{t} \\
\sigma^2(\boldsymbol{x}_{n+1}) &= c - \boldsymbol{K}^T C_N^{-1} \boldsymbol{K}
\end{aligned}
$$

This gives an estimation for the function value as well as gauges the uncertainty about that estimate!

Example

9.6.3. Conclusion

- The computational complexity for building the model is $\mathcal{O}(N^3)$ and for predicting one function value it is $\mathcal{O}(N^2)$ (for the variance).
- The key advantage of GPR is that it is non-parametric and probabilistic.
- Naive implementations can deal with 10 000 to 20 000 data points, while advanced methods (e.g. sparse GPs) can deal with far more than 50 000 data points.
- Hyperparameter (parameters for the kernel/covariance function) optimization is really important, e.g. for the squared-exponential kernel:

$$
K(\boldsymbol{x}, \boldsymbol{y}) = \sigma_f^2 \exp\left\{ -\frac{(\boldsymbol{x} - \boldsymbol{x})^2}{2l^2} \right\} + \sigma_n^2 \delta_{ij}
$$

where σ_f^2 is the signal variance, l is the length-scale and σ_n^2 is the noise variance.

- Gaussian processes are Bayesian approaches to regression with (possible infinity) feature spaces.
- The resulting prediction equations are straightforward obtained in a closed form because of Gaussian properties.
- The hyperparameter optimization is more complex and expensive.
- While it is very computationally expensive, it is one of the most used approaches to statistical learning for regression.

9.7. Wrap-Up

- Formulation of a linear regression problem
- Different methods to perform linear regression (least squares, maximum likelihood, Bayesian)
- Derivation of the equations for different methods
- Influence of a prior distribution over the parameters to overfitting
- Kernels, construction and benifits
- Derivation of the dual formulation and pros/cons
- Gaussian processes and made assumptions
- GP closed form
- Regression with GPs yields mean and variance
- Kernels to not scale with data

10. Classification

In *classification*, the goal is to find a mapping $f : I \to O$ that maps the input space I onto a discrete (and mostly finite) output space O, called *classes*.

As seen in Bayesian decision theory, this breaks down into finding the a-posteriori probability (posterior) of the class C_k given an observation (feature) x

$$p(C_k \mid x) = \frac{p(x \mid C_k)\, p(C_k)}{p(x)} = \frac{p(x \mid C_k)\, p(C_k)}{\sum_j p(x \mid C_j)\, p(C_j)}$$

and then device for class k iff $p(C_k \mid x) > p(C_l \mid x)$ for all $l \neq k$. A classifier that obeys this rule is called a *Bayes optimal classifier*. See chapter 5 for more details.

Note: All of the following examples use a dataset of 250 data points per class that were generated by a mixture of two-dimensional multivariate Gaussians, plotted in figure 10.1 (for linearly separable data).

10.1. Generative vs. Discriminative

There are essentially two different views to solve the classification problem:

Generative Model the class-condition distributions $p(x \mid C_k)$ and use Bayes rule and some prior to compute the class posterior.

Discriminative Model the class posterior $p(C_k \mid x)$ directly, e.g. by separating the data points using a function. These types of models only care about getting the classification right and not whether the class-conditional fits well.

10.2. Discriminant Functions

- *Discriminant functions* model the decision boundary and directly without modeling the densities while still minimizing the error probability.
- In comparison with generative models, discriminative models have the advantage that they are not so sensitive to outliers which the class-conditional-based have to consider when estimating the class-condition distribution, even if they do not matter at the end.
  - This reduces the complexity of the overall model once the model has learned where to place the decision boundary.
  - This shall not mean that such classifiers are inherently superior to probabilistic ones (e.g. they cannot take priors into account)!
- For two classes, decide for class C_1 iff $y_1(x) > y_2(x)$. This is equivalent to defining a function $y(x) = y_1(x) - y_2(x)$ and decide for class C_1 iff $y(x) > 0$.

Figure 10.1.: Classification: Example Data (Linear Separable)

- Some discriminant functions are directly given from a Bayes classifier:

$$\begin{aligned} y_k(x) &= p(C_k \mid x) \\ y_k(x) &= p(x \mid C_k)\, p(C_k) \\ y_k(x) &= \ln\big(p(x \mid C_k)\big) + \ln\big(p(C_k)\big) \end{aligned}$$

  The logarithm in the last step is applicable because $\ln(x)$ is a strictly rising function and, for distributions of the exponential family, drastically reduces the computational overhead and thus reduces numeric instabilities.

10.2.1. Multiple Classes

- Normal multi-class classifiers based on binary (two-class) decisions may lead to ambiguities ("regions of uncertainty").
- A better solution is to have multiple discriminant functions $y_1, \cdots, y_k$ and choose C_k iff $y_k(x) > y_l(x)$ for all $l \neq k$.
- Using this decision rule and linear discriminant functions, the decision regions are connected and convex which removes the "regions of uncertainty" and the ambiguities.

10.2.2. Linear Discriminant Functions

- In *linear discriminant functions*, the decision boundaries are hyperplanes defined by a linear function

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

  where $\boldsymbol{w}$ is the normal vector and w_0 is the offset.

Linear Separability

- Not all data points may be separable by a linear function (e.g. if the data points overlap).
- Figure 10.2 shows two cases where the first is linearly separable and the second is not.

10.3. Fisher Discriminant Analysis

10.3.1. Least Squares Classification

- In *least squares classification*, the discriminant function shall output the values $y(\boldsymbol{x}) = +1$ or $y(\boldsymbol{x}) = -1$, indicating that $\boldsymbol{x}$ belongs to class C_1 or class C_2, respectively.
- With training data inputs $X = \{\, \boldsymbol{x}_1 \in \mathbb{R}^d, \cdots, \boldsymbol{x}_n \,\}$ and training outputs $Y = \{\, y_1 \in \{\, +1, -1 \,\}, \cdots, y_n \,\}$, this yields an overdetermined equation system

$$y_i = \boldsymbol{w}^T \boldsymbol{x}_i + w_0$$

  for all data pairs $(\boldsymbol{x}_i, y_i)$.

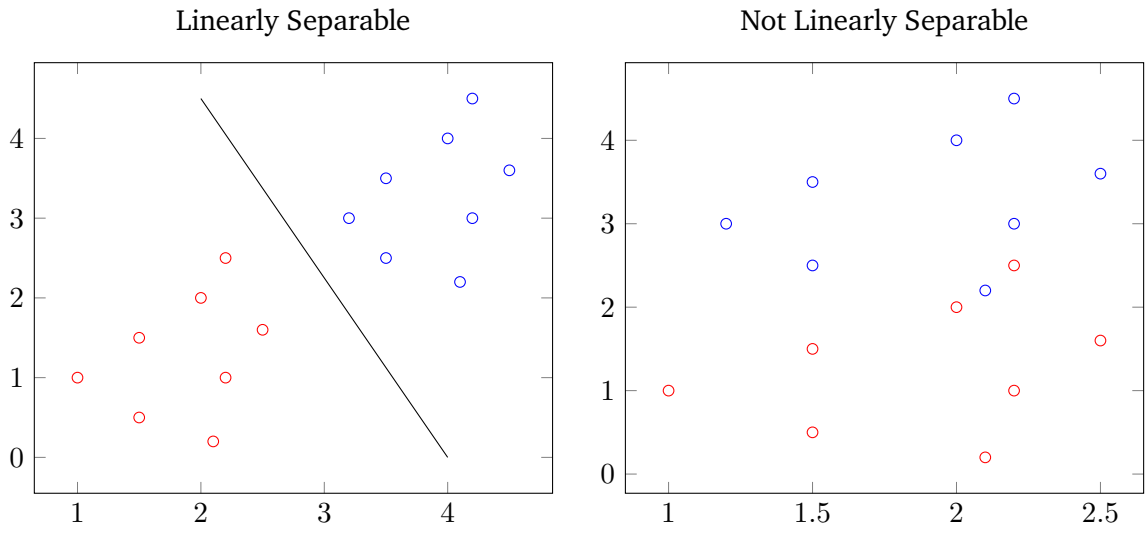

Figure 10.2.: Linear Separability

- The formula can be rewritten in matrix-vector notation yielding just one formula to express everything:

$$\begin{aligned} \hat{\boldsymbol{x}}_i &:= \begin{bmatrix} \boldsymbol{x}_i & 1 \end{bmatrix}^T \\ \hat{\boldsymbol{w}} &:= \begin{bmatrix} \boldsymbol{w} & w_0 \end{bmatrix}^T \\ \hat{X} &:= \begin{bmatrix} \hat{\boldsymbol{x}}_1 & \cdots & \hat{\boldsymbol{x}}_n \end{bmatrix} \in \mathbb{R}^{d \times n} \\ \boldsymbol{y} &:= \begin{bmatrix} y_1 & \cdots & y_n \end{bmatrix}^T \\ \implies \quad \boldsymbol{y} &= \hat{X}^T \hat{\boldsymbol{w}} \end{aligned}$$

- This is an overdetermined equation system and thus not solvable in general, so instead of solving it the squared error of the equation system is minimized:

$$\begin{aligned} \hat{\boldsymbol{w}}^* &= \arg\min_{\hat{\boldsymbol{w}}} \|\hat{X}^T \hat{\boldsymbol{w}} - \boldsymbol{y}\|^2 \\ &= \arg\min_{\hat{\boldsymbol{w}}} \left(\hat{X}^T \hat{\boldsymbol{w}} - \boldsymbol{y}\right)^T \left(\hat{X}^T \hat{\boldsymbol{w}} - \boldsymbol{y}\right) \\ &= \arg\min_{\hat{\boldsymbol{w}}} \hat{\boldsymbol{w}}^T \hat{X} \hat{X}^T \hat{\boldsymbol{w}} - 2\boldsymbol{y}^T \hat{X}^T \hat{\boldsymbol{w}} + \boldsymbol{y}^T \boldsymbol{y} \end{aligned}$$

Take the derivative w.r.t. $\hat{\boldsymbol{w}}$ and set it to zero:

$$\begin{aligned} & 0 \stackrel{!}{=} \frac{\partial}{\partial \hat{\boldsymbol{w}}} \left(\hat{\boldsymbol{w}}^T \hat{X} \hat{X}^T \hat{\boldsymbol{w}} - 2\boldsymbol{y}^T \hat{X}^T \hat{\boldsymbol{w}} + \boldsymbol{y}^T \boldsymbol{y}\right) \\ \iff \quad & 0 = \hat{X} \hat{X}^T \hat{\boldsymbol{w}} + \hat{X}^T \hat{X} \hat{\boldsymbol{w}} - 2\boldsymbol{y}^T \hat{X}^T \end{aligned}$$

- This yields the following best-effort solution for $\hat{\boldsymbol{w}}^*$ by utilizing the left pseudo-inverse of $\hat{X}$:

$$\hat{\boldsymbol{w}}^* = \left(\hat{X} \hat{X}^T\right)^{-1} \hat{X} \boldsymbol{y}$$

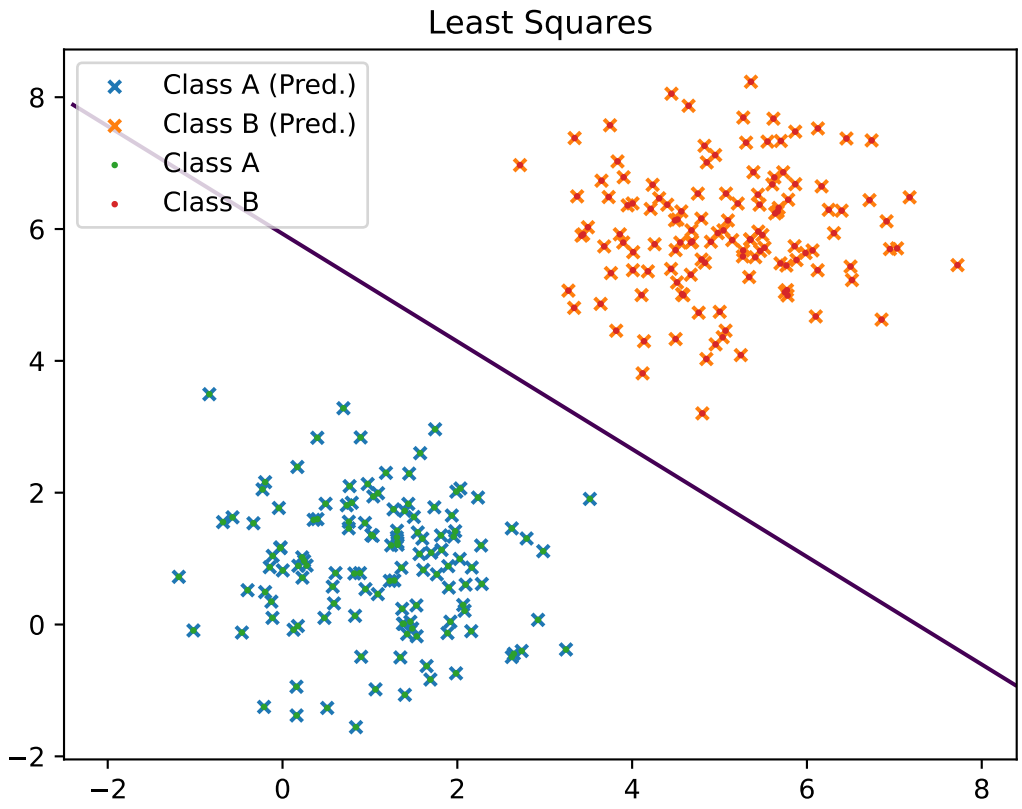

Figure 10.3.: Classification: Least Squares

Problems

- The least-squares solution for discriminative classification is very sensitive to outliers and breaks down even if there are only a few. This is due to the squared error which treats outliers more important than small errors.
- Calculating the matrix inverse is computationally expensive (but can be rewritten as solving a linear equation system).

Example Figure 10.3 shows the result from applying least squares regression to the sample data. As the data is linearly separable, there is no misclassification.

10.3.2. Fishers' Linear Discriminant

- Idea: Find a linear projection of the data and classify the projected values on this line.
- Same as for linear discriminant functions, check against a threshold:

$$\boldsymbol{w}^T \boldsymbol{x} + w_0 \geq 0$$

First Attempt: Maximize the Distance Take the two means

$$\boldsymbol{m}_1 = \frac{1}{|C_1|} \sum_{i \in C_1} \boldsymbol{x}_i \qquad \boldsymbol{m}_2 = \frac{1}{|C_2|} \sum_{i \in C_2} \boldsymbol{x}_i$$

with the projections $m_1 = \boldsymbol{w}^T \boldsymbol{m}_1$ and $m_2 = \boldsymbol{w}^T \boldsymbol{m}_2$ and maximize the distance $(m_1 - m_2)^2$. Here rises the problem that the distance grows unbounded with $\boldsymbol{w}$, so fix the norm of $\boldsymbol{w}$ to $\|\boldsymbol{w}\| = 1$. This yields the following optimization problem:

$$\begin{aligned} \arg\max_{\boldsymbol{w}} \; & J(\boldsymbol{w}) = (\boldsymbol{w}^T \boldsymbol{m}_1 - \boldsymbol{w}^T \boldsymbol{m}_2)^2 \\ \text{s.t.} \quad & \|\boldsymbol{w}\|^2 = 1 \end{aligned}$$

By performing Lagrangian optimization, the solution is

$$\boldsymbol{w} = \frac{\boldsymbol{m}_1 - \boldsymbol{m}_2}{\|\boldsymbol{m}_1 - \boldsymbol{m}_2\|}$$

This parameter causes a large class overlap, so do more: Maximize the mean distance while minimizing the variance of each class.

