

Data Mining / Intelligent Data Analysis

Christian Borgelt

Dept. of Mathematics / Dept. of Computer Science
Paris Lodron University of Salzburg
Hellbrunner Straße 34, 5020 Salzburg, Austria

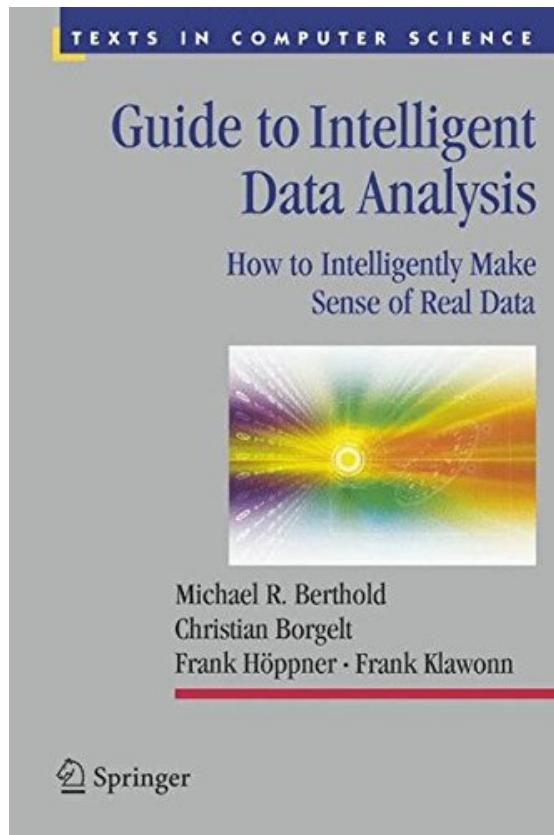
`christian.borgelt@sbg.ac.at`

`christian@borgelt.net`

`http://www.borgelt.net/`

`http://www.borgelt.net/teach/ida.html`

Bibliography



Textbook
Springer-Verlag
Heidelberg, DE 2010
(in English)

picture not available
in online version

picture not available
in online version

Textbook, 4th ed.
Morgan Kaufmann
Burlington, CA, USA 2016
(in English)

Textbook, 3rd ed.
Morgan Kaufmann
Burlington, CA, USA 2011
(in English)

Data Mining / Intelligent Data Analysis

- **Introduction**
- **Data and Knowledge**
 - Characteristics and Differences of Data and Knowledge
 - Quality Criteria for Knowledge
 - Example: Tycho Brahe and Johannes Kepler
- **Knowledge Discovery and Data Mining**
 - How to Find Knowledge?
 - The Knowledge Discovery Process (KDD Process)
 - Data Analysis / Data Mining Tasks
 - Data Analysis / Data Mining Methods
- **Summary**

Introduction

- Today every enterprise uses electronic information processing systems.
 - Production and distribution planning
 - Stock and supply management
 - Customer and personnel management
- Usually these systems are coupled with a database system (e.g. databases of customers, suppliers, parts etc.).
- Every possible individual piece of information can be retrieved.
- However: **Data alone are not enough.**
 - In a database one may “not see the wood for the trees”.
 - General patterns, structures, regularities go undetected.
 - Often such patterns can be exploited to increase turnover (e.g. joint sales in a supermarket).

Examples of Data

- “Columbus discovered America in 1492.”
- “Mr Jones owns a Volkswagen Golf.”

Characteristics of Data

- refer to single instances
(single objects, persons, events, points in time etc.)
- describe individual properties
- are often available in huge amounts (databases, archives)
- are usually easy to collect or to obtain
(e.g. cash registers with scanners in supermarkets, Internet)
- do not allow us to make predictions

Examples of Knowledge

- “All masses attract each other.”
- “Every day at 5 pm there runs a train from Hannover to Berlin.”

Characteristic of Knowledge

- refers to *classes* of instances
(*sets* of objects, persons, points in time etc.)
- describes general patterns, structure, laws, principles etc.
- consists of as few statements as possible (this is an objective!)
- is usually difficult to find or to obtain
(e.g. natural laws, education)
- allows us to make predictions

Criteria to Assess Knowledge

- Not all statements are equally important, equally substantial, equally useful.
⇒ Knowledge must be assessed.

Assessment Criteria

- Correctness (probability, success in tests)
- Generality (range of validity, conditions of validity)
- Usefulness (relevance, predictive power)
- Comprehensibility (simplicity, clarity, parsimony)
- Novelty (previously unknown, unexpected)

Priority

- Science: correctness, generality, simplicity
- Economy: usefulness, comprehensibility, novelty

Tycho Brahe (1546–1601)

Who was Tycho Brahe?

- Danish nobleman and astronomer
- In 1582 he built an observatory on the island of Ven (32 km NE of Copenhagen).
- He determined the positions of the sun, the moon and the planets (accuracy: one angle minute, without a telescope!).
- He recorded the motions of the celestial bodies for several years.

Brahe's Problem

- He could not summarize the data he had collected in a uniform and consistent scheme.
- The planetary system he developed (the so-called Tychonic system) did not stand the test of time.

Johannes Kepler (1571–1630)

Who was Johannes Kepler?

- German astronomer and assistant of Tycho Brahe.
- He advocated the Copernican planetary system.
- He tried all his life to find the laws that govern the motion of the planets.
- He started from the data that Tycho Brahe had collected.

Kepler's Laws

1. Each planet moves around the sun in an ellipse, with the sun at one focus.
2. The radius vector from the sun to the planet sweeps out equal areas in equal intervals of time.
3. The squares of the periods of any two planets are proportional to the cubes of the semi-major axes of their respective orbits: $T \sim a^{\frac{3}{2}}$.

How to find Knowledge?

We do not know any universal method to discover knowledge.

Problems

- Today huge amounts of data are available in databases.

*We are drowning in information,
but starving for knowledge.*

John Naisbett

- Manual methods of analysis have long ceased to be feasible.
- Simple aids (e.g. displaying data in charts) are too limited.

Attempts to Solve the Problems

- Intelligent Data Analysis
- Knowledge Discovery in Databases
- Data Mining

Knowledge Discovery and Data Mining

Knowledge Discovery and Data Mining

As a response to the challenge raised by the growing volume of data a new research area has emerged, which is usually characterized by one of the following phrases:

- **Knowledge Discovery in Databases (KDD)**

Usual characterization:

KDD is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. [Fayyad *et al.* 1996]

- **Data Mining (DM)**

- Data mining is that step of the knowledge discovery process in which data analysis methods are applied to find interesting patterns.
- It can be characterized by a set of types of tasks that have to be solved.
- It uses methods from a variety of research areas.
(statistics, databases, machine learning, artificial intelligence, soft computing etc.)

The Knowledge Discovery Process (KDD Process)

Preliminary Steps

- estimation of potential benefit
- definition of goals, feasibility study

Main Steps

- check data availability, data selection, if necessary: data collection
- preprocessing (60–80% of total overhead)
 - unification and transformation of data formats
 - data cleaning (error correction, outlier detection, imputation of missing values)
 - reduction / focusing (sample drawing, feature selection, prototype generation)
- **Data Mining** (using a variety of methods)
- visualization (also in parallel to preprocessing, data mining, and interpretation)
- interpretation, evaluation, and test of results
- deployment and documentation

The Knowledge Discovery Process (KDD Process)

pictures not available in online version

Typical depictions of the KDD Process

top: [Fayyad *et al.* 1996]

Knowledge Discovery and Data Mining:
Towards a Unifying Framework

right: **CRISP-DM** [Chapman *et al.* 1999]

CRoss Industry Standard Process
for Data Mining

Data Analysis / Data Mining Tasks

- **Classification**

Is this customer credit-worthy?

- **Segmentation, Clustering**

What groups of customers do I have?

- **Concept Description**

Which properties characterize fault-prone vehicles?

- **Prediction, Trend Analysis**

What will the exchange rate of the dollar be tomorrow?

- **Dependence/Association Analysis**

Which products are frequently bought together?

- **Deviation Analysis**

Are there seasonal or regional variations in turnover?

Data Analysis / Data Mining Methods 1

- **Classical Statistics**

(charts, parameter estimation, hypothesis testing, model selection, regression)

tasks: classification, prediction, trend analysis

- **Bayes Classifiers**

(probabilistic classification, naive and full Bayes classifiers, Bayesian network classifiers)

tasks: classification, prediction

- **Decision and Regression Trees / Random Forests**

(top down induction, attribute selection measures, pruning, random forests)

tasks: classification, prediction

- **k-nearest Neighbor / Case-based Reasoning**

(lazy learning, similarity measures, data structures for fast search)

tasks: classification, prediction

Data Analysis / Data Mining Methods 2

- **Artificial Neural Networks**

(multilayer perceptrons, radial basis function networks, learning vector quantization)

tasks: classification, prediction, clustering

- **Cluster Analysis**

(k -means and fuzzy clustering, Gaussian mixtures, hierarchical agglomerative clustering)

tasks: segmentation, clustering

- **Association Rule Induction**

(frequent item set mining, rule generation)

tasks: association analysis

- **Inductive Logic Programming**

(rule generation, version space, search strategies, declarative bias)

tasks: classification, association analysis, concept description

Statistics

Statistics

- **Descriptive Statistics**

- Tabular and Graphical Representations
- Characteristic Measures
- Principal Component Analysis

- **Inductive Statistics**

- Parameter Estimation
 - (point and interval estimation, finding estimators)
- Hypothesis Testing
 - (parameter test, goodness-of-fit test, dependence test)
- Model Selection
 - (information criteria, minimum description length)

- **Summary**

Statistics: Introduction

Statistics is the art to collect, to display, to analyze, and to interpret data in order to gain new knowledge.

[Sachs 1999]

[...] statistics, that is, the mathematical treatment of reality, [...]

Hannah Arendt [1906–1975]

There are three kinds of lies: lies, damned lies, and statistics.

Benjamin Disraeli [1804–1881]

Statistics, *n.* Exactly 76.4% of all statistics (including this one) are invented on the spot. However, in 83% of cases it is inappropriate to admit it.

The Devil's IT Dictionary

Basic Notions

- **Object, Case**

Data describe objects, cases, persons etc.

- **(Random) Sample**

The objects or cases described by a data set is called a *sample*, their number is the *sample size*.

- **Attribute**

Objects and cases are described by *attributes*, patients in a hospital, for example, by age, sex, blood pressure etc.

- **(Attribute) Value**

Attributes have different possible *values*.

The age of a patient, for example, is a non-negative number.

- **Sample Value**

The value an attribute has for an object in the sample is called *sample value*.

Scale Types / Attribute Types

Scale Type	Possible Operations	Examples
nominal (categorical, qualitative)	test for equality	sex/gender blood group
ordinal (rank scale, comparative)	test for equality greater/less than	exam grade wind strength
metric (interval scale, quantitative)	test for equality greater/less than difference maybe ratio	length weight time temperature

- Nominal scales are sometimes divided into *dichotomic* (binary, two values) and *polytomic* (more than two values).
- Metric scales may or may not allow us to form a **ratio** of values:
weight and length do, temperature does not.
time as duration does, time as calendar time does not.
- **Counts** may be considered as a special type (e.g. number of children).

Descriptive Statistics

Tabular Representations: Frequency Table

- Given data set: $\vec{x} = (3, 4, 3, 2, 5, 3, 1, 2, 4, 3, 3, 4, 4, 1, 5, 2, 2, 3, 5, 3, 2, 4, 3, 2, 3)$

a_k	h_k	r_k	$\sum_{i=1}^k h_i$	$\sum_{i=1}^k r_i$
1	2	$\frac{2}{25} = 0.08$	2	$\frac{2}{25} = 0.08$
2	6	$\frac{6}{25} = 0.24$	8	$\frac{8}{25} = 0.32$
3	9	$\frac{9}{25} = 0.36$	17	$\frac{17}{25} = 0.68$
4	5	$\frac{5}{25} = 0.20$	22	$\frac{22}{25} = 0.88$
5	3	$\frac{3}{25} = 0.12$	25	$\frac{25}{25} = 1.00$

- Absolute Frequency** h_k (frequency of an attribute value a_k in the sample).
- Relative Frequency** $r_k = \frac{h_k}{n}$, where n is the sample size (here $n = 25$).
- Cumulated Absolute/Relative Frequency** $\sum_{i=1}^k h_i$ and $\sum_{i=1}^k r_i$.