Final Attempt: Maximize the Distance, Minimize the Variance Let s_1^2 and s_2^2 be in *within-class variances*:

$$s_1^2 = \sum_{i \in C_1} (\boldsymbol{w}^T \boldsymbol{x}_i - m_1)^2 \qquad s_2^2 = \sum_{i \in C_2} (\boldsymbol{w}^T \boldsymbol{x} - m_2)^2$$

with $m_1 = \boldsymbol{w}^T \boldsymbol{m}_1$ and $m_2 = \boldsymbol{w}^T \boldsymbol{m}_2$. The *Fisher criterion* now formulates the optimization problem as:

$$\arg\max_{\boldsymbol{w}} J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

The nominator and the denominator can be rewritten to make the criterion easier to optimize:

$$\begin{aligned} (m_1 - m_2)^2 &= (\boldsymbol{w}^T \boldsymbol{m}_1 - \boldsymbol{w}^T \boldsymbol{m}_2)^2 \\ &= \left(\boldsymbol{w}^T (\boldsymbol{m}_1 - \boldsymbol{m}_2)\right)^2 \\ &= \boldsymbol{w}^T \underbrace{(\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T}_{\underbrace{S_B :=}_{\text{between-class covariance}}} \boldsymbol{w} \end{aligned}$$

$$s_1^2 + s_2^2 = \sum_{i \in C_1} (\boldsymbol{w}^T \boldsymbol{x}_i - m_1)^2 + \sum_{i \in C_2} (\boldsymbol{w}^T \boldsymbol{x} - m_2)^2$$

$$s_1^2 + s_2^2 = \sum_{i \in C_1} \left(\boldsymbol{w}^T (\boldsymbol{x}_i - \boldsymbol{m}_1)\right)^2 + \sum_{i \in C_2} \left(\boldsymbol{w}^T (\boldsymbol{x} - \boldsymbol{m}_2)\right)^2$$

$$s_1^2 + s_2^2 = \sum_{i \in C_1} \boldsymbol{w}^T (\boldsymbol{x}_i - \boldsymbol{m}_1)(\boldsymbol{x}_i - \boldsymbol{m}_1)^T \boldsymbol{w} + \sum_{i \in C_2} \boldsymbol{w}^T (\boldsymbol{x}_i - \boldsymbol{m}_2)(\boldsymbol{x}_i - \boldsymbol{m}_2)^T \boldsymbol{w}$$

$$s_1^2 + s_2^2 = \boldsymbol{w}^T \underbrace{\left[\sum_{i \in C_1} (\boldsymbol{x}_i - \boldsymbol{m}_1)(\boldsymbol{x}_i - \boldsymbol{m}_1)^T + \sum_{i \in C_2} (\boldsymbol{x}_i - \boldsymbol{m}_2)(\boldsymbol{x}_i - \boldsymbol{m}_2)^T \right]}_{\underbrace{S_W :=}_{\text{within-class covariance}}} \boldsymbol{w}$$

This way, the cost function can be rewritten:

$$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} = \frac{\boldsymbol{w}^T S_B \boldsymbol{w}}{\boldsymbol{w}^T S_W \boldsymbol{w}}$$

Differentiating the cost function w.r.t. $\boldsymbol{w}$ and setting it to zero yields

$$(\boldsymbol{w}^T S_B \boldsymbol{w}) S_W \boldsymbol{w} = (\boldsymbol{w}^T S_W \boldsymbol{w}) S_B \boldsymbol{w}$$

As the factory $\boldsymbol{w}^T S_B \boldsymbol{w}$ and $\boldsymbol{w}^T S_W \boldsymbol{w}$ are scalars, the vector $S_W \boldsymbol{w}$ and $S_B \boldsymbol{w}$ are colinear:

$$S_W \boldsymbol{w} \parallel S_B \boldsymbol{w}$$

Thus, with $S_B \boldsymbol{w} = (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^T \boldsymbol{w}$ it also holds that $S_B \boldsymbol{w} \parallel (\boldsymbol{m}_1 - \boldsymbol{m}_2)$ leading to *Fisher's linear discriminant*:

$$S_W \boldsymbol{w} \parallel S_B \boldsymbol{w} \parallel (\boldsymbol{m}_1 - \boldsymbol{m}_2) \quad \Longrightarrow \quad \boldsymbol{w} \propto S_W^{-1} (\boldsymbol{m}_1 - \boldsymbol{m}_2)$$

- Caution: Fisher's linear discriminant only yields a projection, the threshold w_0 is still missing and has to be found, e.g. by using a Bayes classifier with Gaussian class-conditionals.
- Fisher's linear discriminant is Bayes optimal iff the class-conditional distributions (likelihoods) are equal with diagonal covariance.
- It is essentially equivalent to linear discriminant analysis (it is equivalent to a certain case of the least squares classifier).
- Problem: It is still very sensitive to noise.

10.4. Perceptron Algorithm

The *perceptron algorithm* tries to find a separating hyperplane, given that the data is linearly separable. It depends on the following discriminator (called *perceptron discriminant function*):

$$y(\boldsymbol{x}) = \operatorname{sign}\left(\boldsymbol{w}^T \boldsymbol{x} + b\right)$$

with the sign function

$$\operatorname{sign} : \mathbb{R} \to \{-1, 0, +1\} : o \mapsto \begin{cases} -1 & \text{iff } o < 0 \\ 0 & \text{iff } o = 0 \\ +1 & \text{iff } o > 0 \end{cases}$$

Algorithm7 shows the perceptron algorithm for a dataset $\mathcal{D} = \left\{ (\boldsymbol{x}, y) \mid \boldsymbol{x} \in \mathbb{R}^d, y \in \{-1, +1\} \right\}$ for n iterations. The initialization vectors can be chosen differently.

Example Figure 10.4 shows the result of the perceptron algorithm after convergence (took 7 iterations).

Algorithm 7: Perceptron Algorithm

$\boldsymbol{w}^{(1)} \leftarrow \mathbf{1}$
$b^{(1)} \leftarrow 0$
for $k = 1, \cdots, n$ **do**
    $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)}$
    $b^{(k+1)} \leftarrow b^{(k)}$
    for $\forall(\boldsymbol{x}_i, y_i) \in \mathcal{D}$ **do**
        if $\text{sign}\left(\boldsymbol{w}^T \boldsymbol{x} + b\right) \neq y$ **then**
            if $y = -1$ **then**
                $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k+1)} - \boldsymbol{x}$
                $b^{(k+1)} \leftarrow b^{(k+1)} - 1$
            if $y = +1$ **then**
                $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k+1)} + \boldsymbol{x}$
                $b^{(k+1)} \leftarrow b^{(k+1)} + 1$

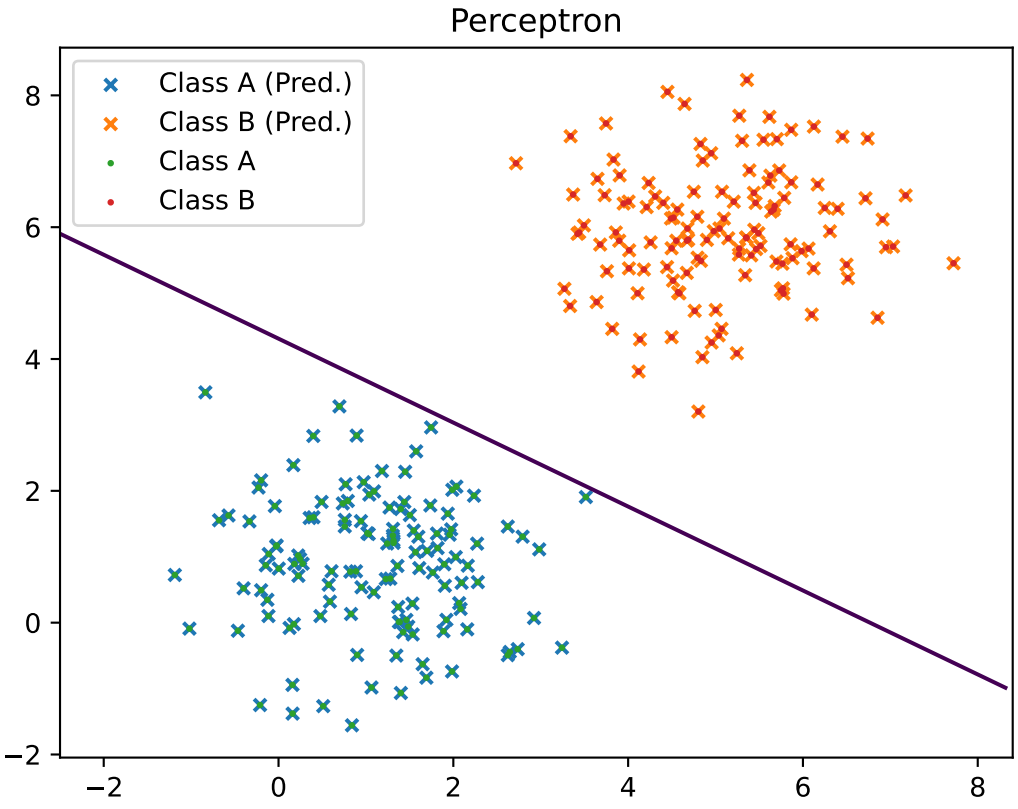

Figure 10.4.: Classification: Perceptron (7 Iterations)

10.4.1. Intuition

10.4.2. Linear Separability

- The perceptron algorithm is not able to handle non linearly separable data, e.g. the XOR function.
- This halted research for decades.
- Simple solution: Transform the input space nonlinearly to make it linearly separable.
- Insight: Create features and learn from them, not from the raw data! This automatically done by neural networks.

10.5. Probabilistic Discriminative Models

The class posterior can be expressed using Bayes rule and the sigmoid function (for two classes):

$$p(C_1 \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid C_1)\, p(C_1)}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x} \mid C_1)\, p(C_1)}{p(\boldsymbol{x} \mid C_1)\, p(C_1) + p(\boldsymbol{x} \mid C_2)\, p(C_2)} = \frac{1}{1 + \frac{p(\boldsymbol{x} \mid C_1)\, p(C_1)}{p(\boldsymbol{x} \mid C_2)\, p(C_2)}} = \sigma(a)$$

for $a = \ln \frac{p(\boldsymbol{x} \mid C_1)\, p(C_1)}{p(\boldsymbol{x} \mid C_2)\, p(C_2)}$.

10.5.1. Logistic Regression

- In *logistic regression*, it is assumed that a is given by a discriminant function $a = \boldsymbol{w}^T \boldsymbol{x} + w_0$, so the challenge is to find $\boldsymbol{w}$ and w_0 to model the class posterior the best.
- This is an appropriate assumption if:
  - The class conditionals are Gaussians with equal covariances.
  - But also for a other distributions.
  - There must be some independence of the form of the class-conditionals.
- Logistic regression works by maximizing the likelihood $p(Y \mid X, \boldsymbol{w}, w_0)$ (where y_i is 0 iff $\boldsymbol{x}_i \in C_1$ and is 1 iff $\boldsymbol{x}_i \in C_2$), assuming the data is drawn i.i.d.:

$$\begin{aligned} p(Y \mid X, \boldsymbol{w}, w_0) &= \prod_{i=1}^{N} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{w}, w_0) \\ &= \prod_{i=1}^{N} p(C_1 \mid \boldsymbol{x}_i, \boldsymbol{w}, w_0)^{1-y_i}\, p(C_2 \mid \boldsymbol{x}_i, \boldsymbol{w}, w_0)^{y_i} \\ &= \prod_{i=1}^{N} \sigma(\boldsymbol{w}^T \boldsymbol{x}_i + w_0)^{1-y_i}\, (1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i + w_0))^{y_i} \end{aligned}$$

- The key idea is to now apply the logarithm and do gradient descent, for a derivation see Bishop 4.3.
- More robust classifiers can be retrieved by incorporating priors and taking a Bayesian approach.

10.6. Wrap-Up

- Bayes optimal classifier
- Discriminant functions
- Formalization (intuitively and mathematically) of classification as linearly separable
- Computation of the least squares solution for classification and its failure
- Fisher's linear discriminant and difference to least squares
- Perceptron and its failure for XOR and how to overcome it
- Difference between generative and discriminative models
- Logistic regression

11. Linear Dimensionality Reduction

In this chapter, linear models are implied for simplicity, if not stated otherwise.

- *Dimensionality reduction* is part of the unsupervised learning methods which reduces the dimension of the data.
- One possible application is the visualization of the data.
- Motivation from least squares regression: LSR requires the inversion of a $d \times d$ matrix, where d is the dimension. If it is possible to find a new $d^{\text{new}} \ll d$ which represents the data well enough, the computation cost can be reduced while not loosing precision.
- The key problem is to find representations (especially transformations) of the data into a lower-dimensional subspace, that capture the "essence" of the data.
- More formally: For every original data point $\boldsymbol{x}^n \in \mathbb{R}^M$, find a low-dimensional representation $\boldsymbol{a}^n \in \mathbb{R}^D$ with $D \ll M$. This is a mapping $f : \mathbb{R}^M \to \mathbb{R}^D : \boldsymbol{x}^n \mapsto \boldsymbol{a}^n$.
- For simplicity, restrict this mapping function to be linear with a matrix $B \in \mathbb{R}^{D \times M}$:

$$\boldsymbol{a}^n = B\boldsymbol{x}^n$$

11.1. Introduction

Linear Combinations

- A vector can always be written as a linear combination

$$\boldsymbol{x} = \sum_{i=1}^{M} a_i \boldsymbol{u}_i$$

  where $\boldsymbol{u}_i^T \boldsymbol{u}_j = \delta_{ij}$. Thus, the $\boldsymbol{u}_i$ build an orthonormal basis of the feature space.
- By rewriting the linear combination, a_i can be expressed as a projection $a_i = \boldsymbol{u}_i^T \boldsymbol{x}$:

$$\boldsymbol{x} = \sum_{i=1}^{M} a_i \boldsymbol{u}_i = a_j \boldsymbol{u}_j + \sum_{\substack{i=1 \\ i \neq j}}^{M} a_i \boldsymbol{u}_i$$

$$\iff \quad a_j \boldsymbol{u}_j = \boldsymbol{x} - \sum_{\substack{i=1 \\ i \neq j}}^{M} a_i \boldsymbol{u}_i$$

$$\iff \quad a_j = \boldsymbol{u}_j^T \boldsymbol{x} - \sum_{\substack{i=1 \\ i \neq j}}^{M} a_i \underbrace{\boldsymbol{u}_j^T \boldsymbol{u}_i}_{= \delta_{ji} \underset{j \neq i}{=} 0}$$

$$\iff \quad a_j = \boldsymbol{u}_j^T \boldsymbol{x}$$

- The linear combination can be decomposed as

$$\boldsymbol{x}^n = \sum_{i=1}^{D} a_i \boldsymbol{u}_i + \underbrace{\sum_{j=D+1}^{M} b_j \boldsymbol{u}_j}_{\text{Error}} \approx \tilde{\boldsymbol{x}}$$

  with the reconstructed data $\tilde{\boldsymbol{x}}$, yielding the following optimization problem (minimizing the mean squared error over the training data):

$$\boldsymbol{u}_1, \cdots, \boldsymbol{u}_D = \arg \min_{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_D} E(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_D) = \arg \min_{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_D} \sum_{n=1}^{N} \|\boldsymbol{x}^n - \tilde{\boldsymbol{x}}^n\|^2$$

Minimizing the Error

- The error can be rewritten (assuming a single basis vector to find the first principal direction):

$$\begin{aligned} E(\boldsymbol{u}) &= \sum_{n=1}^{N} \|\boldsymbol{x}^n - \tilde{\boldsymbol{x}}^n\|^2 \\ &= \sum_{n=1}^{N} \|\boldsymbol{x}^n - (\boldsymbol{u}^T \boldsymbol{x}^n)\boldsymbol{u}\|^2 \\ &= \sum_{n=1}^{N} \|\boldsymbol{x}^n\|^2 - 2(\boldsymbol{u}^T \boldsymbol{x}^n)^2 + (\boldsymbol{u}^T \boldsymbol{x}^n)^2 \, \boldsymbol{u}^T \boldsymbol{u} \\ &= \sum_{n=1}^{N} \|\boldsymbol{x}^n\|^2 - (\boldsymbol{u}^T \boldsymbol{x}^n)^2 \\ &= \sum_{n=1}^{N} \|\boldsymbol{x}^n\|^2 - a_n^2 \end{aligned}$$

- So minimizing the error is equivalent to maximizing the variance of the projection (assuming a zero mean on the data, this can be achieved by subtracting the mean from every data point $\boldsymbol{x}^n - \bar{\boldsymbol{x}}$).
- Thus, the goal changes to finding the axis with the largest variance.
- The resulting axis are orthogonal and decorrelate the data (in the coordinate frame of the new axis, the data is uncorrelated). This only works for Gaussians!