Tabular Representations: Contingency Tables

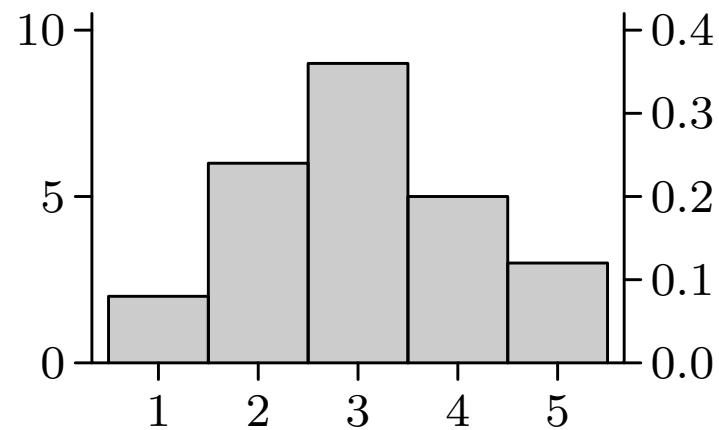
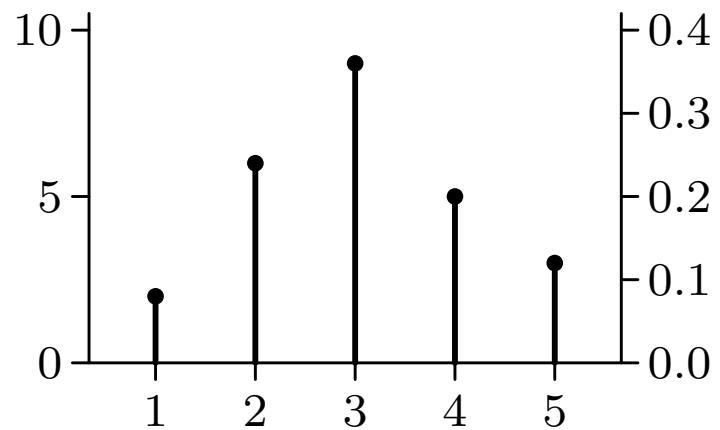
- Frequency tables for two or more attributes are called **contingency tables**.
- They contain the absolute or relative frequency of **value combinations**.

	a_1	a_2	a_3	a_4	\sum
b_1	8	3	5	2	18
b_2	2	6	1	3	12
b_3	4	1	2	7	14
\sum	14	10	8	12	44

- A contingency table may also contain the **marginal frequencies**, i.e., the frequencies of the values of individual attributes.
- Contingency tables for a higher number of dimensions (≥ 4) may be difficult to read.

Graphical Representations: Pole and Bar Chart

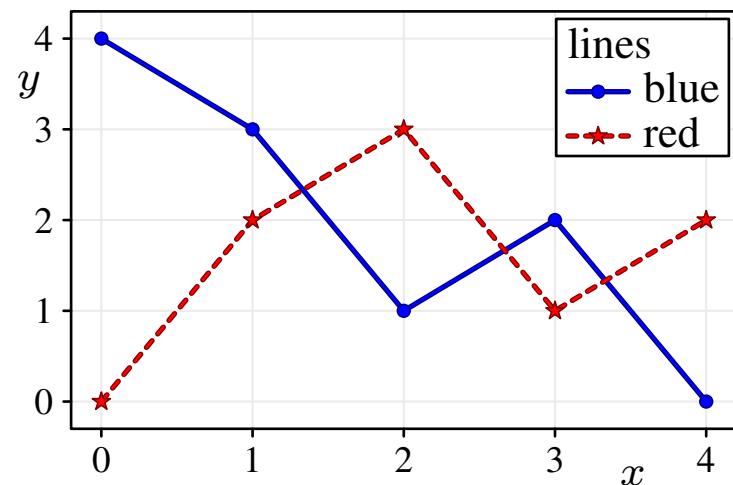
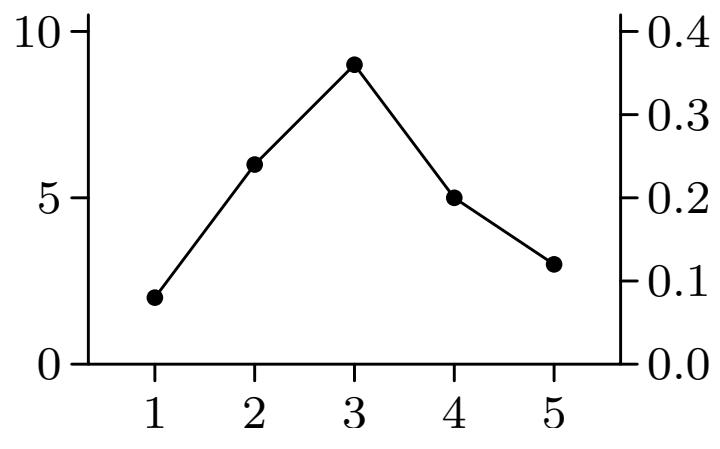
- Numbers, which may be, for example, the frequencies of attribute values are represented by the lengths of poles/sticks (left) or the height of bars (right).



- Bar charts are the most frequently used and most comprehensible way of displaying absolute frequencies.
- A wrong impression can result if the vertical scale does not start at 0 (for frequencies or other absolute numbers).

Frequency Polygon and Line Chart

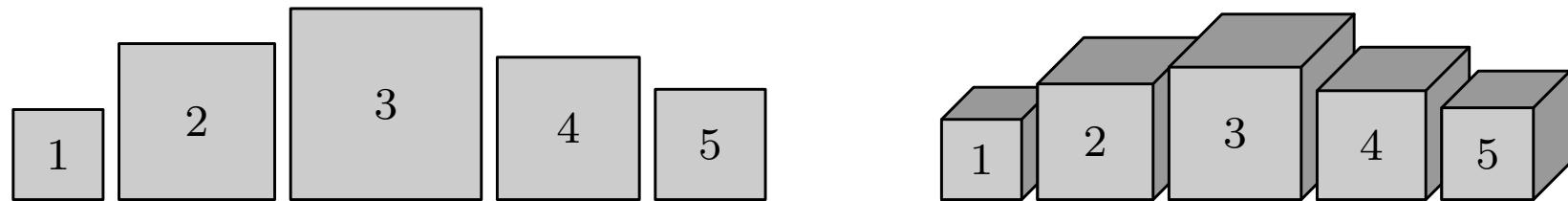
- Frequency polygon: the ends of the poles of a pole chart are connected by lines.
(Numbers are still represented by lengths.)



- If the attribute values on the horizontal axis are not ordered, connecting the ends of the poles does not make sense.
- Line charts are frequently used to display time series.

Area and Volume Charts

- Numbers may also be represented by geometric quantities other than lengths, like areas or volumes.
- Area and volume charts are usually less comprehensible than bar charts, because humans have more difficulties to compare and assess the relative size of areas and especially of volumes than lengths.
(exception: the represented numbers describe areas or volumes)

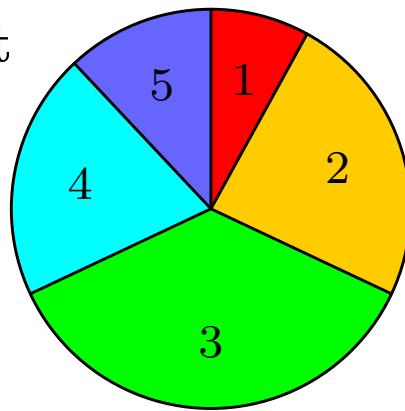


- Sometimes the height of a two- or three-dimensional object is used to represent a number. The diagram then conveys a misleading impression.

Pie and Stripe Charts

- Relative numbers may be represented by angles or sections of a stripe.

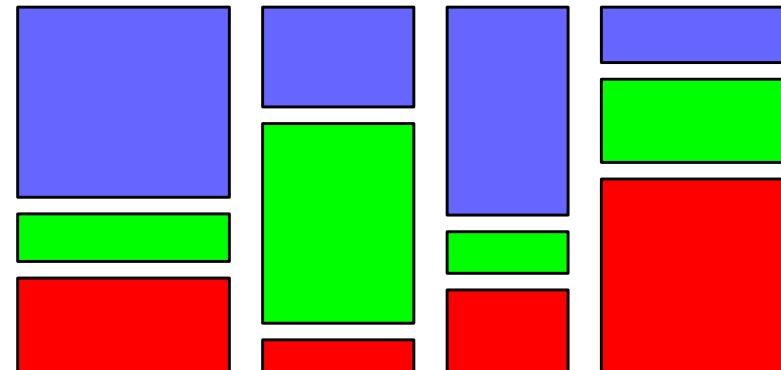
Pie Chart



Stripe Chart



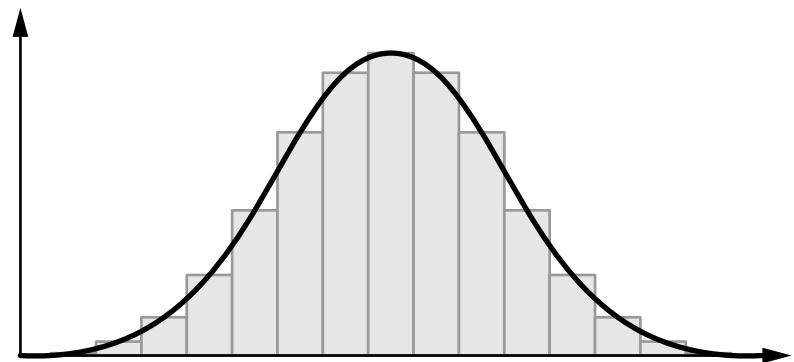
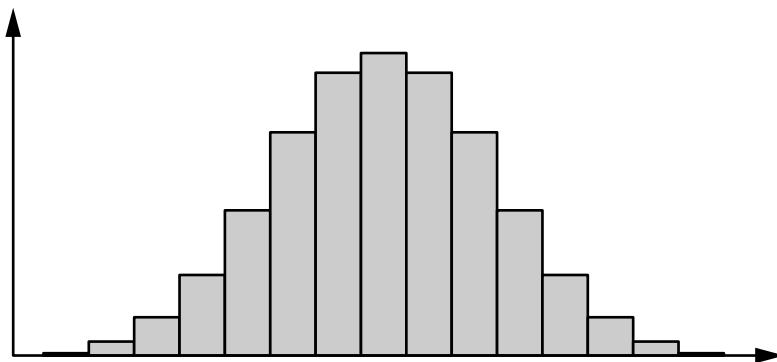
Mosaic Chart



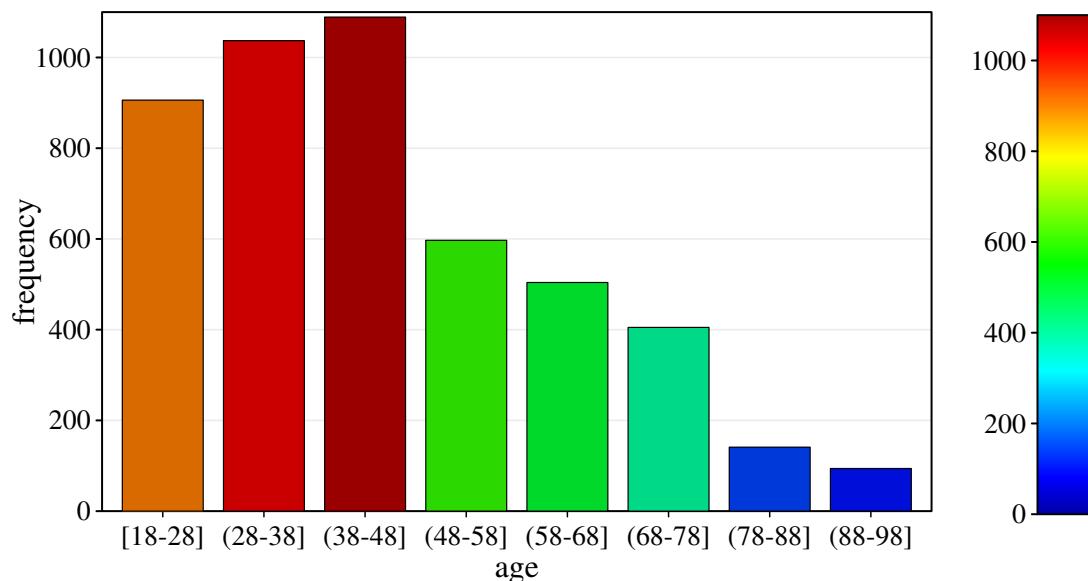
- Mosaic charts can be used to display contingency tables.
- More than two attributes are possible, but then separation distances and color must support the visualization to keep it comprehensible.