11.2. Principal Component Analysis

- The goal of *principal component analysis* is to find the so-called *principal directions* and the variance of the data along each principal direction.
- In the following, λ_i is the *marginal variance* along the principal direction $\boldsymbol{u}_i$.
- The first principal direction $\boldsymbol{u}_1$ is the direction along the variance of the data is maximal (C is the covariance matrix):

$$\boldsymbol{u}_1 = \arg\max_{\boldsymbol{u}} \boldsymbol{u}^T C \boldsymbol{u}$$

- The second principal direction maximizes the variance of the data in the orthogonal complement of the first principal direction.

11.2.1. Derivation

Let $X = \begin{bmatrix} \boldsymbol{x}^2 & \cdots & \boldsymbol{x}^n \end{bmatrix} \in \mathbb{R}^{M \times N}$ be a matrix of N vectors in a M-dimensional input space. Let $\boldsymbol{u} \in \mathbb{R}^M$ be a unit vector in the input space. The projection of the vector $\boldsymbol{x}^j$ onto the vector $\boldsymbol{u}$ can be computed as:

$$a_j = \boldsymbol{u}^T \boldsymbol{x}^j = \sum_{i=1}^{M} X_{ij} u_i$$

The goal is to find a direction $\boldsymbol{u}$ that maximizes the variance of the projections of all input vectors.

Variance of the Projection The variance of the projection can be computed as (with $\mu_i = \frac{1}{N}\sum_{j=1}^{N} X_{ij}$):

$$\bar{a} = \frac{1}{N}\sum_{j=1}^{N} a_j = \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{M} X_{ij} u_i = \sum_{i=1}^{M} u_i \mu_i$$

$$\sigma^2 = \frac{1}{N}\sum_{j=1}^{N}(a_j - \bar{a})^2 = \frac{1}{N}\sum_{j=1}^{N}\left(\sum_{i=1}^{M} u_i X_{ij} - \sum_{i=1}^{M} u_i \mu_i\right)^2 = \boldsymbol{u}^T C \boldsymbol{u}$$

Maximizing the Variance The variance has to be maximized with the constraint $\|\boldsymbol{u}\| = 1$:

$$\max_{\boldsymbol{u}} J(\boldsymbol{u}) = \boldsymbol{u}^T C \boldsymbol{u}$$
$$\text{s.t.} \qquad \|\boldsymbol{u}\| = 1$$

The Lagrangian formulation

$$L(\boldsymbol{u}, \lambda) = \boldsymbol{u}^T C \boldsymbol{u} - \lambda \sum_{k=1}^{M}\left(u_k^2 - 1\right) = \sum_{i=1}^{M}\sum_{j=1}^{M} u_i C_{ij} u_j - \lambda \sum_{k=1}^{M}\left(u_k^2 - 1\right)$$

yields the solution

$$C\boldsymbol{u} = \lambda \boldsymbol{u}$$

This is the Eigenvalue-Eigenvector equation! So solving for the eigenvalues and eigenvectors gives the following results:

- The larges eigenvalue gives the maximal variance and
- the corresponding eigenvector gives the direction with the maximal variance.

11.2.2. Conclusion

As the covariance matrix C is real, symmetric and positive-definite, the eigenvalues and -vectors can be grouped in an Eigendecomposition:

$$C = U \Lambda U^T = \begin{bmatrix} \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_M \end{bmatrix} \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_M \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1^T \\ \vdots \\ \boldsymbol{u}_M^T \end{bmatrix}$$

- If $\lambda_k \approx 0$ for $k > D$ for some $D \ll M$, the subset of the first D eigenvectors (with the greatest value) can be used as a basis to approximate the data vectors.
- The eigenvalues λ_k then represent the *explained variance* in that direction.
- This representation has the minimal mean squared error of all linear representations of dimension D.
- The following steps have to be done to represent the data in a lower space (and reconstruct it):
  1. Center the data around the mean (compute it and subtract it from all data points). May standardize the variance (compute the variance and divide the normalized data by it).
  2. Compute the covariance matrix, decompose it into eigenvalues and -vectors and find the first D eigenvalues with the highest corresponding eigenvalues. As the covariance matrix must be positive semi-definite, all eigenvalues are $\lambda_i \geq 0$.
  3. Transform the data into a lower dimensional space (with $B = \begin{bmatrix} \boldsymbol{u}_1 & \cdots & \boldsymbol{u}_D \end{bmatrix}$):

     $$\boldsymbol{a}^n = B^T \left(\boldsymbol{x}^n - \bar{\boldsymbol{x}} \right)$$

     If the data was standardized, the last term has to be divided by the variance.
  4. To reconstruct the data, reverse-apply the formula:

     $$\tilde{\boldsymbol{x}}^n = \bar{\boldsymbol{x}} + B \boldsymbol{a}^n$$

     If the data was standardized, multiply it by the variance.

Example Figure 11.1 shows principal component analysis applied to the iris dataset with the data of flowers. The explained variance vs. the number of components is shown in figure 11.2

11.3. Choosing the target Dimension

- A larger D leads to a better approximation (with $D = M$, 100% accuracy as the dimension is not changed).
- There exist two good possibilities to choose the target dimension:
  1. Choose D based on the application performance (choose the smallest D that makes the application work well enough).
  2. Choose D so that the basis captures some fraction of the variance, so choose D so that, for a given η:

     $$\sum_{i=1}^{D} \lambda_i \geq \eta \sum_{i=1}^{M} \lambda_i$$

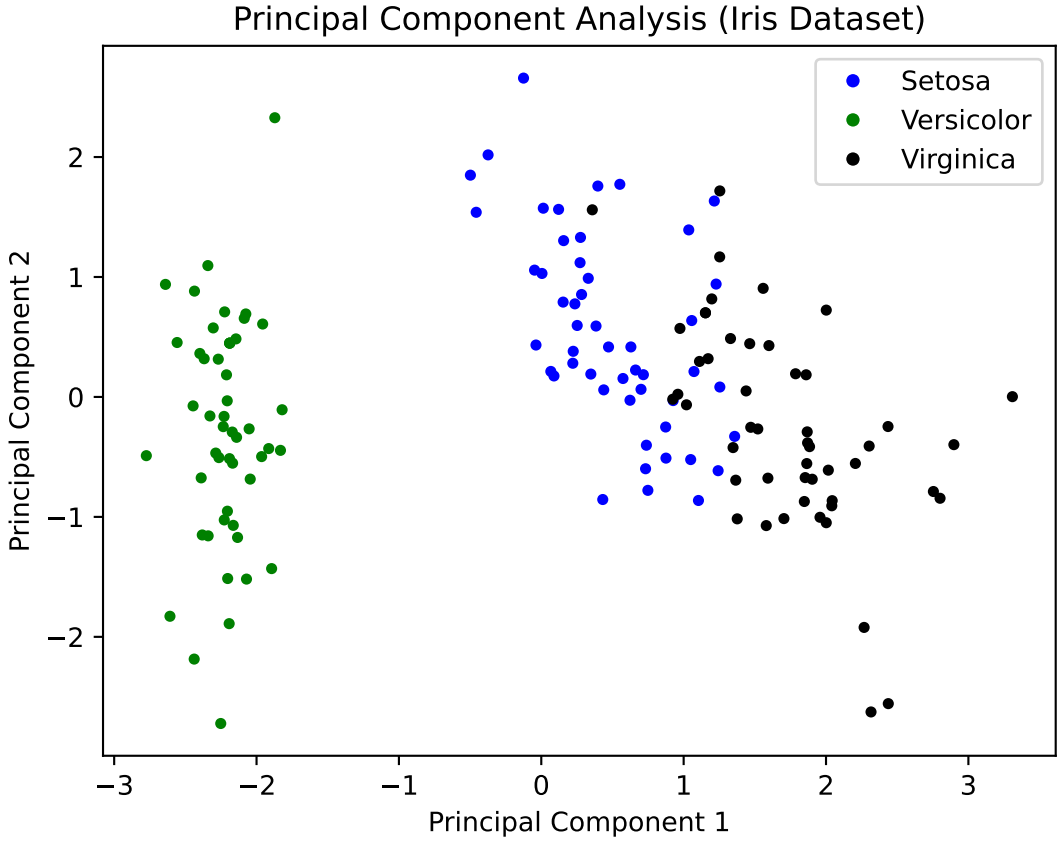

Figure 11.1.: Principal Component Analysis: Iris Dataset

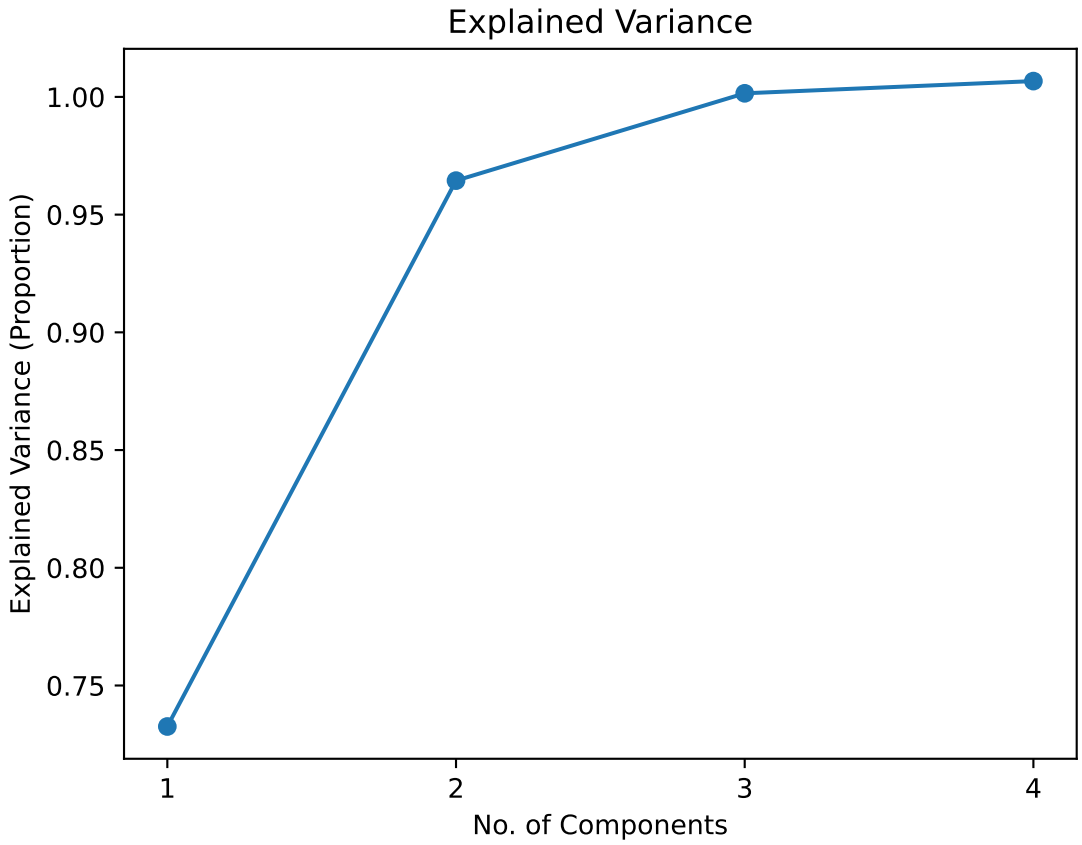

Figure 11.2.: Principal Component Analysis: Iris Dataset (Explained Variance)

11.4. Applications

11.5. Wrap-Up

- Dimensionality reduction and why its needed
- Intuition behind PCA
- Maximization of the variance of the projection
- Relation of PCA to eigenvector and -values

12. Statistical Learning Theory

- In classical statistical learning, the parameters $\boldsymbol{w}$ are estimated for a *fixed model* (the *learning machine*).
- Learning occurs only by optimizing the model parameters. It is assumed that the correct model is known in advance (except in Bayesian learning).
- Selecting the "correct" features is hard and an overly complex model leads to overfitting.
- The real point of interest is the generalization ability and the corresponding risk.
- In *statistical learning theory*, these assumptions are not made and the goal is to find an optimal model from a specified set of models. Optimally means the ability to generalize, i.e. to have the lowest error probability on all data, not just the test data. It is concerned with the question on how to control the generalization abilities of a learning machine.
- It aims at a formal description of the generalization ability and the goal is to develop a rigorous theory as opposed to commonly used heuristics.
- This is a good goal, but the theory itself does not say much about real problems...

12.1. Supervised Learning

- The environment is stationary, the data points have an unknown but fixed probability density

$$\boldsymbol{x}_i \sim p_X$$

- The supervisor returns the intended classification label for every data point $\boldsymbol{x}$, possibly with some noise ϵ

$$y = g(\boldsymbol{x}, \epsilon)$$

- The learning machine is represented through a class of functions with parameters $\boldsymbol{w}$ that return an output y for every input $\boldsymbol{x}$

$$y = f(\boldsymbol{x}, \boldsymbol{w})$$

- From the view of the learning machine, choose a particular function $y = f(\boldsymbol{x}, \boldsymbol{w})$ given a set of training examples $\{ \boldsymbol{x}_i, y_i \}_{i=1}^{N}$. Goal: Approximate the desired output y optimally.
- This optimality can be expressed by a loss function, e.g. quadratic loss

$$L\big(L, f(\boldsymbol{x}, \boldsymbol{w})\big) = \big(y - f(\boldsymbol{x}, \boldsymbol{w})\big)^2$$

12.2. Assessment of Optimality: Risk

The risk for a particular data point can be expressed with a loss function

$$L\big(y, f(\boldsymbol{x}, \boldsymbol{w})\big)$$

This yields the *empirical risk* as the average over all available samples

$$R_{\text{emp}}(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} L\big(y_i, f(\boldsymbol{x}, \boldsymbol{w})\big)$$

where N is the number of samples.

In reality, the *true risk*

$$R(\boldsymbol{w}) = \int L\big(y, f(\boldsymbol{x})\big)\, p(\boldsymbol{x}, y)\, \mathrm{d}\boldsymbol{x}\, \mathrm{d}y = \mathbb{E}_{\boldsymbol{x}, y \sim p(\boldsymbol{x},)}\Big(L\big(y, f(\boldsymbol{x}, \boldsymbol{w})\big)\Big)$$

is far more interesting, where $p(\boldsymbol{x}, y)$ is the joint probability density of $\boldsymbol{x}$ and y. The risk is the expected error over all data sets and is the expectation of the generalization error.

Problem: The probability density $p(\boldsymbol{x}, y)$ is fixed, but unknown. So the true risk cannot be computed directly.

12.2.1. Empirical vs. True Risk

- **True Risk**
  - Advantage: The actual measure for the generalization ability.
  - Disadvantage: Depends on $p(\boldsymbol{x}, y)$ which is unknown $\implies$ the true risk cannot be computed directly.
- **Empirical Risk**
  - Disadvantage: No "real" measure for the generalization ability.
  - Advantage: Does not depend on $p(\boldsymbol{x}, y)$ and can be computed directly.
  - Learning algorithms usually minimize the empirical risk.
- The interest point are the dependencies between the two risks.
- The empirical risk is an approximation for the true risk that works well if the distribution is very concentrated. It gets very good if there are infinitely many samples.

12.2.2. Convergence Properties

At first assume that the empirical risk converges to the true risk with more samples ($\inf$ is the infimum):

$$\lim_{N \to \infty} \inf_{\boldsymbol{w}} R_{\text{emp}}(\boldsymbol{w}) = \inf_{\boldsymbol{w}} R(\boldsymbol{w})$$

Also assume that the convergence has to be uniform:

$$\lim_{N \to \infty} P\left(\sup_{\boldsymbol{w}} \big|\, R(\boldsymbol{w}) - R_{\text{emp}}(\boldsymbol{w}) \,\big| > \epsilon\right) = 0$$

Intuition: “The learning machine gets better the more data it has.”

If the convergence is uniform $P\left(\sup_{\boldsymbol{w}} \left| R(\boldsymbol{w}) - R_{\text{emp}}(\boldsymbol{w}) \right| > \epsilon\right) < p^*$ for some $p^* > 0$, then with probability $1 - p^*$ it holds that

$$\begin{aligned} \left| R(\boldsymbol{w}_{\text{emp}}) - R_{\text{emp}}(\boldsymbol{w}_{\text{emp}}) \right| &< \epsilon \\ \left| R(\boldsymbol{w}_0) - R_{\text{emp}}(\boldsymbol{w}_0) \right| &< \epsilon \end{aligned}$$

Hence it holds that

$$P\Big(\left| R(\boldsymbol{w}_0) - R_{\text{emp}}(\boldsymbol{w}_{\text{emp}}) \right| > 2\epsilon\Big) < p^*$$

Under the necessary and sufficient condition that the convergence is uniform, minimizing the empirical risk guarantees the minimization of the true risk in the limit of $N \to \infty$.

- **Advantages**
  - Existence of a formal criterion to what can be expected in terms of generalization.
  - The necessary and sufficient condition is the uniform convergence.
- **Disadvantages**
  - In reality, the training data is very limited.
  - “Taking the limit” with $N \to \infty$ is impossible.

12.3. Risk Bound

Idea: Determine an upper *risk bound* on the true risk based on the empirical risk

$$R(\boldsymbol{w}) \leq R_{\text{emp}}(\boldsymbol{w}) + \epsilon(N, p^*, h)$$

where N is the number of training samples, p^* is the probability that the bound is met and h is the *learning power* of the learning machine, formally called *VC-dimension*.

12.3.1. VC-Dimension

- VC stands for Vapnik–Chervonenkis, the developers of the VC-theory.
- Informal definition of the VC-dimension:
  - The VC-dimension of a family of functions is the maximum number of samples that can be correctly classified by a function from that family (independent of the label configuration).
  - The VC-dimension is a measure of the capacity (“learning power”) of a classifier.
  - The VC-dimension is the number of data points that can be shattered by a function.
- Example: The VC-dimension of linear classifiers (hyperplanes) in $\mathbb{R}^n$ is $(n + 1)$.
- Often (but not always!) the VC-dimension is directly related to the number of parameters.

12.3.2. Example

For the loss function, the true risk and the empirical risk

$$L\big(y, f(\boldsymbol{x}, \boldsymbol{w})\big) = \frac{1}{2}\big|y - f(\boldsymbol{x}, \boldsymbol{w})\big|$$
$$R(\boldsymbol{w}) = \int \frac{1}{2}\big|y - f(\boldsymbol{x}, \boldsymbol{w})\big|\, p(\boldsymbol{x}, y)\, \mathrm{d}\boldsymbol{x}\, \mathrm{d}y$$
$$R_{\mathrm{emp}}(\boldsymbol{w}) = \frac{1}{2N}\sum_{i=1}^{N}\big|y_i - f(\boldsymbol{x}_i, \boldsymbol{w})\big|$$

with probability p^* it holds that

$$R(\boldsymbol{w}) \leq R_{\mathrm{emp}}(\boldsymbol{w}) + \sqrt{\frac{h\big(\ln(2N/h) + 1\big) - \ln\big((1-p^*)/4\big)}{N}}$$

- The upper bound is independent of $p(\boldsymbol{x}, y)$!
- As the true risk is not computable, but the VC-dimension is known, a bound of the type

  $$R(\boldsymbol{w}) \leq R_{\mathrm{emp}}(\boldsymbol{w}) + \epsilon(N, p^*, h)$$

  with a confidence interval ϵ can always be computed.
- However, in practice, this bound is very loose and the true risk may be much lower.

12.4. Structural Risk Minimization

- Given a family of n models $f_i(\boldsymbol{x}_i, \boldsymbol{w}_i)$ with $h_1 \leq h_2 \leq \cdots \leq h_n$.
- Minimize the empirical risk for every model and choose the model that minimizes the risk bound (the right side of the risk bound equation).
- In general, this is not the same model that minimizes the empirical risk.
- This formally lowers the upper bound on the true risk.
- The result is only sensible if the upper bound in the true risk is a tight bound (which is typically not).

12.5. Wrap-Up

- Statistical learning theory
- Empirical vs. true risk
- Incompleteness of the empirical risk
- VC-Dimension
- Relation between the VC-dimension and the model complexity

13. Neural Networks

- Selecting the "right" features for a problem is really hard, but the representation of the data matters a lot.
- Neural networks learn complex data representations by combining simpler ones (features of features).
- The big shifts that lead to neural networks are:
  - Too little data $\rightarrow$ too much data.
  - Linear and convex $\rightarrow$ nonlinear and nonconvex.
  - Intuitive features $\rightarrow$ harder features, key focus on learning.
  - "Right number of parameters" $\rightarrow$ "always too many".
  - Optimization becomes easier by being deep.
- Neural networks have a long history...
  - **Pre-computational (1888-):** Neuron in biology fully isolated by Ramon y Cajal
  - **Fields Starts (1943-):** McCullogh&Pitts Neuron and Networks
  - **1st Hype (1957-):** Rosenblatt's Perceptron
  - **1st Winter (1969-):** Papert/Minsky book perceptron with XOR example (not linear separable)
  - **2nd Hype (1986-1994):** Rummelthart/Hinton/Williams rediscover backpropagation
  - **2nd Winter (1994-):** Optimization is really hard, Kernels are better!
  - **2007:** Rebooted by NIPS workshops
  - **3rd Hype (2013-now):** Amazing results in computer vision (ImageNet), Natural Language Processing, (Deep) Reinforcement Learning, ...
- Neural networks can be adapted to regression or classification!
  - A linear output node gives a linear regression function.
  - Using a sigmoid output node gives something similar to logistic regression.
  - In either case, by taking the sign of the output, classification can be obtained.
  - Typically not maximum likelihood is used for learning but a different learning criterion.
- The actual power of neural networks comes from the extensions for multi-class classification and multi-layer perceptrons.

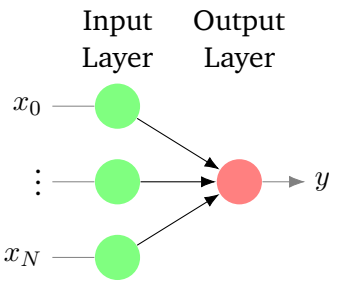

Figure 13.1.: Neural Network: Single-Layer

13.1. Abstraction of a Neuron

- A single neuron can be represented as

$$y = f\Big(\sum_{i=1}^{n} \boldsymbol{W}_i x_i + b\Big) = f(\boldsymbol{W}^T \boldsymbol{x} + b) = f(\hat{\boldsymbol{W}}^T \hat{\boldsymbol{x}})$$

  with the input $\hat{\boldsymbol{x}} = \begin{bmatrix}\boldsymbol{x}^T & 1\end{bmatrix}^T$, parameters and weights $\hat{\boldsymbol{W}} = \begin{bmatrix}\boldsymbol{W}^T & b\end{bmatrix}^T$ and an activation function f.

- Neurons are pooled together in *layers* of m input and n outputs, where each layer has
  - Weight matrix $W \in \mathbb{R}^{n \times m}$
  - Bias vector $\boldsymbol{b} \in \mathbb{R}^{n \times 1}$
  - Input vector $\boldsymbol{x} \in \mathbb{R}^{m \times 1}$
  - Pre-activation vector $\boldsymbol{z} = W\boldsymbol{x} + \boldsymbol{b}$
  - Output vector $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z})$ with $f : \mathbb{R}^{n \times 1} \to \mathbb{R}^{n \times 1}$

13.2. Single-Layer Neural Networks

13.2.1. Logistic Regression

In logistic regression, the class posterior is modeled as

$$p(C_1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{W}^T \boldsymbol{x} + b)$$

and a solution for $\boldsymbol{W}$ and b is found by maximizing the likelihood $p(Y \mid X, \boldsymbol{W}, b)$.
This is equivalent to a neural network as shown in figure 13.1 with a sigmoid activation function.

13.2.2. Multi-Class Network

- A single layer network can also have multiple output neurons, yielding multidimensional linear regression.
- Nonlinear extension is straightforward by applying the sigmoid for a logistic output.

13.2.3. Least-Squares Loss Function

- In supervised learning, N data points $X = \begin{bmatrix}\boldsymbol{x}^1 & \cdots & \boldsymbol{x}1N\end{bmatrix}$ are given.
- For each data point there are c possible target values $k \in 1, \cdots, c$: $T_k = \begin{bmatrix}t_k^1 & \cdots & t_k^N\end{bmatrix}$.
- The model can compute $y_k(\boldsymbol{x}^n, W)$, yielding the least-squares error/loss function:

$$E(W)\frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}\left(y_k(\boldsymbol{x}^n, W) - t_k^n\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}\left(f\left(\sum_{i=1}^{d} W_{ki}\phi_i(\boldsymbol{x}^n)\right) - t_k^n\right)^2$$

with arbitrary feature transformations $\phi_i(\cdot)$.

13.2.4. Learning with Gradient Descent

Assuming the output with a linear activation $y_k(\boldsymbol{x}^n) = \sum_{i=1}^{d} W_{ki}\phi_i(\boldsymbol{x}^n)$, the error function ans its derivative w.r.t. the weights compute as:

$$E(W) = \sum_{n=1}^{N}\frac{1}{2}\sum_{k=1}^{c}\left(\sum_{i=1}^{d} W_{ki}\phi_i(\boldsymbol{x}^n) - t_k^n\right)^2 = \sum_{n=1}^{N} E^n(W)$$

$$\frac{\partial E^n(W)}{\partial W_{lj}} = \left(\sum_{i=1}^{d} W_{li}\phi_i(\boldsymbol{x}^n) - t_l^n\right)\phi_j(\boldsymbol{x}^n) = \left(y_l(\boldsymbol{x}^n) - t_l^n\right)\phi_j(\boldsymbol{x}^n)$$

Then the weights can be updated using gradient descent:

$$W_{lj} \leftarrow W_{lj} - \eta\frac{\partial E(W)}{\partial W_{lj}}\bigg|_W$$

$$\frac{\partial E(W)}{\partial W_{lj}} = \sum_{n=1}^{N}\frac{\partial E^n(W)}{\partial W_{lj}}$$

This is computationally expensive if all data points are used for the gradient estimation.

In a network with a nonlinear activation $y_k(\boldsymbol{x}^n) = f(a_k) = f\left(\sum_{i=1}^{d} W_{ki}\phi_i(\boldsymbol{x}^n)\right)$ the error derivative gets:

$$\frac{\partial E^n(W)}{\partial W_{li}} = f'(a_l)\left(y_l(\boldsymbol{x}^n) - t_l^n\right)\phi_j(\boldsymbol{x}^n)$$

In a logistic neural network:

$$f(a) = \sigma(a) \qquad \sigma'(a) = \sigma(a)\left(1 - \sigma(a)\right)$$

13.3. Multi-Layer Neural Networks

Multi-layer neural networks have input and output layers like the single-layer networks, but contain so-called *hidden layers* which lie in between the input and output layers. An example network is shown in figure 13.2, which also has multiple output nodes. Neural networks that have more than one hidden layer are called *deep neural networks*.

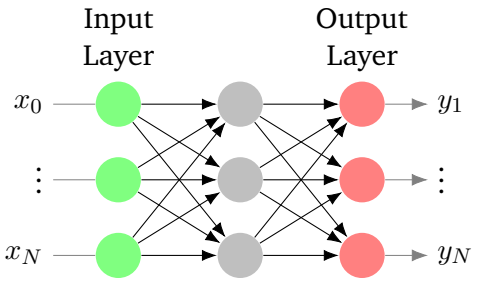

Figure 13.2.: Neural Network: Multi-Layer

In a network with just one hidden layer the output computes as

$$y_k(\boldsymbol{x}) = f^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} \underbrace{f^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)}_{z_i}\right)$$

The function $f^{(k)}$ is the activation function for the k-th layer (the output layer counts as a layer). The hidden layer may have an arbitrary number of nodes h.

A multi-layer network (also called *multi-layer perceptron*) is calculated as

$$y_k(\boldsymbol{x}) = f^{(N)}\left(\sum_{i_{N-1}=0}^{h_{N-1}} W_{ki_{N-1}}^{(N)} f^{(N-1)}\left(\sum_{i_{N-2}=0}^{h_{N-2}} W_{i_{N-1}i_{N-2}}^{(N-1)} f^{(N-2)}\left(\cdots f^{(2)}\left(\sum_{i_1=0}^{h} W_{i_2 i_1}^{(2)} f^{(1)}\left(\sum_{i_0=0}^{d} W_{i_1 i_0}^{(1)} x_{i_0}\right)\right)\right)\right)\right)$$

A multi-layer network can be seen as a machine that builds features on top of features.

13.3.1. One hidden Layer?

The universal function approximation theorem says that one hidden layer can represent every function arbitrarily accurate (Cybenko/Hornik). But this needs and exponential number of units (neurons)! Instead, multiple layers allow a similar effect with much less units.

13.3.2. Model Type and Model Class

Model Type The model type is the choice of the nonlinear parametric model. It is determined by:

- Choice of topology: How are the neural layers connected and how many neurons per layer?
- Choice of neural elements: How is the neuron modeled?

Widely talking, everything in ML is a neural network, maybe with just one layer and one activation function.

- Feedforward neural networks are acyclic directed graphs.
  - Multi-layer perceptrons are fully connected, while
  - Convolutional networks are smartly pruned with weight-sharing.
- Recurrent neural networks are cyclic directed graphs with internal states.

Model Class The model class is the number of hidden neurons and the number of layers.

13.4. Output Neurons, Activation and Loss Functions

13.4.1. Output Neurons

The type of the problem determines the type of the output neurons, all having probabilistic interpretations:

- Linear for regression:

$$\boldsymbol{f}(\boldsymbol{z}) = \boldsymbol{z} \qquad p(\boldsymbol{y} \,|\, \boldsymbol{x}) = \mathcal{N}\left(\boldsymbol{y} \,|\, \boldsymbol{z}, \sigma^2 I\right)$$

- Sigmoid for (two-class) classification:

$$f(z) = \sigma(z) \equiv \frac{1}{1 + e^{-z}} \qquad p(y \,|\, z) = \sigma(z)^y \left(1 - \sigma(z)\right)^{1-y}$$

- Categorical Distribution/Softmax for multi-class classification:

$$f_i(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \equiv p(y = i \,|\, \boldsymbol{z})$$

13.4.2. Loss Functions

Just like the type of the output neuron is linked to the problem, is the loss function linked to the problem:

- Regression
  - Linear output $\implies$ Squared loss
- Classification
  - Linear output $\implies$ Hinge loss
  - Sigmoid $\implies$ Nonlinear log-likelihood
- Multi-Class Classification
  - Softmax $\implies$ Nonlinear log-likelihood

All these are derivable from maximum likelihood.

13.4.3. Activation Functions

- Hidden neurons may be chosen freely, because it is unknown what they actually do (but the derivative controls how much of a rule a neuron plays in learning).
- All the technical choices remain voodoo and depend on intuition.
- There are best practices and heuristics which one to choose.