Histograms

- Intuitively: **Histograms are frequency bar charts for metric data.**
- However: Since there are so many different values,
values have to be grouped in order to arrive at a proper representation.
Most common approach: form equally sized intervals (so-called **bins**)
and count the frequency of sample values inside each interval.
- **Attention:** Depending on the size and the position of the bins
the histogram may look considerably different.
- In sketches often only a rough outline of a histogram is drawn:

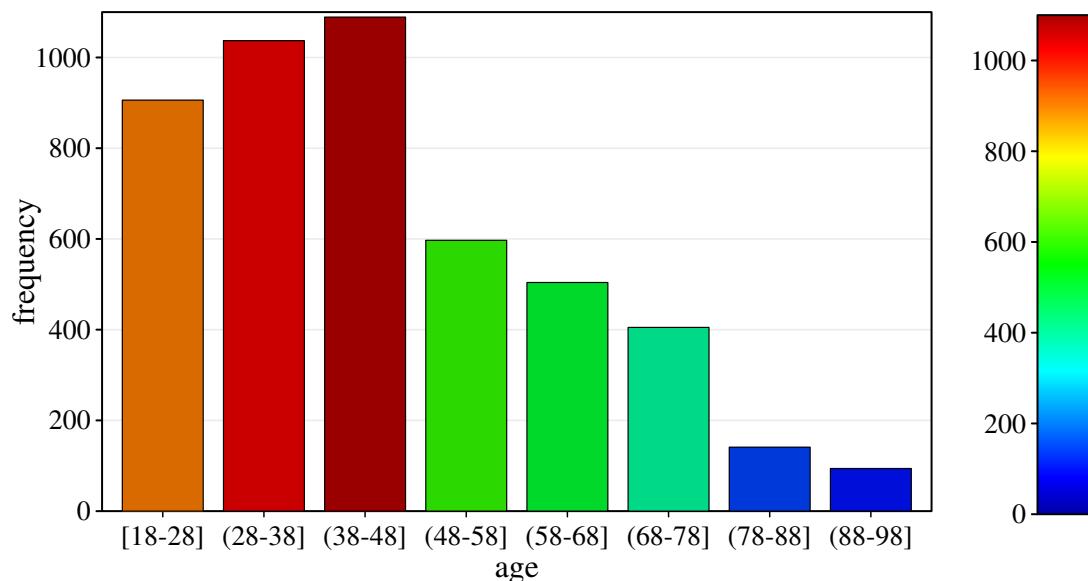


Histograms: Number of Bins

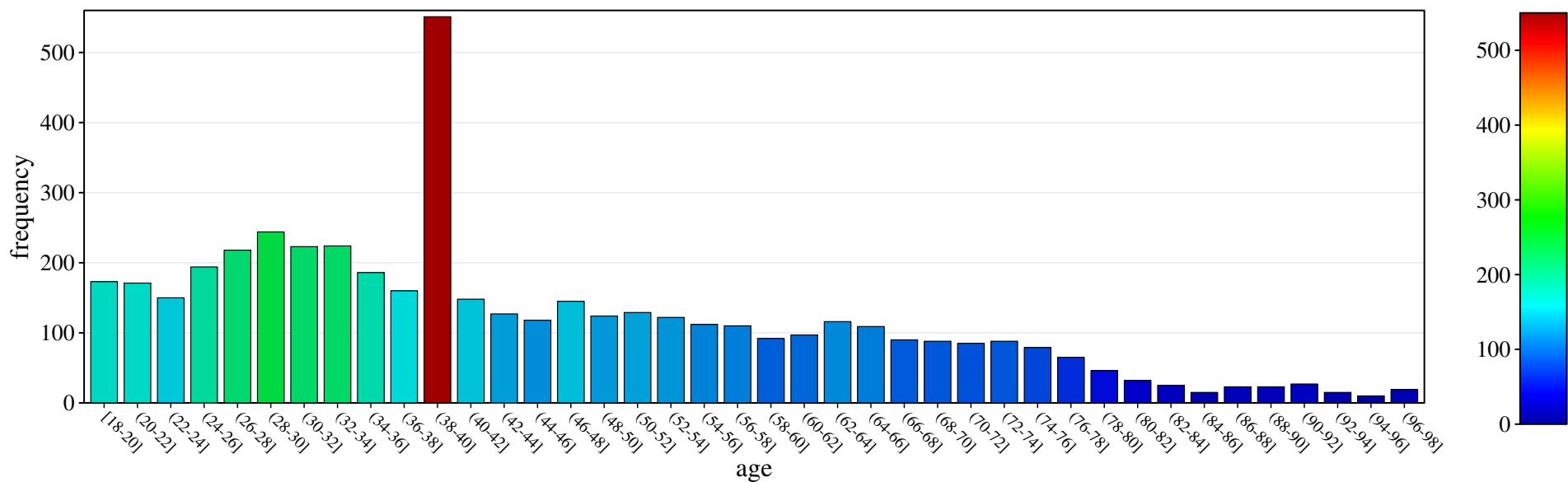


- Ages of customers of a supermarket/store (fictitious data); year of analysis: 2010.
- Depiction as a histogram indicates larger market share among younger people, but nothing is too conspicuous.

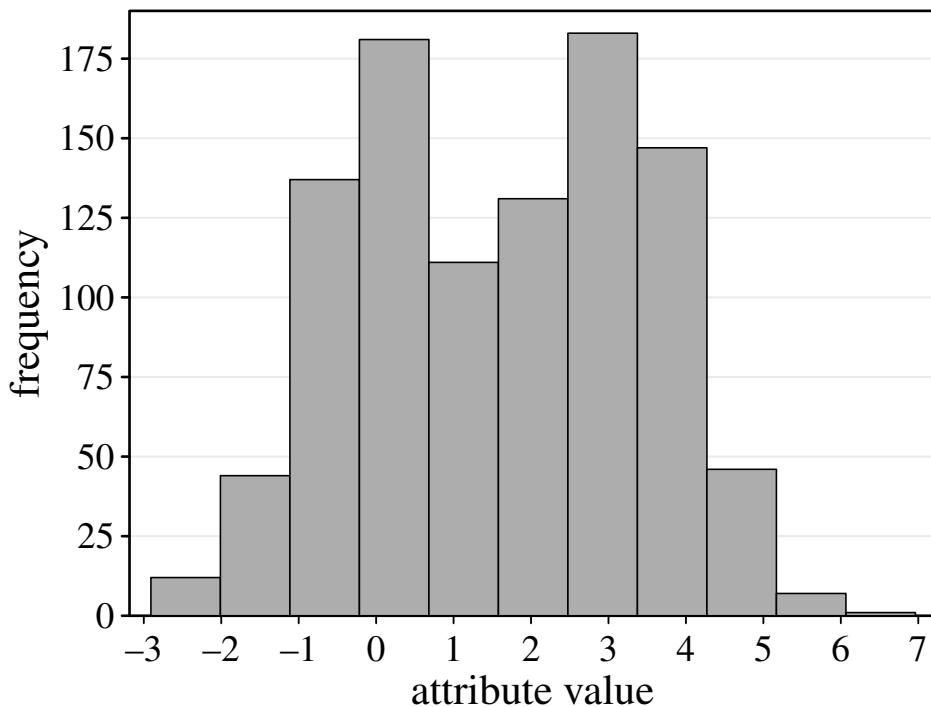
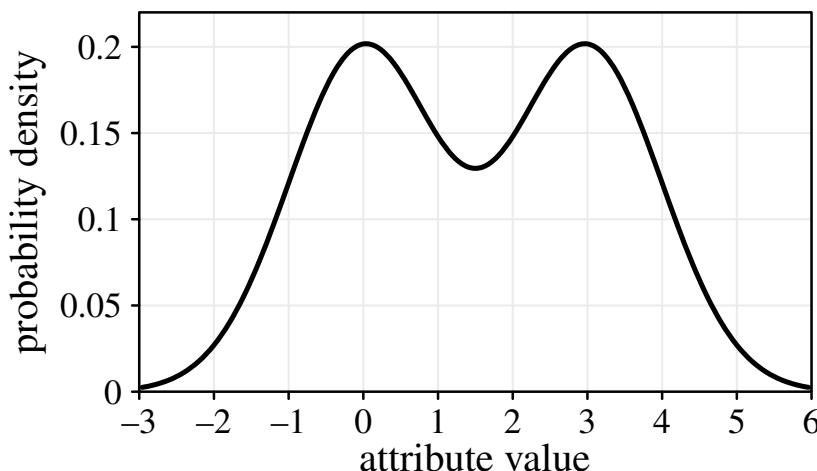
Histograms: Number of Bins



- Ages of customers of a supermarket/store (fictitious data); year of analysis: 2010.
- Depiction as a histogram indicates larger market share among younger people, but nothing is too conspicuous.

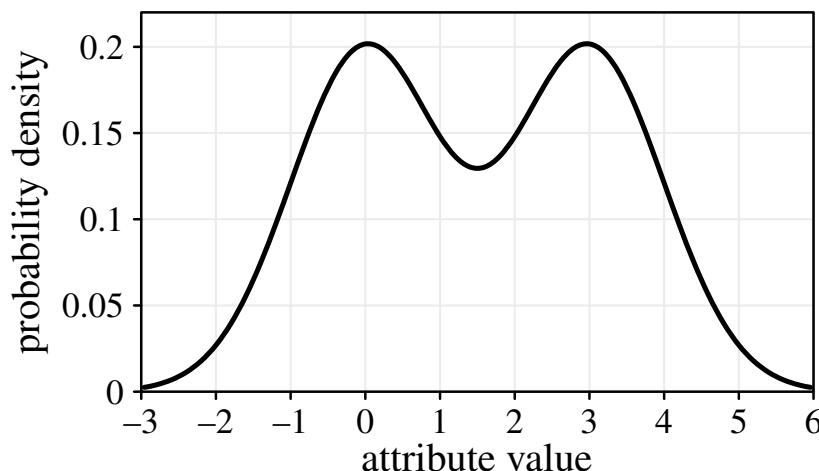


Histograms: Number of Bins

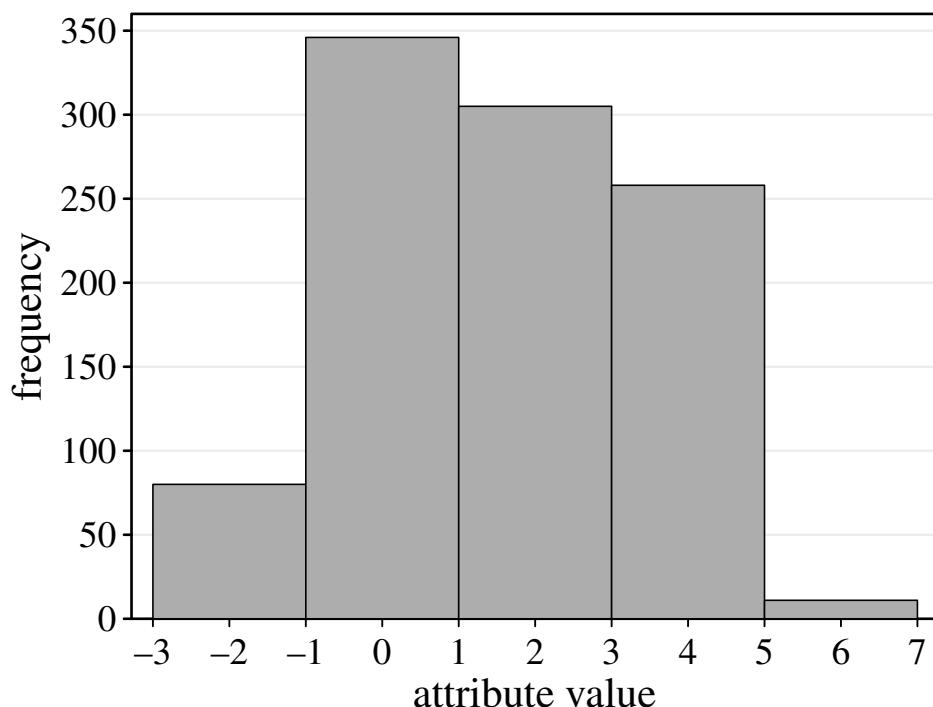


- Probability density function of a sample distribution, from which the data for the following histograms was sampled (1000 values).
- A histogram with 11 bins, for the data that was sampled from the above distribution.
- **How should we choose the number of bins?**
- **What happens if we choose badly?**

Histograms: Number of Bins

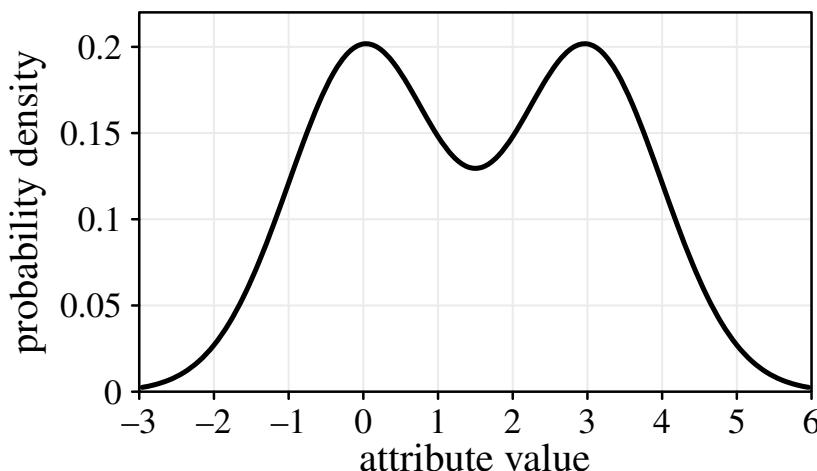


- Probability density function of a sample distribution, from which the data for the following histograms was sampled (1000 values).