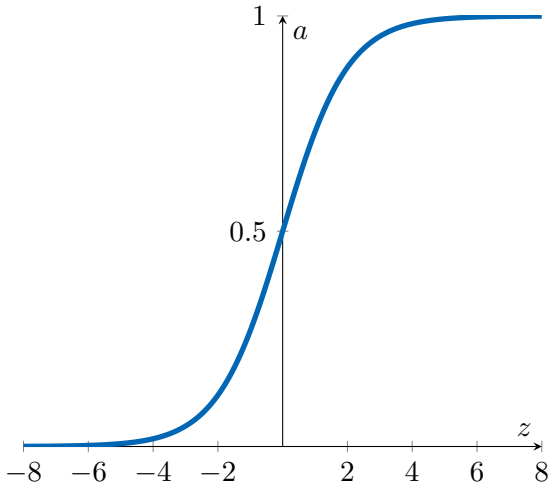

Figure 13.3.: Sigmoid $\sigma(z)$

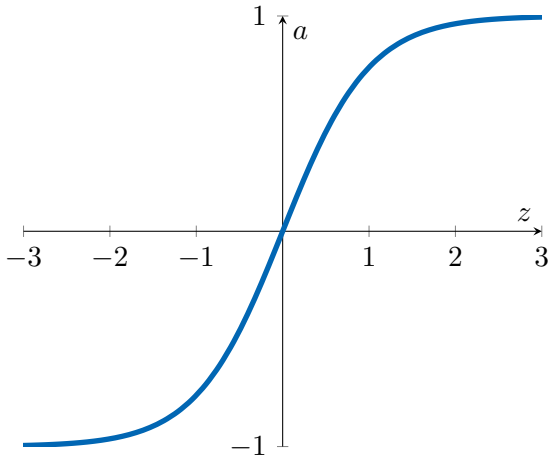

Figure 13.4.: Sigmoid $\tanh(z)$

Sigmoid

Figure 13.3 shows the sigmoid.

$$f(z) = \sigma(z) \qquad f'(z) = \sigma(z)\left(1 - \sigma(z)\right)$$

Problem: The derivative is zero almost everywhere, causing a zero gradient during backpropagation and may stop learning.

Hyperbolic Tangent

Figure 13.4 shows the hyperbolic tangent.

$$f(z) = \tanh(z) \qquad f'(z) = 1 - \tanh^2(z)$$

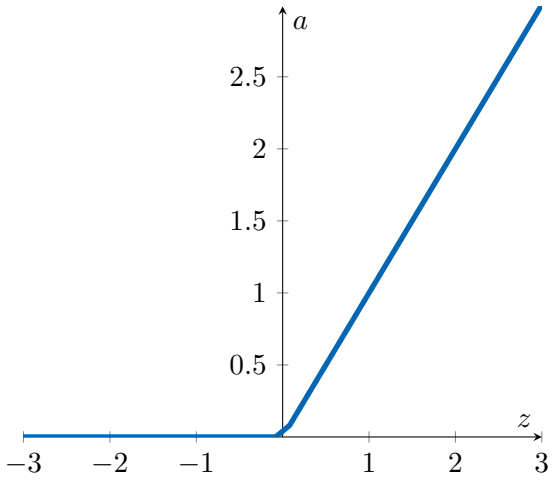

Figure 13.5.: Rectified Linear Unit ReLU $\max(0, z)$

Rectified Linear Unit (ReLU)

Figure 13.5 shows the rectified linear unit.

$$f(z) = \max(0, z) \qquad f'(z) = \begin{cases} 1 & \text{iff } z > 0 \\ 0 & \text{iff } z < 0 \end{cases}$$

Problem: A bad initialization of the parameters can lead to a zero gradient. In practice, initialize the bias to a positive value.

13.5. Forward- and Backpropagation

- *Forward propagation* computes the activations for each layer, the outputs for each layer and the resulting loss function.
- *Backward propagation* computes the contribution of each parameter to the loss (the gradient) and updates the parameters using gradient descent.

13.5.1. Backpropagation

- Backpropagation, also known as *backprop*, calculates the gradient with the chain rule.
- Problems in multi-layer networks:
  - Non-convex, many local optima
  - Might get stuck in a poor local optima
  - The design of a working backpropagation algorithm is quite complex, causing the second winter of ML between 2000 and 2014.
- But these methods work very well!

Example In a simple neural network with no hidden layer and just one neuron per layer, the derivative for the bias $\frac{\partial L}{\partial b}$ computes as

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_1} \frac{\partial a_1}{\partial z_0} \frac{\partial z_0}{\partial a_0} \frac{\partial a_0}{\partial b}$$

The computations for the weight matrix $\frac{\partial L}{\partial W}$ are similar

Skip Connections

- For parameters that are closer to the input, the gradient needs to flow from the loss to those parameters.
- In very deep neural networks, the application of the chain rule may lead to a zero gradient, causing no learning to happen.
- One solution is to use *skip connections* to "jump" over layers.

13.5.2. Formulas

Notice that the loss function is not always squared error, so the derivative of the loss function might change. Let $\boldsymbol{y}^d$ be the desired output.

Forwardpropagation

$$\begin{aligned}
L(\boldsymbol{y}^d, \boldsymbol{y}) &= \frac{1}{2}(\boldsymbol{y}^d - \boldsymbol{y})^T(\boldsymbol{y}^d - \boldsymbol{y}) \\
\boldsymbol{y} &= \hat{W}_n \begin{bmatrix} \boldsymbol{a}_n^T & 1 \end{bmatrix} \\
\boldsymbol{a}_n &= \boldsymbol{f}_{n-1}(\boldsymbol{z}_{n-1}) \\
\boldsymbol{z}_{n-1} &= \hat{W}_{n-1} \begin{bmatrix} \boldsymbol{a}_{n-1}^T & 1 \end{bmatrix} \\
\boldsymbol{a}_{n-1} &= \boldsymbol{f}_{n-2}(\boldsymbol{z}_{n-2}) \\
\boldsymbol{z}_{n-2} &= \hat{W}_{n-2} \begin{bmatrix} \boldsymbol{a}_{n-2}^T & 1 \end{bmatrix} \\
&\vdots \\
\boldsymbol{a}_2 &= \boldsymbol{f}_1(\boldsymbol{z}_1) \\
\boldsymbol{z}_1 &= \hat{W}_1 \begin{bmatrix} \boldsymbol{a}_1^T & 1 \end{bmatrix} \\
\boldsymbol{a}_1 &= \boldsymbol{x}
\end{aligned}$$

Backpropagation

$$
\begin{aligned}
\mathrm{d}L &= -(\boldsymbol{y}^d - \boldsymbol{y})^T \,\mathrm{d}y \\
\mathrm{d}\boldsymbol{y} &= W_n \,\mathrm{d}\boldsymbol{a}_n \\
\mathrm{d}\boldsymbol{a}_n &= \boldsymbol{f}'_{n-1}(\boldsymbol{z}_{n-1}) \,\mathrm{d}\boldsymbol{z}_{n-1} \\
\mathrm{d}\boldsymbol{z}_{n-1} &= W_{n-1} \,\mathrm{d}\boldsymbol{a}_{n-1} \\
\mathrm{d}\boldsymbol{a}_{n-1} &= \boldsymbol{f}'_{n-2}(\boldsymbol{z}_{n-2}) \,\mathrm{d}\boldsymbol{z}_{n-2} \\
\mathrm{d}\boldsymbol{z}_{n-2} &= W_{n-2} \,\mathrm{d}\boldsymbol{a}_{n-2} \\
&\vdots \\
\mathrm{d}\boldsymbol{a}_2 &= \boldsymbol{f}'_1(\boldsymbol{z}_1) \,\mathrm{d}\boldsymbol{z}_1 \\
\mathrm{d}\boldsymbol{z}_1 &= W_1 \,\mathrm{d}\boldsymbol{a}_1 \\
\mathrm{d}\boldsymbol{a}_1 &= \mathrm{d}\boldsymbol{x}
\end{aligned}
$$

So $\mathrm{d}L$ can be computes as

$$
\mathrm{d}L = -(\boldsymbol{y}^d - \boldsymbol{y})^T \left(\prod_{k=n}^{K+1} W_k \boldsymbol{f}'_{k-1}(\boldsymbol{z}_{k-1}) \right) \mathrm{d}\boldsymbol{z}_K
$$

for all layers $K \in \{ 1, 2, \cdots, n \}$.

Vectorized

13.5.3. Approximating the Gradient

Instead of calculating backpropagation, the gradient can also be estimated using the finite differences for changes in each parameter W_j:

$$
\frac{\partial L}{\partial w_j} \approx \frac{L(\boldsymbol{w} + \varepsilon \boldsymbol{u}_j) - L(\boldsymbol{w})}{\varepsilon}
$$

where ε is a small perturbation and $\boldsymbol{u}_j$ is a unit vector in the j direction.

But, for a network with M parameters, forward propagation has to be done M times! This is very costly for large networks.

Backpropagation can compute the derivatives by forwardpropagate and backpropagate each one time, no matter how many parameters.

13.6. Gradient Descent

The basic update rule for gradient descent is

$$
\hat{W}^{k+1} = \hat{W}^k - \alpha \nabla_W L
$$

with the learning rate α and the gradient $\nabla_W L$ from backpropagation.

The key question are:

- How to update W?
- How to choose α?
- How to initialize W?

13.6.1. When to update W?

- **Full** Gradient Descent
  - Use the whole training set at once.
  - This is expensive for large data sets.

$$\nabla_W J = \frac{1}{n} \sum_{i=1}^{n} \nabla_W L(\boldsymbol{x}_i, \boldsymbol{y}_i, W)$$

- **Stochastic** Gradient Descent
  - Use one data point of the training set.
  - Needs an adaptive learning rate η_t with $\sum_{t=1}^{\infty} \eta_t = \infty$ and $\sum_{t=1}^{\infty} \eta_t^2 < \infty$.
  - The gradient estimation has a high variance.

$$\nabla_W J \approx \nabla_W L(\boldsymbol{x}_i, \boldsymbol{y}_i, W)$$

- **Mini-Batch** Gradient Descent
  - Use a subset of the training set.

$$\nabla_W J \approx \frac{1}{k} \sum_{i=1}^{k} \nabla_W L(\boldsymbol{x}_i, \boldsymbol{y}_i, W)$$

- The collected data can introduce a strong bias in successive data samples, so the data must be shuffled before applying stochastic or mini-batch gradient descent. This way, the bias can be reduced (but not removed).
- Nowadays, the usage of the term *stochastic gradient descent* refers to mini-batch gradient descent.

13.6.2. Adaptive Learning Rate

- A very high learning can increase the loss a lot, while a too low learning rate causes the algorithm to run long until convergence.
- Finding the right learning rate is pretty hard.
- Adaptive learning rates that change over time can help as the learning rate should be higher in flat regions, but small in valleys (to not "jump out").

Momentum

Insight

- Running Average

$$\bar{m}_0 = 0, \quad \bar{m}_{k+1} = \gamma_k \bar{m}_k + (1 - \gamma_k) m_k$$

- Geometric Average (constant γ)

$$\bar{m}_{k+1} = (1-\gamma)\sum_{i=1}^{k}\gamma^{k-i}m_i$$

- Arithmetic Average ($\gamma_k = \frac{k-1}{k}$)

$$\bar{m}_{k+1} = \frac{1}{k}\sum_{i=1}^{k} m_i$$

Practically Applied to momentum terms with $M_0 = 0$:

$$\begin{aligned} M_{k+1} &= \gamma_k M_k + (1-\gamma_k)\nabla_W J(W_k) \\ W_{k+1} &= W_k - \alpha_k M_{k+1} \end{aligned}$$

Adadelta

Insight Take large steps in plateaus as they do not have much risk and take smaller steps in steep areas.

Practically Normalize by the running average of the gradient norm with a small ε to prevent from dividing by zero, $V_0 = 0$ and the Hadamard product $\odot$:

$$\begin{aligned} G_k &= \nabla_W J(W_k) \\ V_{k+1} &= \gamma V_k + (1-\gamma) G_k \odot G_k \\ W_{k+1,ij} &= W_{k+1,ij} - \frac{\alpha_k}{\sqrt{V_{k,ij} + \varepsilon}} G_{k,ij} \end{aligned}$$

Note There exist two versions: One with the ε in and out of the square root, but both in the fraction.

Adam

Insight Combine momentum term with Adagrad.

Practically Just combine both equations:

$$\begin{aligned} G_k &= \nabla_W J(W_k) \\ V_{k+1} &= \gamma_1 V_k + (1-\gamma_1) G_k \odot G_k \\ M_{k+1} &= \gamma_2 M_k + (1-\gamma_2) G_k \\ W_{k+1,ij} &= W_{k+1,ij} - \frac{\alpha_k}{\sqrt{\eta_{\gamma_1^k} V_{k,ij} + \varepsilon}} \eta_{\gamma_1^k} M_{k+1,ij} \end{aligned}$$

The initialization $V_0 = M_0 = 0$ leads to an underestimation fixed by $\eta_{\gamma_i^k} = \frac{1}{1-\gamma_i^k}$. With $\gamma_1 = 0.9$, $\gamma_2 = 0.999$ and $\varepsilon = 10^{-8}$, Adam is not too sensitive to parameter changes.

Note Adam violates the convergence guarantees…

13.6.3. Small Neural Networks

For small neural networks, there exist better methods to get the direction of descent. But these are all too expensive for big networks.

Hessian Approaches

- Get second-order descent with $\delta \boldsymbol{w} = H^{-1} \nabla J$ and the Hessian $H = \nabla^2 J$.
- Estimate the Hessian with BFGS method.
- Use line search instead of a fixed learning rate.

Conjugate Gradient

- Momentum term with variable learning rate, e.g.

  $$\delta \boldsymbol{w}_t = \nabla J(\boldsymbol{w}_t) + \frac{\|\nabla J(\boldsymbol{w}_t)\|^2}{\|\nabla J(\boldsymbol{w}_{t-1})\|^2} \delta \boldsymbol{w}_t$$

  with Powell restarts.
- Problem: Does not work well with stochastic gradient descent.

Levenberg-Marquart Linearize the network

$$f(\boldsymbol{x}_i, \boldsymbol{w}) = f(\boldsymbol{x}_i, b) + \nabla_{\boldsymbol{w}} f(\boldsymbol{x}_i, \boldsymbol{b})\big|_{\boldsymbol{w}=b}^T \delta \boldsymbol{w} = \boldsymbol{f}_{i0} + \boldsymbol{J}_i \delta \boldsymbol{w}$$

and solve the least squares regression problem

$$J \approx \frac{1}{2} \|\boldsymbol{y} - (\boldsymbol{f}_0 + \boldsymbol{J} \delta \boldsymbol{w})\|^2 + \frac{1}{2} \delta \boldsymbol{w}^T W \delta \boldsymbol{w}$$

yielding $\delta \boldsymbol{w} = (J^T J + W)^{-1} \boldsymbol{J}_i^T (\boldsymbol{y} - \boldsymbol{f}_0)$

- This is basically the Gauss-Newton-Method.
- Levenberg $W = \lambda I$ keeps the matrix invertible
- Marquardt $W = \lambda \operatorname{diag}(J^T J)$
- Adadelta approximates Levenbergs method parameterwise.

13.6.4. Initialization

Random Initialization

- Can lead to problems in gradient descent.
- For instance, large absolute values cause problems with sigmoid and negative values cause problems with ReLU.

Gaussian Initialization

- Draw the parameters from a Gaussian: $W_{kij} \sim \mathcal{N}(0, m^{-1})$, $b_k \sim \mathcal{N}(0, 1)$
- This basically normalizes the parameters.

Xavier/Normalized Initialization Initialize the weights from a uniform distribution (where n_i is the number of neurons in the i-th layer and W_j are the parameters of the layer connecting the hidden layer j and the next hidden layer $j+1$):

$$W_j \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \quad \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

Note: Xavier assumes that the activation functions are symmetric and linear around zero, so this works for $\tanh$ or sigmoid, but not for ReLU!

13.7. Overfitting

- Neural networks have hundreds, thousands or millions of parameters.
- But in most cases, no datasets with such many samples are available.
- So neural networks are prone to overfit.
- Overfitting can be fought with an algorithmic realization of a prior on the parameters:
  - **Regularization**
  - **Early stopping**
    Stop the training when the validation error starts rising again.
  - **Input noise augmentation**
    Adding noise $\boldsymbol{\epsilon}_i$ to the inputs reduces the chance of overfitting $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \boldsymbol{\epsilon}_i$.
  - **Dropout**
    Focus efficiently on the relevant neurons and prune others by zeroing out the weights intermittently and letting a subset of neurons predict:

    $$a_i = f_i(z)d_i \text{ with } d_i \in \{0, 1\} \text{ and } p(d_i = 1) = p_{\text{dropout}} = 0.5$$

  - **Weight decay**
    A ridge loss $J(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \boldsymbol{w}^T \boldsymbol{w}$ yields weight decay:

    $$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \alpha_k\big(\nabla_{\boldsymbol{w}} L(\boldsymbol{w}_k) + \lambda \boldsymbol{w}_k\big) = (1 - \alpha_k)\boldsymbol{w}_k + \alpha_k \nabla_{\boldsymbol{w}} L(\boldsymbol{w}_k)$$

13.7.1. Batch Normalization

- Covariate shift
  - Changes in the input distribution make learning hard.
  - This is especially problematic with mini-batches.
  - Hidden values change as their preceding layers change.

- This can be fought by *batch normalization*:

$$\tilde{x}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

  - This is like dropout with better performance.
  - Similar to normalization in ridge regression.
  - More complex: removal of batch normalization.

13.8. Theoretical Results

- The features are learned rather than hand-crafted by the machine learner.
- More layers capture more invariances.
- More data is needed to train deeper networks.
- More computation power (e.g. on GPUs).
- Better regularization methods like dropout.
- New nonlinearities: max pooling, ReLU
- However, the understanding of what deep networks really so remains shallow.
- Theory Fields
  - Approximation, depth width and invariance theory
  - Generalization and regularization theory

13.9. Other Network Architectures

13.9.1. Convolutional Neural Network (CNN)

- *Convolutional neural networks* (CNNs) are particularly suited for feature extraction in spatially correlated data like images.
- Feature maps are computed by applying convolutional kernels to the input or feature maps.
- Pooling reduces the dimensionality. For instance, max_pooling(k) takes the pixels with the largest values among k neighboring pixels.
- Instead of computing the pre-activation of a layer with a matrix, use a convolution operation:

$$s(t) = (x * w)(t) = \int x(a)\, w\,(t-a)\,\mathrm{d}a$$

  where x is the input signal and w is often called the kernel.
- This acts as a filter on the input.

Fully Connected vs. Convolutional

- **Fully Connected**
  - With high dimensional input data the number of parameters explodes (gray image with 1000x1000 pixels, hidden layer with 1000 neurons has 1 billion parameters, just for the first layer).
  - Does not extract local features which are usually present in images.
- **Convolutional**
  - The learned parameters are the kernel weights which are much smaller than the input and shared over the whole input.
  - Computes local features since the output of a kernel involves a computation over adjacent pixels.

13.9.2. Recurrent Neural Network (RNN)

- *Recurrent neural networks* (RNNs) are networks with memory where the output is fed into the input again.
- This can be used for time dependent/series data:
  - Natural language processing
  - Speech recognition
  - Dynamical systems
  - Stock market
  - Brain-computer interface
  - etc.

13.9.3. Long Short-Term Memory Network (LSTM)

- Gradient computation in RNNs is done with *backpropagation through time* (BPTT). A parameter is updates by adding all contributions to the loss over time.
- This leads to vanishing and exploding gradients.
- *Long short-term memory networks* (LSTMs) fight the gradient problems with a different architecture to let the gradient flow better in BPTT and are thus capable of more efficient learning than traditional RNNs.

13.10. Applications

13.10.1. Computer Vision

13.10.2. Autonomous Systems

13.11. Radial Basis Function Networks

A multi-layer perceptron uses univariate projections to span the space of data.

- Pros
  - Universal function approximation
  - Large range generalization (extrapolation)
  - Good for high dimensional data
- Cons
  - Hard to train
  - Danger of interference

Radial basis function networks (RBFNs) use a different approach:

- Only one hidden layer
- Use spatially localized kernels for learning (note: there are basis functions that are not spatially localized).
- They use radial basis functions as activation functions, i.e. functions $\phi\big(\|\boldsymbol{x}-\boldsymbol{c}\|\big)$ that only depend on the norm of some data $\boldsymbol{x}$ around some center $\boldsymbol{c}$, e.g. the Gaussian kernel (note that k is the iteration of gradient descent):

  $$\phi(\boldsymbol{x},\boldsymbol{x}_k)=\exp\left\{-\frac{(\boldsymbol{x}-\boldsymbol{c}_k)^T D(\boldsymbol{x}-\boldsymbol{x}_k)}{2}\right\}$$

  with some positive definite D.
- The "output layer" then is just a linear regression $y=\sum_{i=1}^{k} w_i\phi(\boldsymbol{x},\boldsymbol{x}_k)=\boldsymbol{w}^R\Phi\phi(\boldsymbol{x},\boldsymbol{x}_k)$
- They often need regularization (e.g. ridge regression). The non-ridge case with squares loss yields the solution

  $$\boldsymbol{w}=(\Phi^T\Phi)^{-1}\Phi^T\boldsymbol{t}$$

  with $\boldsymbol{t}=\begin{bmatrix}t_1 & \cdots & t_n\end{bmatrix}^T$ and $\Phi=\begin{bmatrix}\phi_{11} & \cdots & \phi_{1m}\\ \vdots & \ddots & \dots\\ \phi_{n1} & \cdots & \phi_{nm}\end{bmatrix}$.
- The "input layer" can be optimized using gradient descent w.r.t. the distance metric and the center of the RBFs.
- Gradient descent can make D non-positive definite $\implies$ use Cholesky decomposition.
- An iterative procedure is needed for optimization, i.e. alternately update of $\boldsymbol{w}$ and $\boldsymbol{x}_k$ and D_k.
- Summarized, RBFs are powerful and efficient for learning, but the number of RBFs and the hyperparameter optimization is important and difficult!
- Theoretical remark: Poggio and Girosi (1990) showed that RBF networks arise naturally from minimizing the penalized cost function

  $$J=\frac{1}{2}\sum_n\big(t_n-y(x_n)\big)^2+\frac{1}{2}\gamma\int\big|G(\boldsymbol{x})\big|^2\,\mathrm{d}\boldsymbol{x}$$

  with, e.g. $G(\boldsymbol{x})=\frac{\partial^2 y}{\partial\boldsymbol{x}^2}$, a smoothless prior.

13.12. Wrap-Up

- Neural networks and the relation to the brain
- Building of stacks of features
- Why one network layer is enough but impractical
- Forward and backwardpropagation
- Different ways of gradient descent
  - Full, stochastic, mini-batch
  - Speedup via learning rate adaption
  - Initialization of parameters
- Overfitting causes and defenses
- CNNs for spatially correlated data
- LSTMs for time series data
- RBF networks

14. Support Vector Machines

All machine learning is generally about lowering the structural risk bound in the true risk:

$$R(\boldsymbol{w}) \leq R_{\text{emp}}(\boldsymbol{w}) + \epsilon(N, p^*, h)$$

where N is the number of samples, p^* is the probability that the bound is met and h is the VC-dimension.

- Classical machine learning algorithms keep $\epsilon(N, p^*, h)$ constant and try to minimize $R_{\text{emp}}(\boldsymbol{w})$. The confidence interval is fixed by keeping some model parameters fixed, e.g. the number of neurons on a neural network.
- *Support vector machines* keep $R_{\text{emp}}(\boldsymbol{x})$ constant and minimize $\epsilon(N, p^*, h)$. With separable data, $R_{\text{emp}}(\boldsymbol{x}) = 0$. The confidence interval is controlled by changing the VC-dimension ("capacity control").

14.1. Linear SVMs

- Use linear classifiers.
- Approximate implementation of the structural risk minimization principle.
- If the data is linearly separable, the empirical risk of the SVM will be zero and the risk bound will be approximately minimized.
- SVMs have built-in "guaranteed" generalization abilities.

Assuming linearly separable data and given N sample training points $\{ \boldsymbol{x}_i, y_i \}_{i=1}^N$ with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{ -1, +1 \}$. The there exist a hyperplane $y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$ that separates the data. Intuitively, the "correct" hyperplane is the hyperplane with the maximum distance to the data of each class (called the *margin*). So the goal is to maximize this margin to minimize the VC-dimension.

Formally, this makes sense given the key result from Vapnik: If the data points lie in a sphere of radius R, $\|\boldsymbol{x}_i\| < R$ and the margin of the linear classifier in d dimensions is γ, then

$$h \leq \min \left\{ d, \quad \left\lceil \frac{2R^2}{\gamma^2} \right\rceil \right\}$$

So maximizing the margin lowers the upper bound on the VC-dimension!

Example Figure 14.1 shows a linear support vector machine working on a linear separable dataset with marked support vectors and the margin.

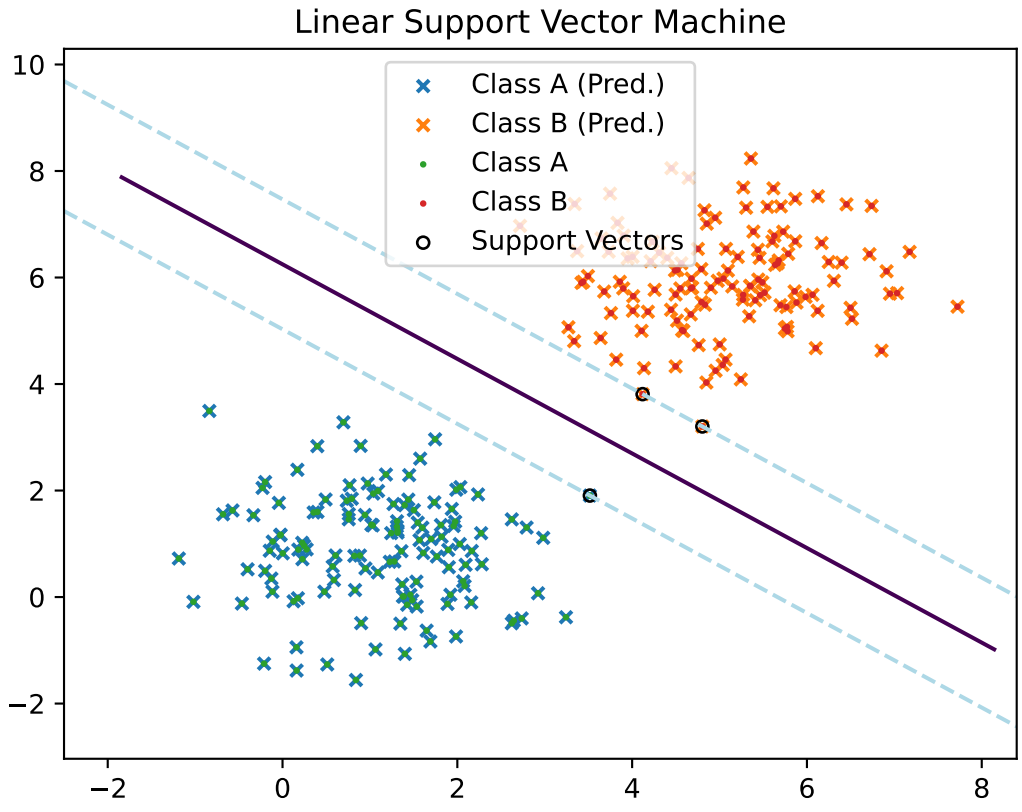

Figure 14.1.: Linear Support Vector Machine

14.1.1. Optimization Formulation

Find a hyperplane that separates the data linearly

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 \quad \forall i$$

and enforce $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = 1$ for at least one data point. The data points lie directly on the margin and are called *support vectors*.

The distance to the hyperplane is

$$\frac{y(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|} = \frac{\boldsymbol{w}^T\boldsymbol{x}_i + b}{\|\boldsymbol{w}\|}$$

so the margin is $\frac{1}{\|\boldsymbol{w}\|}$.

As maximizing the margin $\frac{1}{\|\boldsymbol{w}\|}$ is equivalent to minimize $\|\boldsymbol{w}\|^2$, the problem can be formulated as a quadratic minimization problem with linear constraints:

$$\begin{aligned} &\arg\min_{\boldsymbol{w},b} \; J(\boldsymbol{w}, b) = \frac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t.} \quad &y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 \geq 0 \quad \forall i \end{aligned}$$

This yields the Lagrangian formulation:

$$L(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N} \alpha_i \big(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1\big)$$

Taking the derivative w.r.t. $\boldsymbol{w}$ and b yields:

$$\begin{aligned} \frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial \boldsymbol{w}} &\overset{!}{=} 0 \implies \boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \\ \frac{\partial L(\boldsymbol{w}, b, \boldsymbol{\alpha})}{\partial b} &\overset{!}{=} 0 \implies \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned}$$

So the separating hyperplane is a linear combination of the input data. But the α_i are still unknown.

Dual Formulation

First rewrite the Lagrangian and then insert the equations 14.1.1 and 14.1.1 to get the dual:

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N} \alpha_i \big(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1\big) \\ &= \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{w}^T\boldsymbol{x}_i - b\sum_{i=1}^{N} \alpha_i y_i + \sum_{i=1}^{N} \alpha_i \end{aligned}$$

Using $\sum_{i=1}^{N} \alpha_i y_i = 0$:

$$\implies \qquad \hat{L}(\boldsymbol{w}, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{w}^T\boldsymbol{x}_i + \sum_{i=1}^{N} \alpha_i$$

Using $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$:

$$
\begin{aligned}
&= \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x}_j^T \boldsymbol{x}_i\right) + \sum_{i=1}^{N} \alpha_i \\
&= \frac{1}{2}\boldsymbol{w}^T \boldsymbol{w} - \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x}_j^T \boldsymbol{x}_i\right) + \sum_{i=1}^{N} \alpha_i \\
\implies \qquad \tilde{L}(\boldsymbol{\alpha}) &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x}_j^T \boldsymbol{x}_i\right) + \sum_{i=1}^{N} \alpha_i \\
&= \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x}_j^T \boldsymbol{x}_i\right)
\end{aligned}
$$

The last equation 14.1.1 is called the *Wolfe dual formulation*.

The original problem can now be solved by maximizing the dual function $\tilde{L}$:

$$
\begin{aligned}
\arg\min_{\boldsymbol{\alpha}} \tilde{L}(\alpha) &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\boldsymbol{x}_j^T \boldsymbol{x}_i\right) \\
\text{s.t.} \qquad \alpha_i &\geq 0 \quad \forall i \\
\sum_{i=1}^{N} \alpha_i y_i &= 0
\end{aligned}
$$

As almost all $\alpha_i \approx 0$, the separating hyperplane is given by N_S support vectors

$$\boldsymbol{w} = \sum_{i=1}^{N_S} \alpha_i y_i \boldsymbol{x}_i$$

The offset b can also be computed, but the derivation is skipped here:

$$b = \frac{1}{N_S}\sum_{i=1}^{N_s}\left(y_i - \sum_{j=1}^{N_S} \alpha_j y_j \left(\boldsymbol{x}^T \boldsymbol{x}\right)\right)$$

- Both the original (primal) SVM formulation and the dual rare quadratic optimization problems with linear constraints, called *quadratic programming problems*. These are convex and have a unique optima that can easily be compute, e.g. with libraries like `cvxopt`.
- The dual form is especially useful for nonlinear SVMs!

14.1.2. Sparsity

- Almost all α_i are roughly zero, so there are only a few support vectors.
- As the hyperplane was written as

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha y_i \boldsymbol{x}_i$$

it can be described by only a few data points.

- So by once calculating the support vectors, the SVM is described by only a few vectors that do not take up much memory/storage!