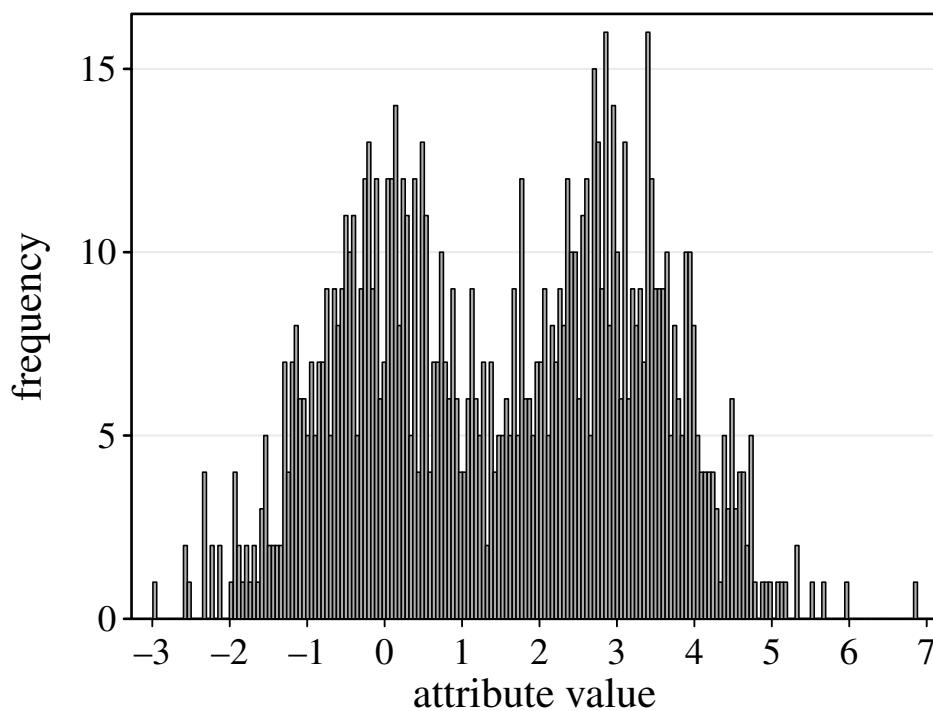


- A histogram with too few bins, for the same data as before.
- As a consequence of the low number of bins, the distribution looks unimodal (only one maximum), but skewed (asymmetric).

Histograms: Number of Bins

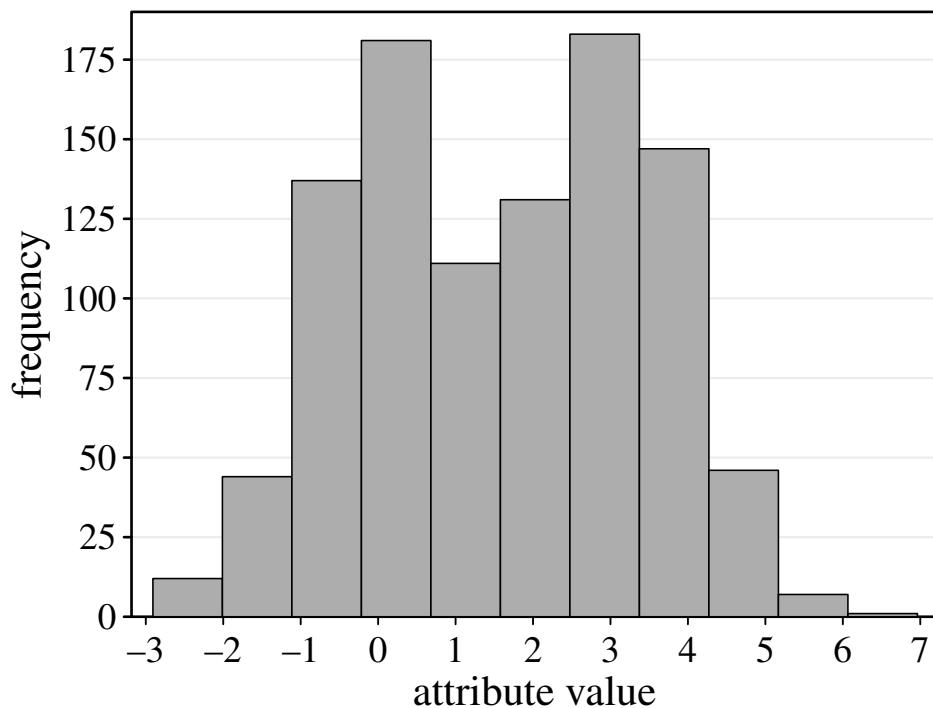
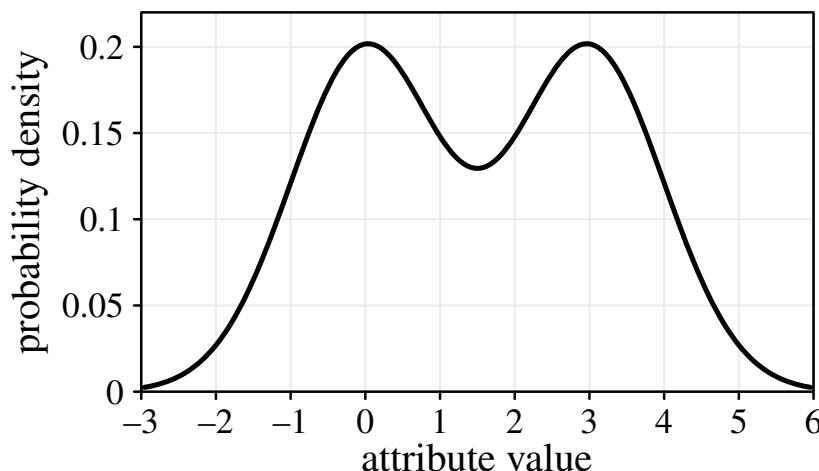


- Probability density function of a sample distribution, from which the data for the following histograms was sampled (1000 values).



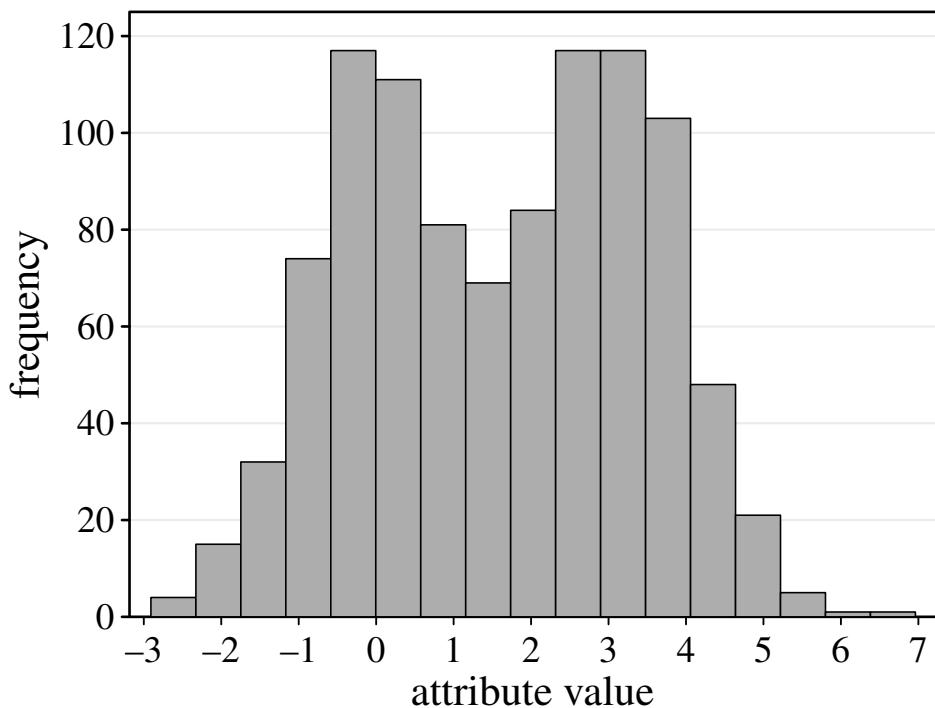
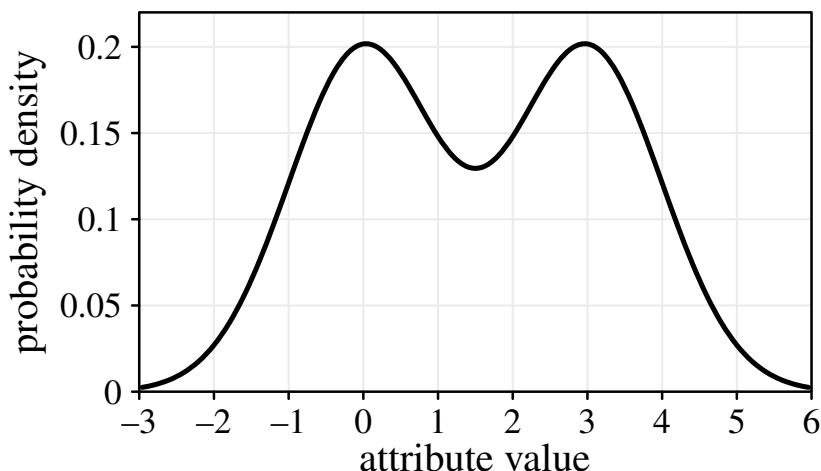
- A histogram with too many bins, for the same data as before.
- As a consequence of the high number of bins, the shape of the distribution is not well discernable.

Histograms: Number of Bins



- Probability density function of a sample distribution, from which the data for the following histograms was sampled (1000 values).
- A histogram with 11 bins, a number computed with **Sturges' Rule**:
$$k = \lceil \log_2(n) + 1 \rceil,$$
where n is the number of data points (here: $n = 1000$).
- Sturges' rule is tailored to data from normal distributions and data sets of moderate size ($n \leq 200$).