14.2. Nonlinear SVMs

Nonlinear SVMs are more powerful by using a feature transformation

$$\boldsymbol{x} \in \mathbb{R}^d \qquad \phi : \mathbb{R}^d \to H$$

into a (possible higher-dimensional) feature space H. The hyperplane is then written in this feature space, yielding a linear classifier in H

$$\boldsymbol{w}^T \phi(\boldsymbol{x}) + b = 0$$

But in R^d, the classifier is nonlinear and thus can separate nonlinear data. This is the same trick as in least-squares regression: Make the data linear separable rather than building a complex nonlinear classifier.

14.2.1. Optimization Formulation

The nonlinear dual form (with nonlinear transformations) can be obtained as

$$\begin{aligned} \arg\min_{\boldsymbol{\alpha}} \tilde{L}(\alpha) &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \big(\phi(\boldsymbol{x}_j)^T \phi(\boldsymbol{x}_i)\big) \\ \text{s.t.} \qquad & \alpha_i \geq 0 \quad \forall i \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned}$$

where the actual feature transformation $\phi(\boldsymbol{x}_i)$ only appears in scalar products with another $\phi(\boldsymbol{x}_j)$.

Also the discriminant function $y(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b$ can be written by just scalar products of the feature transformations (using $\boldsymbol{w} = \sum_{i=1}^{N_S} \alpha_i y_i \phi(\boldsymbol{x}_i)$):

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b = \sum_{i=1}^{N_S} \alpha_i y_i \big(\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x})\big) + b$$

This leads to the kernel trick.

14.2.2. Kernel Trick

As both the dual and the discriminant function can be written in terms of the scalar product of the features, only this has to be calculated. The *kernel trick* replaces every occurrence of the scalar product with a kernel function

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$

So if a function can be found that is equivalent to this scalar product, the mapping into a higher-dimensional features space can be avoided. This even means the feature space can be infinity-dimensional!

Polynomial Kernel

A polynomial kernel $K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y})^d$ of degree d is the kernel for a feature transformation into the space of all ordered monomials of degree d. The transformed space H then has the dimensionality

$$\dim(H) = \binom{d+N-1}{d}$$

where N is the dimension of the untransformed input space. The classifier then has the VC-dimension $\dim(H) + 1$.

Example: $d = 2$ For $d = 2$, the kernel becomes

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y})^2 = x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + y_1^2 y_2^2$$

which is equivalent to the scalar product

$$\phi(\boldsymbol{x})^T \phi(\boldsymbol{y}) = \begin{bmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{bmatrix}^T \begin{bmatrix} y_1^2 \\ \sqrt{2} y_1 y_2 \\ y_2^2 \end{bmatrix} = x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2$$

Radial Basis Function Kernel (RBF)

$$K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left\{ - \frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2} \right\}$$

- Measures the similarity between $\boldsymbol{x}$ and $\boldsymbol{y}$.
- Construct an infinite-dimensional feature space H so that the hyperplane also has infinity VC-dimension.
- If the radius σ of the kernel is chosen too low, every data point has its "own" kernel, leading to massive overfitting. So the radius has to be bound to limit the VC-dimension.

Mercer's Condition

To check whether a kernel really is a kernel, Mercer's condition can be used: A function $K(\boldsymbol{x}, \boldsymbol{y})$ is a valid kernel, if for every $g(\boldsymbol{x})$ with a converging integral $\int g(\boldsymbol{x})^2 \, \mathrm{d}\boldsymbol{x} < \infty$ it holds that

$$\iint K(\boldsymbol{x}, \boldsymbol{y}) \, g(\boldsymbol{x}) \, g(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{y} \geq 0$$

These kernels are known to satisfy Mercer's condition:

- Inhomogeneous polynomial kernel:

$$K(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T \boldsymbol{y} + c)^d$$

- Gaussian RBF kernel:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \exp\{-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\}$$

- Hyperbolic tangent kernel:

$$K(\boldsymbol{x}, \boldsymbol{y}) = \tanh\big(a \boldsymbol{x}^T \boldsymbol{y} + b\big)$$

Constructing Kernels

- Intuition: A kernel measures the similarity of two data points in the transformed space.
- The construction of a kernel most always be done by finding a feature transformation and encoding it in a kernel.
- The notion of similarity can also be encoded in the kernel function directly.
- Also, kernels can be combined with a few simple rules. Let $K_1(\boldsymbol{x}, \boldsymbol{y})$ and $K_2(\boldsymbol{x}, \boldsymbol{y})$ are valid kernels, then all of the following are also valid kernels:

$$
\begin{gathered}
cK_1(\boldsymbol{x}, \boldsymbol{y}) \\
K_1(\boldsymbol{x}, \boldsymbol{y}) + K_2(\boldsymbol{x}, \boldsymbol{y}) \\
K_1(\boldsymbol{x}, \boldsymbol{y})\, K_2(\boldsymbol{x}, \boldsymbol{y}) \\
f(\boldsymbol{x})\, K_2(\boldsymbol{x}, \boldsymbol{y})\, f(\boldsymbol{y}) \\
\dots
\end{gathered}
$$

14.3. Non-Separable Data

If the data is not separable, there are multiple solutions:

1. Simple solution: Transform the data points into a higher dimensional feature space so that they become linearly separable. But this leads to a high VC-dimension and is prone to overfit.
2. Better solution: Let some data point allow to violate the margin.

14.3.1. Slack Variables

Introduce *slack variables* $\xi_i \geq 0$ that, instead of requiring perfect linearly separable with

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1$$

allow small violations from perfect separation:

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) \geq 1 - \xi_i$$

This allows the data points to be off by ξ_i from the margin.

Even if the data is separable, it may be good to introduce slack variable for an occasional penalty.

Optimization Formulation

This changes the optimization problem to:

$$
\begin{aligned}
& \arg\min_{\boldsymbol{w}, b} J(\boldsymbol{w}, b) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{N} \xi_i \\
\text{s.t.} \quad & y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 + \xi_i \geq 0 \quad \forall i \\
& \xi_i \geq 0 \quad \forall i
\end{aligned}
$$

with the weight C specifying the tradeoff that is made. A larger C allows more violations.

This introduces a *box constraint* $0 \leq \alpha_i \leq C$ to the dual formulation:

$$\begin{aligned} \arg\min_{\boldsymbol{\alpha}} \tilde{L}(\alpha) &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \left(\phi(\boldsymbol{x}_j)^T \phi(\boldsymbol{x}_i)\right) \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \quad \forall i \\ & \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned}$$

This is also a quadratic programming problem and thus can be solved efficiently.

14.3.2. Lack of Sparseness

- If there is a large class overlap, SVMs may need many support vectors.
- This reduces the sparsity.
- Alternative: Relevant vector machines (RVMs)
  - Probabilistic alternative to SVMs.
  - Gives much sparser results.
  - But no notion of margin maximization.

14.4. Applications

14.4.1. Text Classification

- Problem: Classify document into a number of categories.
- The text is represented using word statistics, i.e. histograms of the frequency.
  - Count how often every word occurs and ignore their order (“bag of words”).
  - Very high-dimensional feature space (roughly 10 000 dimensions).
  - Very few features that are not relevant (thus it is not feasible to apply dimensionality reduction).
- SVMs are doing really well on this problem.

14.4.2. Handwritten Digit Classification

14.4.3. Support Vector Regression

SVMs may also be adapted to regression tasks, but this does not work very well.

14.5. Wrap-Up

- Main idea of SVMs
- Reason for maximizing the margin
- Translation of the SVM problem into an quadratic programming problem
- Interpretation of the support vectors
- Using SVMs for non-linearly separable data
- Kernel trick
- Construction of kernels
- Formulation of SVMs with slack variables

A. Self-Test Questions

The text below also contains answers for the self-test questions! To prevent you from reading the answers accidentally, the questions start on a new page after the demo questions.

A.1. Demo

In what case can you toggle the visibility of the answers?

Answer If my PDF viewer supports it.

In what case can you definitely not toggle the visibilities of the answers?

Answer If I have printed the document.

A.2. Organization

What are some of Machine Learning applications?

Answer For example natural language processing and autonomous driving.

When can we benefit from using Machine Learning methods?

Answer Machine learning can be helpful if the problem is too hard to program by hand (e.g. image recognition and natural language processing).

What are the different types of learning?

Answer

- Supervised: Given labeled data (input/output pairs).
- Unsupervised Given unlabeled data (only input).
- Semi-Supervised: Given some labeled and some unlabeled data.
- Reinforcement Learning: No data given.

What is the difference between classification and regression? Can you give some examples of both tasks (and identify the domain and codomain)?

Answer

- Classification sorts data into discrete classes. A sample use case is the recognition of hand-written digits. The domain are images and the codomain are the number from zero to one.
- Regression maps data onto a continuous output space and is able to extrapolate missing data. A sample use case is the analysis and prediction of weather. The domain are date or date-times and the codomain may be the temperature.

What are the challenges when solving a Machine Learning problem?

Answer

- Generalization: The learned function should generalize and work for new data and not only for the training data, called
- Overfitting: The algorithm just “memorized” the learning data and cannot handle other (new) data.
- Features: Choosing the right features is hard but important.
- Curse of dimensionality: Too high-dimensional features cause problems.

What is generalization? What is overfitting?

Answer Overfitting is quite the opposite of generalization. If an algorithm overfits, it just memorizes the training data and is not capable of handling new data. If it is, the function ha generalized.

A.3. Linear Algebra Refresher

Remember vectors and what you can do with them.

Answer Yeah I do remember.

Remember matrices and what you can do with them.

Answer Yeah I do remember.

What is a projection? How do you use it?

Answer N/A

How to compute the inverse of a matrix?

Answer One method is the Gaussian algorithm: You write the identity matrix to the right and the given matrix to the left and then transform both matrices in parallel until the identity matrix is on the left. Then the inverse is on the right.

What are Eigenvalues and Eigenvectors?

Answer The eigenvectors of a matrix are those (non-trivial) vectors that do not get rotated but only scaled when multiplied by the matrix. The corresponding eigenvalue of an eigenvector is the factor it gets scaled by.

What is a change of basis? What is a linear transformation? Are they the same?

Answer N/A

A.4. Statistics Refresher

What is a random variable?

Answer A random variable is a variable that can have multiple values that, if randomly sampled, follow a specific probability distribution.

What is a distribution?

Answer A probability distribution defines the probability that the value of a random variable falls into a specific region or has a specific value. It maps all possible values of the domain onto a probability between 0 and 1.

What is a Binomial distribution?

Answer The binomial distribution $\text{Bin}(k \mid N, \mu)$ is the probability that in N trials with the singular probability μ exactly k trials have been a success.

How does a Poisson distribution relate to Binomial distributions?

Answer The Poisson distribution is the Binomial distribution with $N \to \infty$.

What is a Gaussian distribution?

Answer The Gaussian distribution is the most common probability distribution and has some neat properties, e.g. that the sum of $N \to \infty$ random i.i.d. variables is Gaussian distributed. Its density function is given as

$$\mathcal{N}\left(x \mid \mu, \sigma^2\right) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\}$$

What is an expectation?

Answer The expectation value $\mathbb{E}(X)$ of a random variable X is the value that the random variable has in the mean. For continuous distributions with probability density $p(x)$, the expectation value is given as

$$\mathbb{E}(X) = \int_{-\infty}^{\infty} x p(x) \, \mathrm{d}x$$

What is a joint distribution?

Answer A joint distribution $p(x_1, \cdots, x_n)$ of n random variables is the probability that the tuple of both numbers fall into any particular range of set of values. For independent variables the distribution is $p(x, y) = p(x)\, p(y)$

What is a conditional distribution?

Answer The conditional distribution $p(x \mid y)$ is the probability of x given that y is true. It is given as

$$f(x \mid y) = \frac{f(x, y)}{f(y)}$$

What is a distribution with a lot of information?

Answer N/A

How to measure the difference between distributions?

Answer The difference (or similarity) between distribution can be measured using the Kullback-Leibler divergence (KL-divergence):

$$\mathrm{KL}(p \parallel q) = -\int p(x) \ln \frac{q(x)}{p(x)} \mathrm{d}x$$

A.5. Optimization Refresher

Why is optimization important for machine learning?

Answer Every machine learning problem is an optimization problem or can be reduced to be one.

What do well-formulated learning problems look like?

Answer Well-formulated problems have a cost function $J(\theta)$ that has to be minimized or maximized, given some equality constraints $f(\theta) = 0$ and some inequality constraints $g(\theta) \geq 0$. It is commonly notated like this:

$$\begin{aligned} &\arg\min_{\theta} \; J(\theta) \\ &\text{s.t.} \quad f(\theta) = 0 \\ &\qquad\; g(\theta) \geq 0 \end{aligned}$$

Every minimization problem can be a maximization problem and vice versa by multiplying the cost function with -1.

What is a convex set and what is a convex function?

Answer A convex set is a set where every point that lies on a line between any two points is also part of the set. Similarly, for a convex function, every line that can be drawn between any two points of the function does not cross the function. Formal: A set $C \subseteq \mathbb{R}^n$ is convex iff

$$\forall \boldsymbol{x}, \boldsymbol{y} \in C : \forall \alpha \in [0, 1] : \alpha \boldsymbol{x} + (1 - \alpha)\boldsymbol{y} \in C$$

and a function $f : \mathbb{R}^n \to \mathbb{R}$ is convex iff

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \text{Domain}(f) : \forall \alpha \in [0, 1] : (\alpha \boldsymbol{x} + (1 - \alpha)\boldsymbol{y}) \leq \alpha f(\boldsymbol{x}) + (1 - \alpha) f(\boldsymbol{y})$$

How do I find the maximum of a vector-valued function?

Answer Take the gradient w.r.t. to the variable, yielding a vectorial gradient. Then set each component to zero and solve for the variable (this may be complicated due to overdetermined equation systems and similar).

How to deal with constrained optimization problems?

Answer Formulate the Lagrangian $L(\theta) = J(\theta) + \lambda f(\theta) + \mu g(\theta)$, take the derivatives w.r.t. θ and the Lagrangian multipliers λ and μ, set them to zero and solve for θ.

How to solve such problems numerically?

Answer One method for solving them is gradient descent, also called steepest descent. It works by calculating the gradient and then reducing the value of θ by the gradient iteratively. For maximization, the gradient has to be added.

A.6. Bayesian Decision Theory

How can we decide on classifying a query based on simple and general loss functions?

Answer N/A

What does "Bayes Optimal" mean?

Answer A "Bayes optimal" classifier obeys the rule that it chooses class C_1 over C_2 iff $\frac{p(x \mid C_1)}{p(x \mid C_2)} > \frac{p(C_2)}{p(C_1)}$.

How to deal with two or more classes?

Answer Choose class C_i iff $\frac{p(x \mid C_i)}{p(x \mid C_j)} > \frac{p(C_j)}{p(C_i)} \quad \forall i \neq j$.

How to deal with high dimensional feature vectors?

Answer The decision rules still apply, but the posterior probability densities $p(\boldsymbol{x} \mid C_k)$ have to handle multiple features (be multivariate).

How to incorporate prior knowledge on the class distribution?

Answer This is done through the prior $p(C_k)$ for class C_k which can be determined, e.g. by simple counting (if and only if the sample data points are representative).

What are the equations for misclassification rate and risk?

Answer The risk can be encoded as $\lambda(\alpha_i \mid C_j)$, which is the loss of classifying x as class C_i if C_j is the actual class. The risk is then encoded as $R(\alpha_i \mid x) = \mathbb{E}_{C_k \sim p(C_k \mid x)}\big(\lambda(\alpha_i \mid C_k)\big) = \sum_j \lambda(\alpha_i \mid C_j)\, p(C_j \mid x)$ which is the expected risk for classifying x as class C_i. Then decide for the class with the lowest risk.

A.7. Probability Density Estimation

Where do we get the probability of data from?

Answer The probability densities can be estimated if sample data is available. This problem is called "Probability Density Estimation".

What are parametric methods and how to obtain their parameters?