Histograms: Number of Bins



- Probability density function of a sample distribution, from which the data for the following histograms was sampled (1000 values).
- A histogram with 17 bins, a number that was computed with:

$$k = \left\lceil \frac{1}{h} (\max_i\{x_i\} - \min_i\{x_i\}) \right\rceil,$$

where h may be chosen as

$$h = 3.5 \cdot s \cdot n^{-\frac{1}{3}}$$

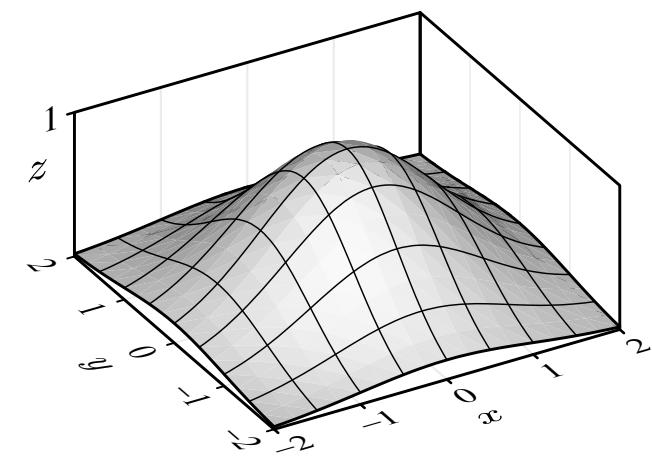
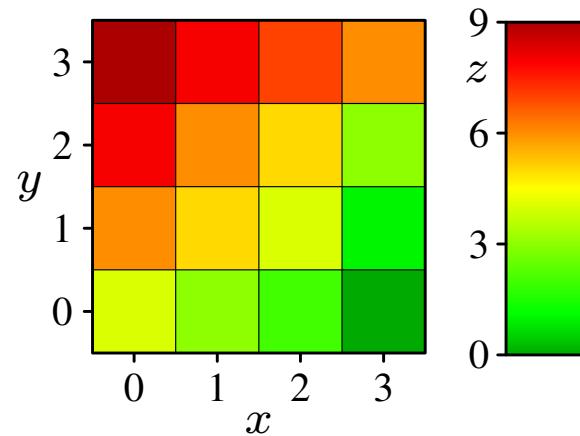
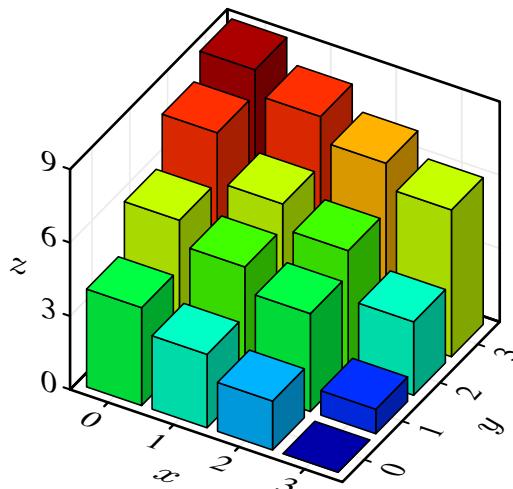
(s : sample standard deviation) or

$$h = 2 \cdot (Q_3 - Q_1) \cdot n^{-\frac{1}{3}}$$

($Q_3 - Q_1$: interquartile range).

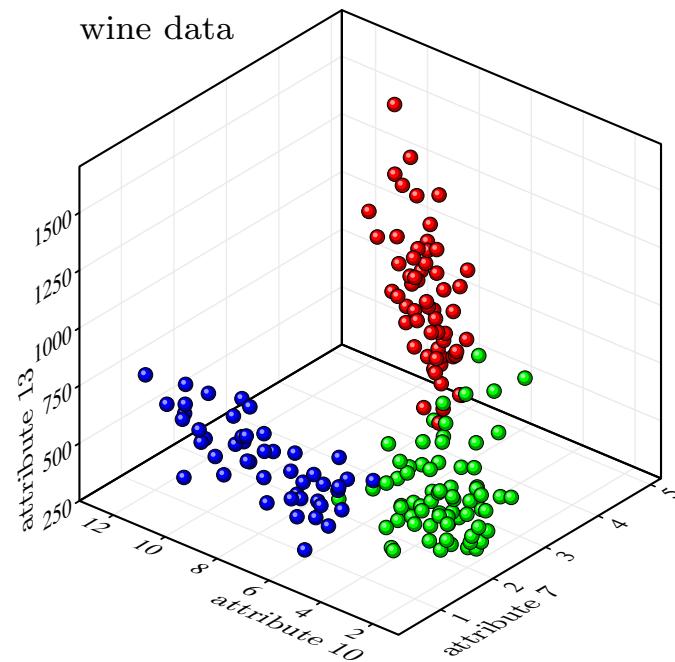
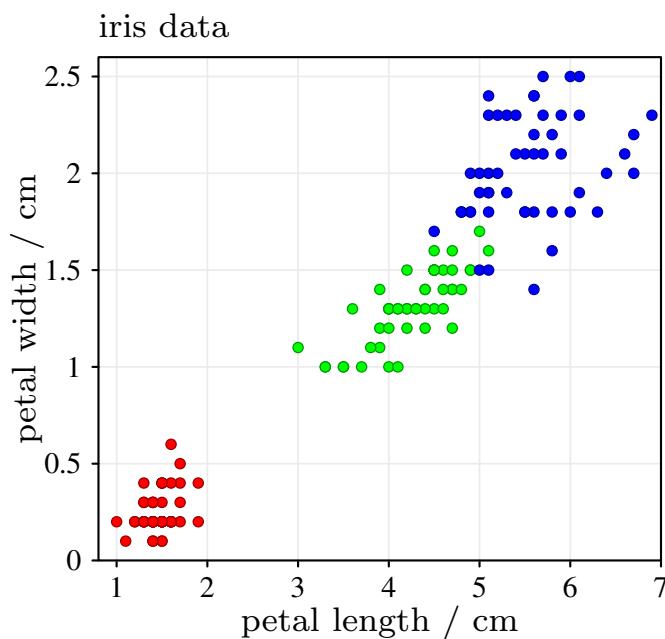
3-Dimensional Diagrams

- 3-dimensional bar charts may be used to display contingency tables (the 3rd dimension represents the value pair frequency).
- The 3rd spacial dimension may be replaced by a color scale. This type of chart is sometimes referred to as a *heatmap*. (In a 3-dimensional bar chart color may also code the z -value (redundantly).)
- Surface plots are 3-dimensional analogs of line charts.



Scatter Plots

- Scatter plots are used to display 2- or 3-dimensional metric data sets.
- Sample values are the coordinates of a point
(that is, numbers are represented by lengths).



- Scatter plots provide simple means to check for dependency.

How to Lie with Statistics

pictures not available in online version

Often the vertical axis of a pole or bar chart does not start at zero, but at some higher value.

In such a case the conveyed impression of the ratio of the depicted values is completely wrong.

This effect is used to brag about increases in turnover, speed etc.

Sources of these diagrams and those on the following transparencies:

Darrell Huff:

How to Lie with Statistics, 1954.

Walter Krämer:

So lügt man mit Statistik, 1991.

How to Lie with Statistics

pictures not available in online version

- Depending on the position of the zero line of a pole, bar, or line chart completely different impressions can be conveyed.

How to Lie with Statistics

pictures not available in online version

- Poles and bars are frequently replaced by (sketches of) objects in order to make the diagram more aesthetically appealing.
- However, objects are perceived as 2- or even 3-dimensional and thus convey a completely different impression of the numerical ratios.

How to Lie with Statistics

pictures not available in online version

How to Lie with Statistics

pictures not available in online version

- In the left diagram the areas of the barrels represent the numerical value. However, since the barrels are drawn 3-dimensional, a wrong impression of the numerical ratios is conveyed.
- The right diagram is particularly striking: an *area measure* is represented by the *side length* of a rectangle representing the apartment.

Good Data Visualization

picture not available in online version

- This is likely the most famous example of good data visualization.
- It easy to understand and conveys information about several quantities, like number of people, location, temperature etc. [Charles Joseph Minard 1869]

Descriptive Statistics:

Characteristic Measures

Descriptive Statistics: Characteristic Measures

Idea: Describe a given sample by few characteristic measures and thus summarize the data.

- **Localization Measures**

Localization measures describe, often by a single number, where the data points of a sample are located in the domain of an attribute.

- **Dispersion Measures**

Dispersion measures describe how much the data points vary around a localization parameter and thus indicate how well this parameter captures the localization of the data.

- **Shape Measures**

Shape measures describe the shape of the distribution of the data points relative to a reference distribution.

The most common reference distribution is the normal distribution (Gaussian).

Localization Measures: Mode and Median

- **Mode** x^*

The mode is the attribute value that is most frequent in the sample.
It need not be unique, because several values can have the same frequency.
It is the most general measure, because it is applicable for all scale types.

- **Median** \tilde{x}

The median minimizes the sum of absolute differences:

$$\sum_{i=1}^n |x_i - \tilde{x}| = \min . \quad \text{and thus it is} \quad \sum_{i=1}^n \operatorname{sgn}(x_i - \tilde{x}) = 0$$

If $x = (x_{(1)}, \dots, x_{(n)})$ is a sorted data set, the median is defined as

$$\tilde{x} = \begin{cases} x_{(\frac{n+1}{2})}, & \text{if } n \text{ is odd,} \\ \frac{1}{2} \left(x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)} \right), & \text{if } n \text{ is even.} \end{cases}$$

The median is applicable to ordinal and metric attributes.

(For non-metric attributes either $x_{(\frac{n}{2})}$ or $x_{(\frac{n}{2}+1)}$ needs to be chosen for even n .)

Localization Measures: Arithmetic Mean

- **Arithmetic Mean** \bar{x}

The arithmetic mean minimizes the sum of squared differences:

$$\sum_{i=1}^n (x_i - \bar{x})^2 = \min . \quad \text{and thus it is} \quad \sum_{i=1}^n (x_i - \bar{x}) = \sum_{i=1}^n x_i - n\bar{x} = 0$$

The arithmetic mean is defined as

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i.$$

The arithmetic mean is only applicable to metric attributes.

- Even though the arithmetic mean is the most common localization measure, the **median** is preferable if
 - there are few sample cases,
 - the distribution is asymmetric, and/or
 - one expects that outliers are present.

How to Lie with Statistics

pictures not available in online version

Dispersion Measures: Range and Interquantile Range

A man with his head in the freezer and feet in the oven
is *on the average* quite comfortable.

old statistics joke

- **Range R**

The range of a data set is the difference between the maximum and the minimum value.

$$R = x_{\max} - x_{\min} = \max_{i=1}^n x_i - \min_{i=1}^n x_i$$

- **Interquantile Range**

The p -quantile of a data set is a value such that a fraction of p of all sample values are smaller than this value. (The median is the $\frac{1}{2}$ -quantile.)

The p -interquantile range, $0 < p < \frac{1}{2}$, is the difference between the $(1 - p)$ -quantile and the p -quantile.

The most common is the *interquartile range* ($p = \frac{1}{4}$).

Dispersion Measures: Average Absolute Deviation

- **Average Absolute Deviation**

The average absolute deviation is the average of the absolute deviations of the sample values from the median or the arithmetic mean.

- Average Absolute Deviation from the **Median**

$$d_{\tilde{x}} = \frac{1}{n} \sum_{i=1}^n |x_i - \tilde{x}|$$

- Average Absolute Deviation from the **Arithmetic Mean**

$$d_{\bar{x}} = \frac{1}{n} \sum_{i=1}^n |x_i - \bar{x}|$$

- It is always $d_{\tilde{x}} \leq d_{\bar{x}}$, since the median minimizes the sum of absolute deviations (see the definition of the median).

Dispersion Measures: Variance and Standard Deviation

- **Variance s^2**

It would be natural to define the variance as the average squared deviation:

$$v^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2.$$

However, inductive statistics suggests that it is better defined as

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2.$$

- **Standard Deviation s**

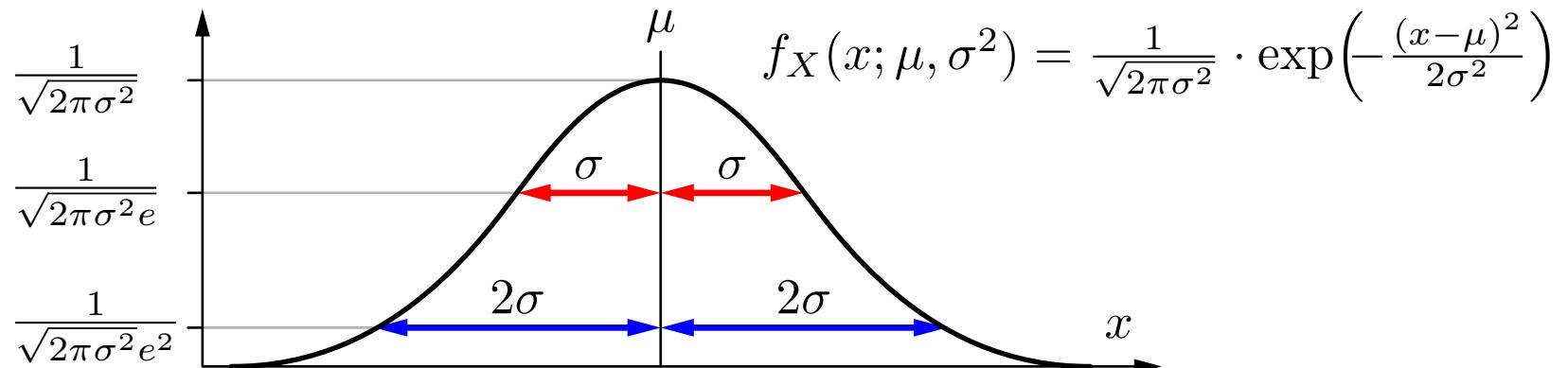
The standard deviation is the square root of the variance, i.e.,

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}.$$

Dispersion Measures: Variance and Standard Deviation

- **Special Case: Normal/Gaussian Distribution**

The variance/standard deviation provides information about the height of the mode and the width of the curve.