Answer Parametric models depend on distributions like Gaussians that have specific parameters (e.g. the mean μ and the variance σ^2). The values of these parameters can then by obtained by estimation, e.g. vi maximum likelihood or maximum a-posteriori, where the last one is a Bayesian approach.

How many parameters have non-parametric methods?

Answer Non-parametric models have any number of parameters as the raw data is used as "parameters".

What are mixture models?

Answer Mixture models are built out of multiple single probability densities. They are all added together with a prior π_i, which is the probability that a data point is samples from the i-th distribution (also called "weight"). The general formula is $p(x) = \sum_i \pi_i p_i(x)$, where π_i is the prior and $p_i(x)$ is the i-th probability density.

Should gradient methods be used for training mixture models?

Answer No, because the derivatives of these models contain cyclic dependencies on the other parameters which makes gradient methods mostly useless and slow.

How does the EM algorithm work?

Answer The whole idea behind EM is to maximize the complete log-likelihood $Q(\theta, \theta^{i-1}) = \int p(y \mid X, \theta^{(i-1)}) \ln p(X,$ in two steps:

- E-Step: Compute the probability density $p(y \mid X, \theta^{(i-1)})$ using the previously estimated (or initialized) parameters $\theta^{(i-1)}$.
- M-Step: Maximize the complete log-likelihood w.r.t. θ with maximum likelihood by using the values that have been computed in the E-Step.

What is the biggest problem of mixture models?

Answer The number of components and type of components that were used to draw the samples are typically unknown and it is (currently) impossible to determine them by an algorithm. There are some heuristics and trial and error can be used, but no more.

A.8. Clustering and Evaluation

How can we find meaningful clusters in the data?

Answer Cluster can be found, e.g. with mean shift clustering that starts at every data point and builds up nets of data points that are nearby (by climbing up the gradient). Another method is, for example, k-means.

How does density estimation with mixture models relate to clustering?

Answer Mixture models can generate clustered data, this can the estimation of them yield an estimation which pile of data was generated by which component. Thus, mixture model estimation is kind of a more powerful clustering method as it also yields the densities.

What is the bias-variance trade-off?

Answer An estimator can typically have a low bias or a low variance, but not both.

What is a BLUE estimator?

Answer An BLUE estimator ("best linear unbiased estimator") is an MVUE estimation that is linear in its features. An MVUE estimation has zero bias and a minimum of variance (called "minimum variance unbiased estimator").

Are maximum likelihood estimators always unbiased?

Answer No, they are not. E.g. the MLE for the variance of a Gaussian is biased with $\mathrm{Bias}(\hat{\sigma}^2) = -\frac{1}{N}\sigma^2$ where N is the number of samples and σ^2 is the real variance.

What is leave on out cross-validation? What do we need it for?

Answer In LOOCV, the whole data set is used for training with the exception of one data point that is used for testing. It is needed if not so many data is available to detect overfitting and to validate the model in general.

A.9. Regression

What is regression (in general) and linear regression (in particular)?

Answer Regression maps an input space to a continuous output space. Linear regression depends on function $y(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x})$ that are linear in the parameters.

What is the cost function of regression and how can I interpret it?

Answer The cost function of regression defines the penalty for a misclassified sample (e.g. least squares). The goal is to minimize this loss function and to have a way to get an actual value from the calculated probability density. This loss function is then minimized w.r.t. to the regression function $f(\boldsymbol{x})$ to get the actual function value. For least squares, this is just mean of he probability density, thus equal to $f(\boldsymbol{x})$.

What is overfitting?

Answer If the regressor overfits, it perfectly goes through the data points but does not really follow the actual function. This is due to the regressor just "memorizing" where the data points lie without generalizing.

How can I derive a Maximum-Likelihood Estimator for Regression?

Answer For MLE for regression, the given samples must be generated with some noise ϵ following some probability distribution, e.g. Gaussian $\epsilon \sim \mathcal{N}(0, \beta^{-1})$. The function value then also is a random variable $y \sim \mathcal{N}\left(f(\boldsymbol{x}), \beta^{-1}\right)$. An estimator for $f(\boldsymbol{x})$ and the precision β can then be derived by using the typical ML approach (take the derivative of the log-likelihood, set it to zero).

Why are Bayesian methods important?

Answer Bayesian methods allow to tame overfitting by putting a prior on the parameters, thus generating a probability distribution over the parameters. This gives much better and more accurate results.

What is MAP and how is it different to full Bayesian regression?

Answer MAP is like the ML approach to regression, but instead of maximizing the likelihood, the posterior $p(\boldsymbol{w} \mid X, \boldsymbol{y}, \alpha, \beta) \propto p(\boldsymbol{y} \mid X, \boldsymbol{w}, \beta)\, p(\boldsymbol{w}, \alpha)$ is maximized. This allows to put a prior on the parameters and thus regularizing the overfitting.

A.10. Classification

How do we get from Bayesian optimal decisions to discriminant functions?

Answer Discriminant functions model the class-conditional posterior that is used in Bayes decision rule directly, e.g. $y_1(x) = p(C_1 \mid x)$ and $y_2(x) = p(C_2 \mid x)$ with a combined discriminant function $y(x) = y_1(x) - y_2(x)$. Then decide for class C_1 iff $y(x) > 0$ and for class C_2 iff $y(x) < 0$, this is equivalent the Bayes optimal decision rule.

How to derive a discriminant function from a probability distribution?

Answer Given class-conditional posteriors $p(C_1 \mid x)$ and $p(C_2 \mid x)$, a discriminant function can be derived as $y(x) = p(C_1 \mid x) - p(C_2 \mid x)$ and similar if only the likelihood and the prior are given (the normalization term can be abandoned as its the same for both distributions, like in Bayes decision like).

How to deal with more than two classes?

Answer Build each discriminant function $y_i(x)$ to formulate how strong the classifiers believes in that class. Then decide for class C_i iff $y_i(x) > y_j(x)$ for all $i \neq qj$.

What does “linearly separable” mean?

Answer Intuitively, a line (or hyperplane) can be drawn through all data points while perfectly separating them (one class on the one side and the other on the other).

What is Fisher discriminant analysis? How does it relate to regression?

Answer Fisher’s discriminant analysis finds a projection through the data points where the data points then can be separated by a decision boundary (which is not given by Fisher’s discriminant analysis). This projection is similar to regression as it also finds a “line” through the data.

Is Fisher’s linear discriminant Bayes optimal?

Answer Yes, if the classes have equal and diagonal class-conditional likelihood covariance matrices.

What are perceptrons? How can we train them?

Answer Perceptrons are simple neural networks, typically with no hidden layer and just one output neuron. The basic perceptron (no hidden layer, one output neuron with the sign-“activation function”) is trained using the perceptron algorithm which is a version of gradient descent with the gradients “inserted”.

What is logistic regression? How can be derive the parameter update rule?

Answer Logistic regression formulates the class-conditional posterior as $p(C_1 \mid x) = \sigma(a)$ where it assumes that a is given by some linear discriminant function $a = \boldsymbol{w}^T \boldsymbol{x} + w_0$. The parameter update rule can then be derived by applying maximum likelihood estimation and then to gradient descent.

A.11. Linear Dimensionality Reduction and Statistical Learning Theory

What does dimensionality reduction mean?

Answer The goal is to find a dimension $D \ll N$ that is lower than the original dimension N, e.g. to reduce the computation cost in kernel regression (where a $D \times D$ matrix has to be inverted) or to visualize the data.

What is PCA? What are the three things that it does?

Answer Principal component analysis (PCA) finds the principal components of the data. That is, it finds the directions in which the variance is the highest and finds how high the variance is in this directions. It also finds the so-called “explained variance”, the amount of variance a component direction explains.

What are the roles of Eigenvectors and Eigenvalues in PCA?

Answer The eigenvectors are the principal components and the corresponding eigenvalues encode how much variance is explained in that direction.

Can you describe applications of PCA?

Answer PCA can be used to decompose images of faces into lower dimensions, change some parameters and then project it back into the original feature space. This can be used to morph images, e.g. to make them more masculine or feminine.

What does risk in statistical learning theory mean?

Answer "Risk" is the expectation of misclassifying a sample, which indirectly encodes the generalization abilities of an estimation. If the risk is high, it seems to overfit.

How is the true risk different from the empirical risk?

Answer The true risk depends on the underlying probability density via an integral over all data points and cannot be calculated directly, but is the real point of interest. The empirical risk applies the loss function onto some sample data points, giving an estimator for the true risk.

What is the learning power of a function approximator?

Answer The learning power expresses how much "capacity" an approximator has. The more learning power an approximator has, the more accurate can the approximations be, but this can also lead to overfitting.

What is expressed by a VC-Dimension?

Answer The VC-dimension specifies how much data points can be scattered by a function (or family of functions). For hyperplanes (linear functions), the VC-dimension is always $\dim(H)+1$, where H is the feature space.

Is the VC-Dimension always correlated with the number of parameters?

Answer No it is not, a counter example is the function

$$f(x, \boldsymbol{w}) = g\big(\sin(w_1 x + w_0)\big)$$
$$g(x) = \begin{cases} +1 & \text{iff } x > 0 \\ -1 & \text{iff } x \leq 0 \end{cases}$$

which has only two parameters but an infinite VC-dimension.

A.12. Neural Networks

How does logistic regression relate to neural networks?

Answer A NN with no hidden layer and just one output neuron equals logistic regression if the output layer has sigmoid as the activation function.

How do neural networks relate to the brain?

Answer The brain is made up of neurons that are connected with each other, grouped in so-called "sheets". Every neuron takes inputs from the previous sheet and "fires" if the "value" is above some threshold.

What kind of functions can single layer neural networks learn?

Answer Assuming this means "a single hidden layer", it can learn an arbitrary function, but the number of parameters are growing exponentially!

Why do two layers help? How many layers do you need to represent arbitrary functions?

Answer Two layers help to reduce the number of parameters needed to approximate a function, which eases the learning. Theoretically, one hidden layer is enough to represent arbitrary functions.

Why were neural networks abandoned in the 1970s, and later in the 1990s? Why did neural networks re-awaken in the 2010s?

Answer They were abandoned in the 1970s because of a book noticing that things like the perceptron do not work for simple nonlinear separable data like the XOR. In the 1990s they were abandoned because kernels were much better for optimization. In the 2010s, the big shift from too less data to too many data made them come back as now there is enough data to train such a network.

What output layer and loss function to use given the task (regression, classification)?

Answer

- Regression
  - Activation function: Linear
  - Loss function: Squared loss
- Classification
  - Activation function: Sigmoid for two-class and softmax for multi-class
  - Loss function: Nonlinear log-likelihood or cross-entropy

Why use a ReLU activation instead of a sigmoid?

Answer With sigmoid, nearly all regions of the gradient are zero, causing the learning to stop once it reaches that point. In ReLU, only the negative site has a zero gradient, causing the learning to progress better. But the success of ReLU highly depends on the initial weights. A negative initialization can cause "ReLU-networks" to not start to learn.

Derive the equations for forward and backwarpropagation for a simple network.

Answer Given a simple network with one hidden layer, an input and an output layer, all with just one neuron, the output computes as:

$$y = f_2(w_2 f_1(w_1 x_1 + b_1) + b_2)$$

or stepwise:

$$\begin{aligned} y &= f_2(a_2) \\ a_2 &= w_2 z_1 + b_2 \\ z_1 &= f_1(a_1) \\ a_1 &= w_1 x + b_1 \end{aligned}$$

Take the derivative of the loss w.r.t. w_1 to get the first gradient:

$$\begin{aligned} \frac{\partial L}{\partial w_1} &= L'(y)\frac{\partial y}{\partial w_1} \\ \frac{\partial y}{\partial w_1} &= f_2'(a_2)\frac{\partial a_2}{\partial w_1} \\ \frac{\partial a_2}{\partial w_1} &= w_2\frac{\partial z_1}{\partial w_1} \\ \frac{\partial z_1}{\partial w_1} &= f_1'(a_1)\frac{\partial a_1}{\partial w_1} \\ \frac{\partial a_1}{\partial w_1} &= x \end{aligned}$$

or more beautiful:

$$\frac{\partial y}{\partial w_1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial a_2}\frac{\partial a_2}{\partial z_1}\frac{\partial z_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

and w.r.t. w_2 to get the second:

$$\begin{aligned} \frac{\partial L}{\partial w_2} &= L'(y)\frac{\partial y}{\partial w_2} \\ \frac{\partial y}{\partial w_2} &= f_2'(a_2)\frac{\partial a_2}{\partial w_2} \\ \frac{\partial a_2}{\partial w_2} &= z_1 \end{aligned}$$

or more beautiful:

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

What is mini-batch gradient descent? Why use it instead of SGD or full gradient descent?

Answer Mini-batch gradient descent uses a subset of the samples for training (another one each iteration), thus reduces the computation cost which is the reason why to use it instead of full gradient descent. Stochastic gradient descent has a much higher variance thus is slow and leads to the parameters "jumping" around.

Why neural networks can overfit and what are the options to prevent it?

Answer Typically, a neural network has much more parameters than training data is available which makes it easy for the net to just memorize the data. Some of the options to prevent overfitting are regularization, early stopping (step when the validation error rises again), input noise augmentation (apply some noise to the input), dropout (randomly prune some neurons and train all others).

A.13. Support Vector Machines

How did learning theory motivate support vector machines?

Answer The typical machine learning algorithms try to minimize the empirical risk to lower an upper bound on the true risk. SVMs minimize the confidence interval $\epsilon(N, p^*, h)$ to lower the boundary by minimizing the VC-dimension.

What does maximum margin separation mean?

Answer The distance of the decision boundary between the classes to the nearest data points to that decision boundary is called margin. This margin is maximized to generalize as much as possible. Intuition: A decision boundary that is close to one of the classes, but far away from the others seems to not fit as well as a decision boundary that lies directly in the center of the classes, thus maximizing the margin to both classes.

Why did the SVM-craze drown the Neural-Networks-craze?

Answer Neural networks seem to be too hard to train/optimize, while kernel methods (like in kernel SVMs) are much easier to compute (especially because of quadratic programming). This caused the 2nd from 1994 in neural networks.

What is a Kernel?

Answer A kernel $K(x, y)$ is a function that is equivalent to the scalar product of feature transformations $\Phi(\cdot)$, so $K(x, y) = \phi(x)^T \phi(y)$ holds. If such a feature transformation only appears in scalar products, the kernel trick can be used to replace the products with a much easier computable kernel that can even represent feature transformations into an infinite feature space!

How can I build Kernels from Kernels?

Answer Kernels $K_1(x, y)$ and $K_2(x, y)$ can be combined, so all of the following are also valid kernels:

$$cK_1(x, y)$$
$$K_1(x, y) + K_2(x, y)$$
$$K_1(x, y)\, K_2(x, y)$$
$$f(x)\, K_1(x, y)\, f(y)$$

What functions does the Radial Basis Function Kernel contain?

Answer N/A

How does support vector regression work?

Answer N/A

A.14. Kernel Regression and Gaussian Processes

Why kernel methods for regression?

Answer Using kernel regression, the regression can work entirely in the feature space and can even consider infinite dimensional feature spaces. Also many other algorithms can be derived from the dual formulation of regression.

How do you get from radial basis functions to kernels?

Answer N/A

What is the role of the two pseudo-inverses in kernel regression?

Answer N/A

Why are kernel regression methods very computationally expensive?

Answer Because they have to invert an $N \times N$ matrix, where N is the number of sample data points.

Why is kernel regression the dual to linear regression?

Answer By formulating the dual of linear regression, it can be found that the feature transformations $\phi(\cdot)$ only appear in scalar product, thus allowing the kernel trick which is then called kernel regression.

What is the major advantage of GPs over Kernel Ridge Regression?

Answer Gaussian processes can also gauge the uncertainty of the estimate.

Why are GPs a Bayesian approach?

Answer Gaussian processes are Bayesian methods because they involve the construction of a prior distribution.

What principle allowed deriving GPs from a Bayesian regression point of view?

Answer N/A