- μ : expected value, estimated by mean value \bar{x}
 - σ^2 : variance, estimated by (empirical) variance s^2
 - σ : standard deviation, estimated by (empirical) standard deviation s
- (Details about parameter estimation are studied later.)

Dispersion Measures: Variance and Standard Deviation

Note that it is often more convenient to compute the variance using the formula that results from the following transformation:

$$\begin{aligned}s^2 &= \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i^2 - 2x_i\bar{x} + \bar{x}^2) \\&= \frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 - 2\bar{x} \sum_{i=1}^n x_i + \sum_{i=1}^n \bar{x}^2 \right) \\&= \frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 - 2n\bar{x}^2 + n\bar{x}^2 \right) = \frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 - n\bar{x}^2 \right) \\&= \frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2 \right)\end{aligned}$$

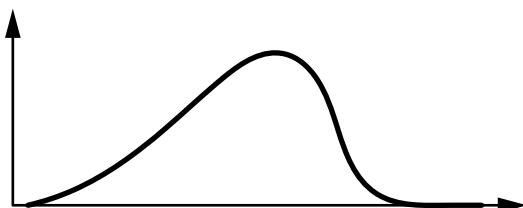
- Advantage: The sums $\sum_{i=1}^n x_i$ and $\sum_{i=1}^n x_i^2$ can both be computed in the same traversal of the data and from them both mean and variance are computable.

Shape Measures: Skewness

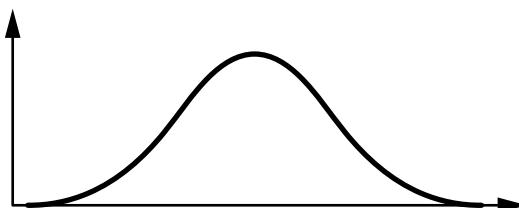
- The **skewness** α_3 (or **skew** for short) measures whether, and if, how much, a distribution differs from a symmetric distribution.
- It is computed from the 3rd moment about the mean, which explains the index 3.

$$\alpha_3 = \frac{1}{n \cdot v^3} \sum_{i=1}^n (x_i - \bar{x})^3 = \frac{1}{n} \sum_{i=1}^n z_i^3$$

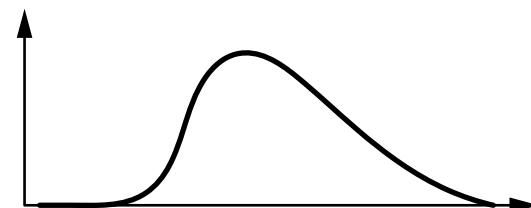
where $z_i = \frac{x_i - \bar{x}}{v}$ and $v^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$.



$\alpha_3 < 0$: right steep



$\alpha_3 = 0$: symmetric



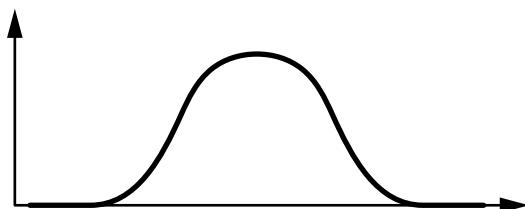
$\alpha_3 > 0$: left steep

Shape Measures: Kurtosis

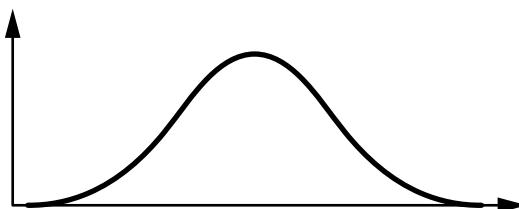
- The **kurtosis** or **excess** α_4 measures how much a distribution is arched, usually compared to a Gaussian distribution.
- It is computed from the 4th moment about the mean, which explains the index 4.

$$\alpha_4 = \frac{1}{n \cdot v^4} \sum_{i=1}^n (x_i - \bar{x})^4 = \frac{1}{n} \sum_{i=1}^n z_i^4$$

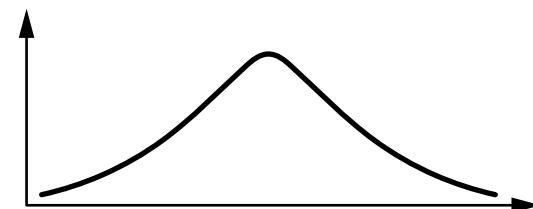
$$\text{where } z_i = \frac{x_i - \bar{x}}{v} \quad \text{and} \quad v^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2.$$



$\alpha_4 < 3$: leptokurtic



$\alpha_4 = 3$: Gaussian



$\alpha_4 > 3$: platikurtic

Moments of Data Sets

- The k -th **moment** of a dataset is defined as

$$m'_k = \frac{1}{n} \sum_{i=1}^n x_i^k.$$

The first moment is the **mean** $m'_1 = \bar{x}$ of the data set.

Using the moments of a data set the **variance** s^2 can also be written as

$$s^2 = \frac{1}{n-1} \left(m'_2 - \frac{1}{n} m'^2_1 \right) \quad \text{and also} \quad v^2 = \frac{1}{n} m'_2 - \frac{1}{n^2} m'^2_1.$$

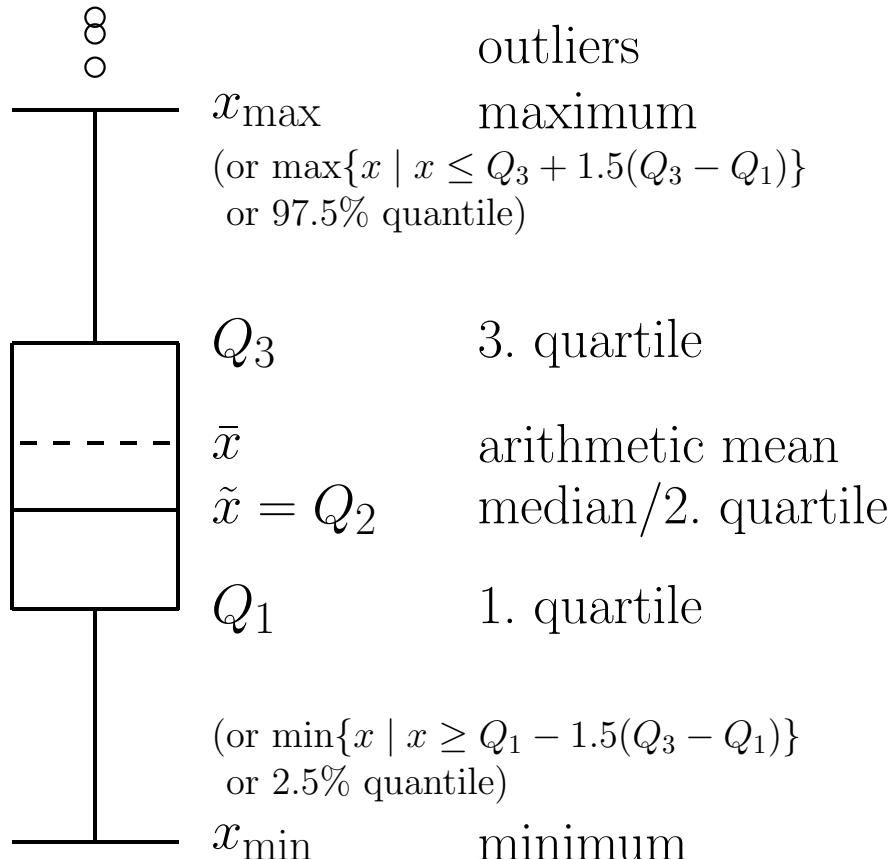
- The k -th **moment about the mean** is defined as

$$m_k = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^k.$$

It is $m_1 = 0$ and $m_2 = v^2$ (i.e., the **average squared deviation**).

The **skewness** is $\alpha_3 = \frac{m_3}{m_2^{3/2}}$ and the **kurtosis** is $\alpha_4 = \frac{m_4}{m_2^2}$.

Visualizing Characteristic Measures: Box Plots



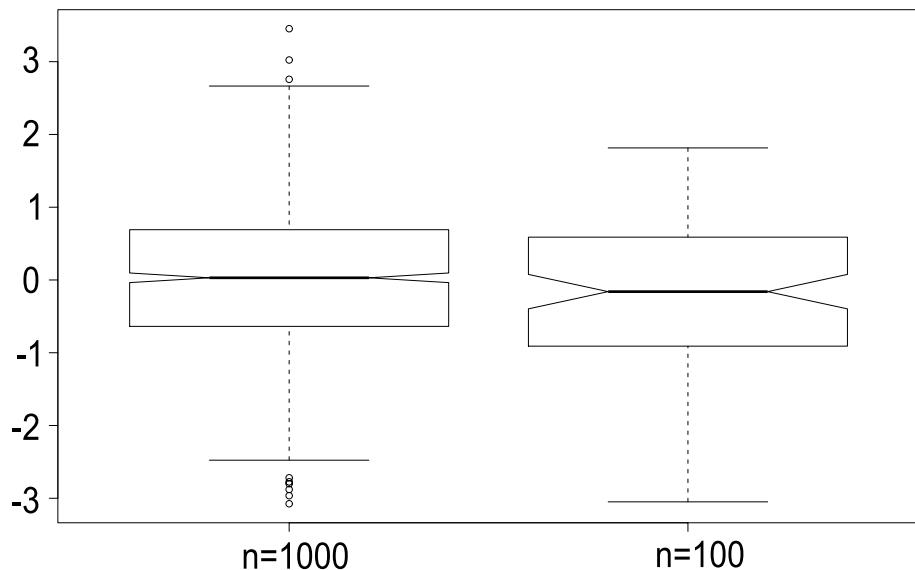
A box plot is a common way to combine some important characteristic measures into a single graphical representation.

Often the central box is drawn constricted () w.r.t. the arithmetic mean in order to emphasize its location.

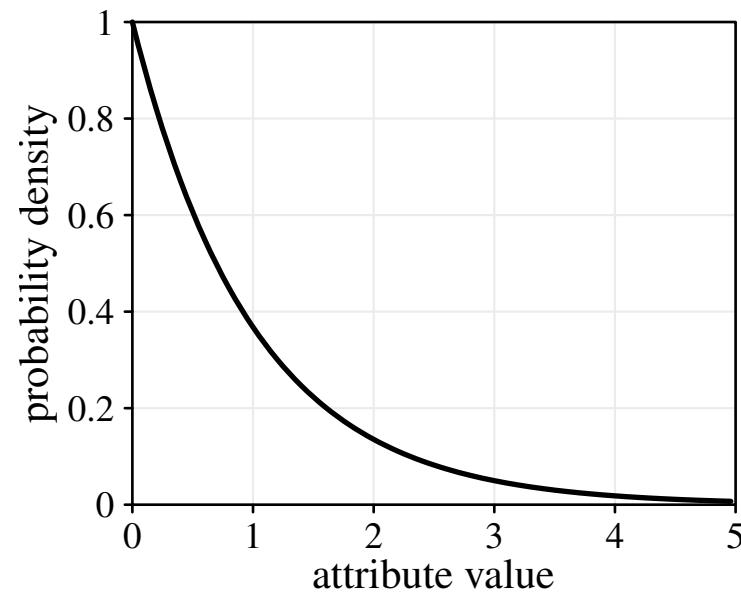
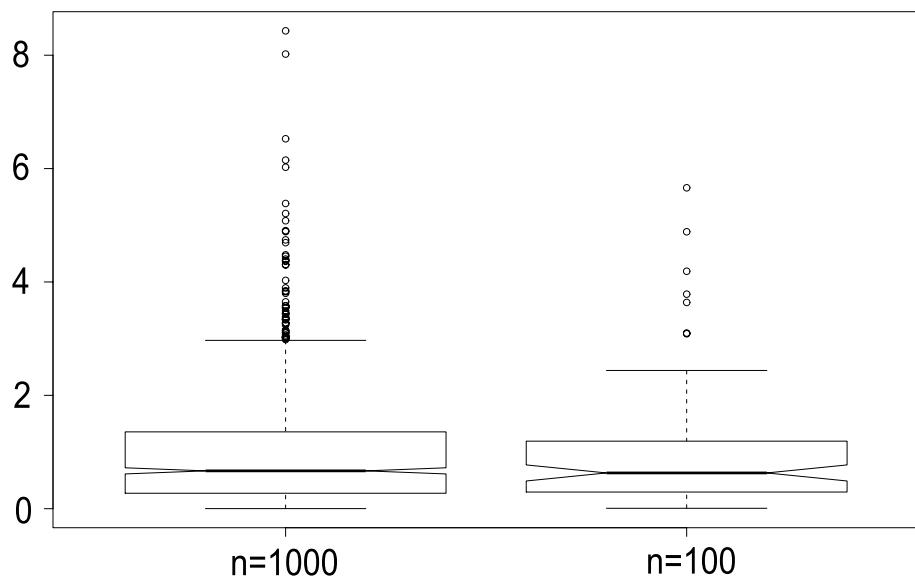
The “whiskers” are often limited in length to $1.5(Q_3 - Q_1)$. Data points beyond these limits are suspected to be outliers.

Box plots are often used to get a quick impression of the distribution of the data by showing them side by side for several attributes or data subsets.

Box Plots: Examples



- left top: two samples from a standard normal distribution.
- left bottom: two samples from an exponential distribution.
- right bottom:
probability density function of the exponential distribution with $\lambda = 1$.



Multidimensional Characteristic Measures

General Idea: Transfer the characteristic measures to vectors.

- **Arithmetic Mean**

The arithmetic mean for multi-dimensional data is the vector mean of the data points. For two dimensions it is

$$\overline{(x, y)} = \frac{1}{n} \sum_{i=1}^n (x_i, y_i) = (\bar{x}, \bar{y})$$

For the arithmetic mean the transition to several dimensions only combines the arithmetic means of the individual dimensions into one vector.

- Other measures are transferred in a similar way.

However, sometimes the transfer leads to new quantities, as for the variance, which requires adaptation due to its quadratic nature.

Excursion: Vector Products

For the variance, the square of the difference to the mean has to be generalized.

Inner Product Scalar Product

$$\vec{v}^\top \vec{v}$$
$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix}$$
$$(v_1, v_2, \dots, v_m) \quad \sum_{i=1}^m v_i^2$$

Outer Product Matrix Product

$$\vec{v}\vec{v}^\top$$
$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix}$$
$$\begin{pmatrix} v_1 & v_2 & \dots & v_m \end{pmatrix}$$
$$\begin{pmatrix} v_1^2 & v_1v_2 & \dots & v_1v_m \\ v_1v_2 & v_2^2 & \dots & v_2v_m \\ \vdots & \ddots & \ddots & \vdots \\ v_1v_m & v_2v_m & \dots & v_m^2 \end{pmatrix}$$

- In principle both vector products may be used for a generalization.
- The second, however, yields more information about the distribution:
 - a measure of the (linear) dependence of the attributes,
 - a description of the direction dependence of the dispersion.

Covariance Matrix

- **Covariance Matrix**

Compute variance formula with vectors (square: outer product $\vec{v}\vec{v}^\top$):

$$\mathbf{S} = \frac{1}{n-1} \sum_{i=1}^n \left(\begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} \right) \left(\begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} \right)^\top = \begin{pmatrix} s_x^2 & s_{xy} \\ s_{xy} & s_y^2 \end{pmatrix}$$

where

$$s_x^2 = s_{xx} = \frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 - n\bar{x}^2 \right) \quad (\text{variance of } x)$$

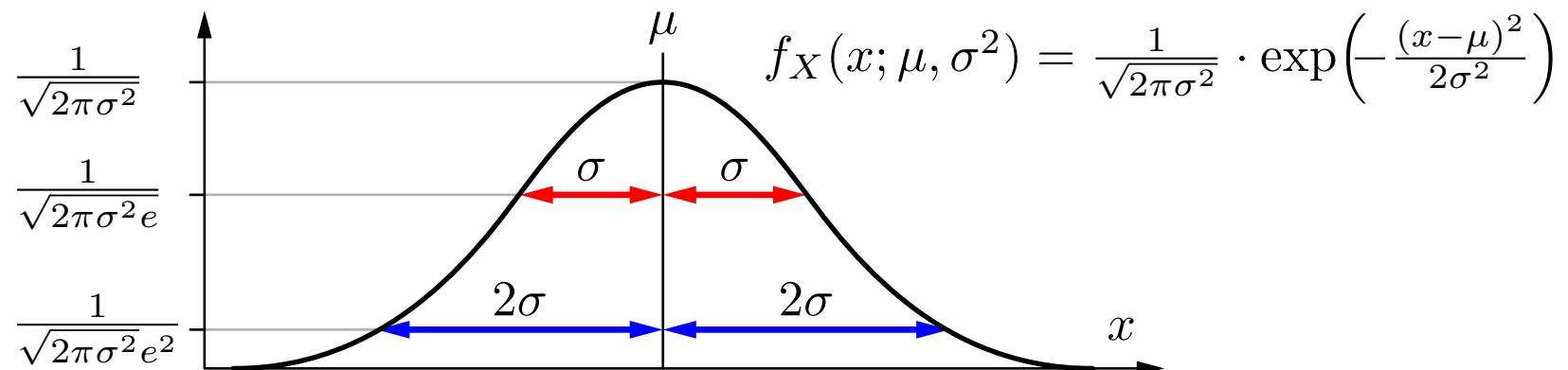
$$s_y^2 = s_{yy} = \frac{1}{n-1} \left(\sum_{i=1}^n y_i^2 - n\bar{y}^2 \right) \quad (\text{variance of } y)$$

$$s_{xy} = \frac{1}{n-1} \left(\sum_{i=1}^n x_i y_i - n\bar{x}\bar{y} \right) \quad (\text{covariance of } x \text{ and } y)$$

Reminder: Variance and Standard Deviation

- **Special Case: Normal/Gaussian Distribution**

The variance/standard deviation provides information about the height of the mode and the width of the curve.



- μ : expected value, estimated by mean value \bar{x} ,
 σ^2 : variance, estimated by (empirical) variance s^2 ,
 σ : standard deviation, estimated by (empirical) standard deviation s .
Important: the standard deviation has the same unit as the expected value.

Multivariate Normal Distribution

- A **univariate normal distribution** has the density function

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

μ : expected value, estimated by mean value \bar{x} ,
 σ^2 : variance, estimated by (empirical) variance s^2 ,
 σ : standard deviation, estimated by (empirical) standard deviation s .

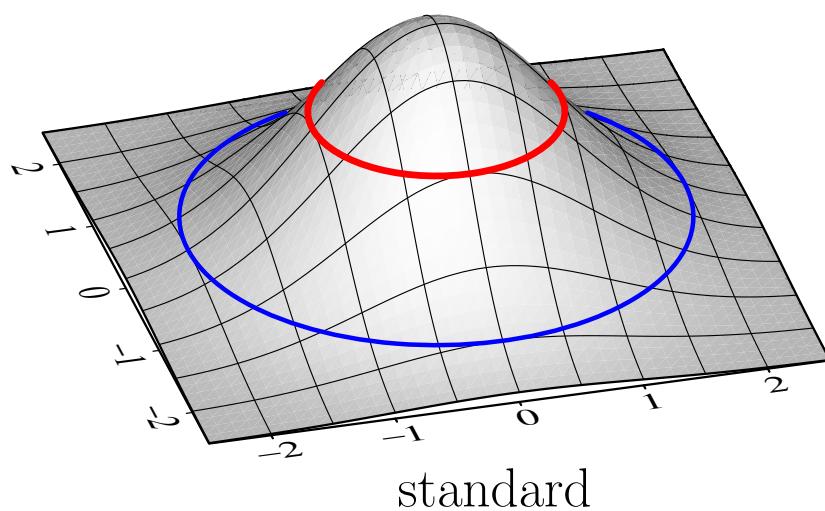
- A **multivariate normal distribution** has the density function

$$f_{\vec{X}}(\vec{x}; \vec{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^m |\Sigma|}} \cdot \exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu})^\top \Sigma^{-1}(\vec{x} - \vec{\mu})\right)$$

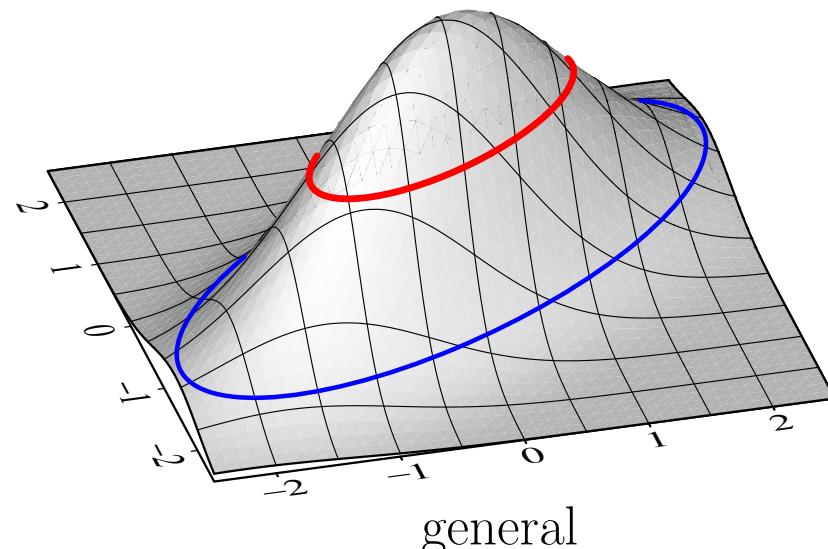
m : size of the vector \vec{x} (it is m -dimensional),
 $\vec{\mu}$: mean value vector, estimated by (empirical) mean value vector $\bar{\vec{x}}$,
 Σ : covariance matrix, estimated by (empirical) covariance matrix \mathbf{S} ,
 $|\Sigma|$: determinant of the covariance matrix Σ .

Interpretation of a Covariance Matrix

- The variance/standard deviation relates the spread of the distribution to the spread of a **standard normal distribution** ($\sigma^2 = \sigma = 1$).
- The covariance matrix relates the spread of the distribution to the spread of a **multivariate standard normal distribution** ($\Sigma = 1$).
- Example: bivariate normal distribution



- **Question:** Is there a multivariate analog of standard deviation?



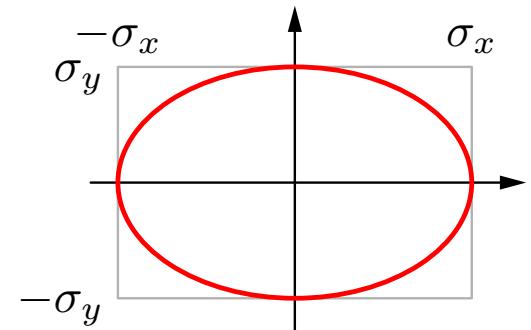
Interpretation of a Covariance Matrix

Question: Is there a multivariate analog of standard deviation?

First insight:

If all covariances vanish,
the contour lines are axes-parallel ellipses.
The upper ellipse is inscribed into the
rectangle $[-\sigma_x, \sigma_x] \times [-\sigma_y, \sigma_y]$.

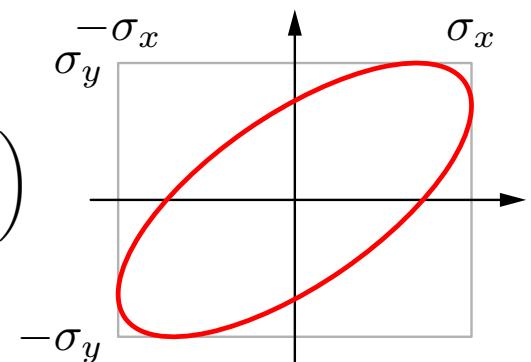
$$\Sigma = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}$$



Second insight:

If the covariances do not vanish,
the contour lines are rotated ellipses.
Still the upper ellipse is inscribed into the
rectangle $[-\sigma_x, \sigma_x] \times [-\sigma_y, \sigma_y]$.

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$$



Consequence: A covariance matrix describes a scaling and a rotation.

Cholesky Decomposition

- Intuitively: **Compute an analog of standard deviation.**
- Let \mathbf{S} be a symmetric, positive definite matrix (e.g. a covariance matrix). Cholesky decomposition serves the purpose to compute a “square root” of \mathbf{S} .
 - symmetric: $\forall 1 \leq i, j \leq m : s_{ij} = s_{ji}$
 - positive definite: for all m -dimensional vectors $\vec{v} \neq \vec{0}$ it is $\vec{v}^\top \mathbf{S} \vec{v} > 0$
- Formally: Compute a left/lower triangular matrix \mathbf{L} such that $\mathbf{L}\mathbf{L}^\top = \mathbf{S}$. (\mathbf{L}^\top is the transpose of the matrix \mathbf{L} .)

$$\begin{aligned}l_{ii} &= \left(s_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \right)^{\frac{1}{2}} \\l_{ji} &= \frac{1}{l_{ii}} \left(s_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk} \right), \quad j = i+1, i+2, \dots, m.\end{aligned}$$

Cholesky Decomposition

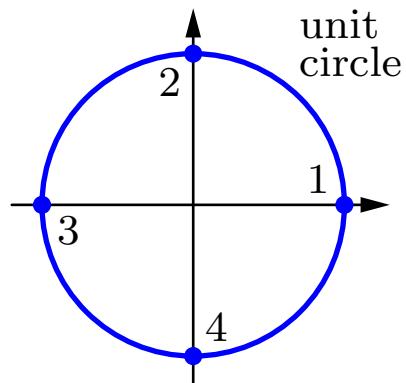
Special Case: Two Dimensions

- Covariance matrix

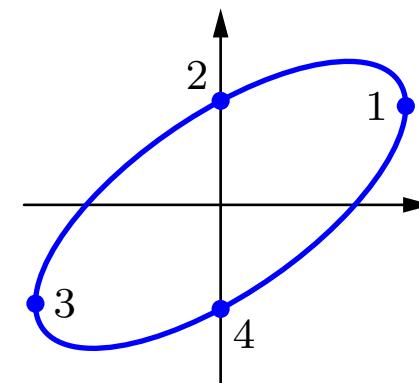
$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$$

- Cholesky decomposition

$$L = \begin{pmatrix} \sigma_x & 0 \\ \frac{\sigma_{xy}}{\sigma_x} & \frac{1}{\sigma_x} \sqrt{\sigma_x^2 \sigma_y^2 - \sigma_{xy}^2} \end{pmatrix}$$



mapping with L
 $\vec{v}' = L\vec{v}$



Eigenvalue Decomposition

- Eigenvalue decomposition also yields an **analog of standard deviation**.
- It is computationally more expensive than Cholesky decomposition.
- Let \mathbf{S} be a symmetric, positive definite matrix (e.g. a covariance matrix).
 - \mathbf{S} can be written as

$$\mathbf{S} = \mathbf{R} \operatorname{diag}(\lambda_1, \dots, \lambda_m) \mathbf{R}^{-1},$$

where the λ_j , $j = 1, \dots, m$, are the eigenvalues of \mathbf{S} and the columns of \mathbf{R} are the (normalized) eigenvectors of \mathbf{S} .

- The eigenvalues λ_j , $j = 1, \dots, m$, of \mathbf{S} are all positive and the eigenvectors of \mathbf{S} are orthonormal ($\rightarrow \mathbf{R}^{-1} = \mathbf{R}^\top$).
- Due to the above, \mathbf{S} can be written as $\mathbf{S} = \mathbf{T} \mathbf{T}^\top$, where

$$\mathbf{T} = \mathbf{R} \operatorname{diag} \left(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_m} \right)$$

Eigenvalue Decomposition

Special Case: Two Dimensions

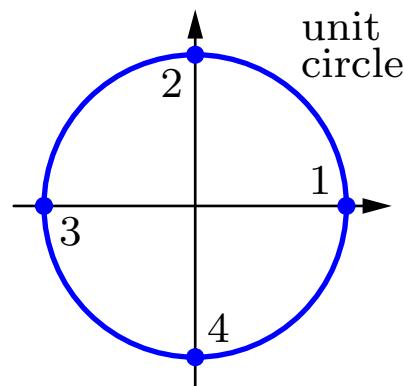
- Covariance matrix

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$$

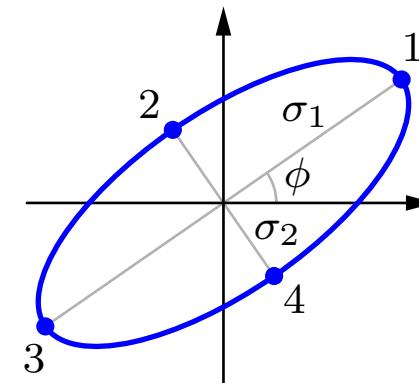
- Eigenvalue decomposition

$$\mathbf{T} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix},$$

$$s = \sin \phi, c = \cos \phi, \phi = \frac{1}{2} \arctan \frac{2\sigma_{xy}}{\sigma_x^2 - \sigma_y^2},$$
$$\sigma_1 = \sqrt{c^2\sigma_x^2 + s^2\sigma_y^2 + 2sc\sigma_{xy}},$$
$$\sigma_2 = \sqrt{s^2\sigma_x^2 + c^2\sigma_y^2 - 2sc\sigma_{xy}}.$$



mapping with $\vec{v}' = \mathbf{T}\vec{v}$



Eigenvalue Decomposition

Eigenvalue decomposition enables us to write a covariance matrix Σ as

$$\Sigma = \mathbf{T}\mathbf{T}^\top \quad \text{with} \quad \mathbf{T} = \mathbf{R} \operatorname{diag} \left(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_m} \right).$$

As a consequence we can write its inverse Σ^{-1} as

$$\Sigma^{-1} = \mathbf{U}^\top \mathbf{U} \quad \text{with} \quad \mathbf{U} = \operatorname{diag} \left(\lambda_1^{-\frac{1}{2}}, \dots, \lambda_m^{-\frac{1}{2}} \right) \mathbf{R}^\top.$$

\mathbf{U} describes the inverse mapping of \mathbf{T} , i.e., rotates the ellipse so that its axes coincide with the coordinate axes and then scales the axes to unit length. Hence:

$$(\vec{x} - \vec{y})^\top \Sigma^{-1} (\vec{x} - \vec{y}) = (\vec{x} - \vec{y})^\top \mathbf{U}^\top \mathbf{U} (\vec{x} - \vec{y}) = (\vec{x}' - \vec{y}')^\top (\vec{x}' - \vec{y}'),$$

where $\vec{x}' = \mathbf{U}\vec{x}$ and $\vec{y}' = \mathbf{U}\vec{y}$.

Result: $(\vec{x} - \vec{y})^\top \Sigma^{-1} (\vec{x} - \vec{y})$ is equivalent to the squared **Euclidean distance** in the properly scaled eigensystem of the covariance matrix Σ .

$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^\top \Sigma^{-1} (\vec{x} - \vec{y})}$ is called **Mahalanobis distance**.

Eigenvalue Decomposition

Eigenvalue decomposition also shows that the determinant of the covariance matrix Σ provides a measure of the (hyper-)volume of the (hyper-)ellipsoid. It is

$$|\Sigma| = |\mathbf{R}| \left| \text{diag}(\lambda_1, \dots, \lambda_m) \right| |\mathbf{R}^\top| = \left| \text{diag}(\lambda_1, \dots, \lambda_m) \right| = \prod_{i=1}^m \lambda_i,$$

since $|\mathbf{R}| = |\mathbf{R}^\top| = 1$ as \mathbf{R} is orthogonal with unit length columns, and thus

$$\sqrt{|\Sigma|} = \prod_{i=1}^m \sqrt{\lambda_i},$$

which is proportional to the (hyper-)volume of the (hyper-)ellipsoid.

To be precise, the volume of the m -dimensional (hyper-)ellipsoid a (hyper-)sphere with radius r is mapped to with a covariance matrix Σ is

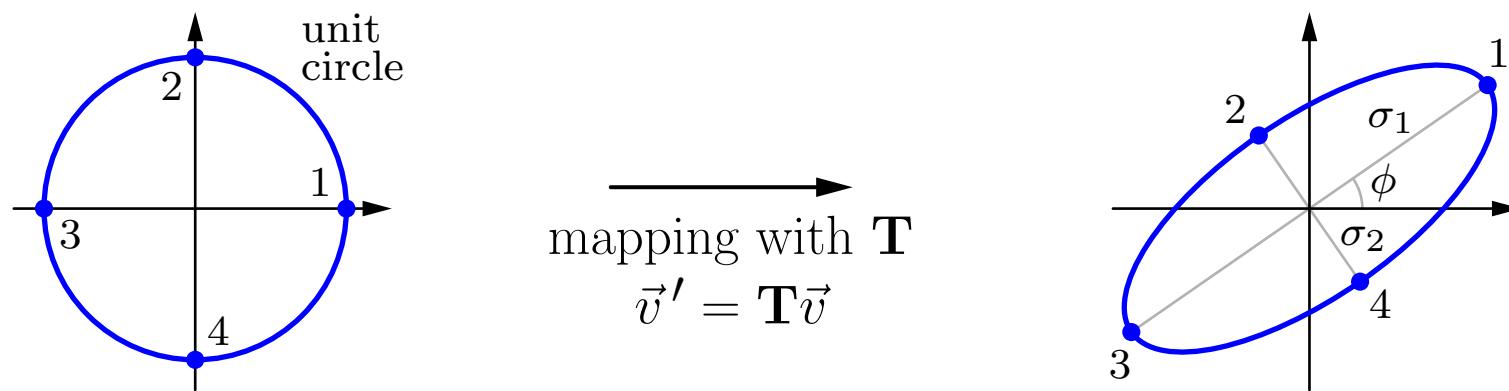
$$V_m(r) = \frac{\pi^{\frac{m}{2}} r^m}{\Gamma\left(\frac{m}{2} + 1\right)} \sqrt{|\Sigma|}, \quad \text{where} \quad \begin{aligned} \Gamma(x) &= \int_0^\infty e^{-t} t^{x-1} dt, & x > 0, \\ \Gamma(x+1) &= x \cdot \Gamma(x), & \Gamma(\frac{1}{2}) = \sqrt{\pi}, & \Gamma(1) = 1. \end{aligned}$$

Eigenvalue Decomposition

Special Case: Two Dimensions

- Covariance matrix and its eigenvalue decomposition:

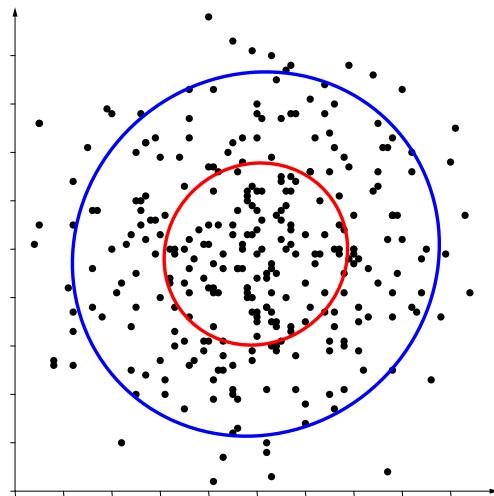
$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix} \quad \text{and} \quad \mathbf{T} = \begin{pmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}.$$



- The area of the ellipse, to which the unit circle (area π) is mapped, is

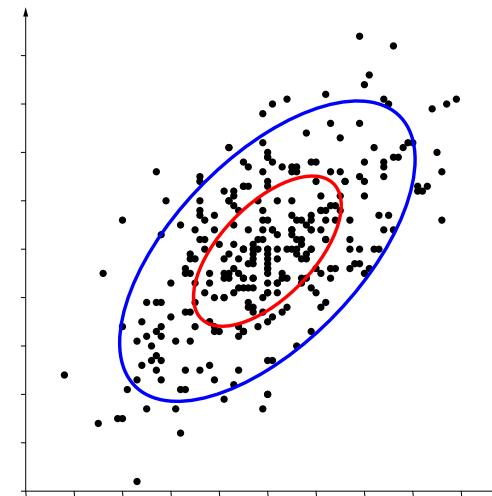
$$A = \pi\sigma_1\sigma_2 = \pi\sqrt{|\Sigma|}.$$

Covariance Matrices of Example Data Sets



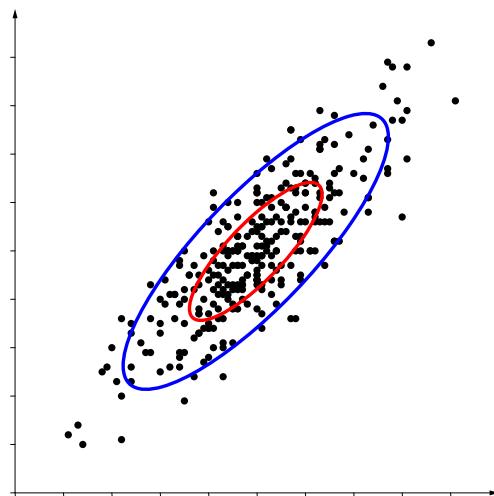
$$\Sigma \approx \begin{pmatrix} 3.59 & 0.19 \\ 0.19 & 3.54 \end{pmatrix}$$

$$L \approx \begin{pmatrix} 1.90 & 0 \\ 0.10 & 1.88 \end{pmatrix}$$



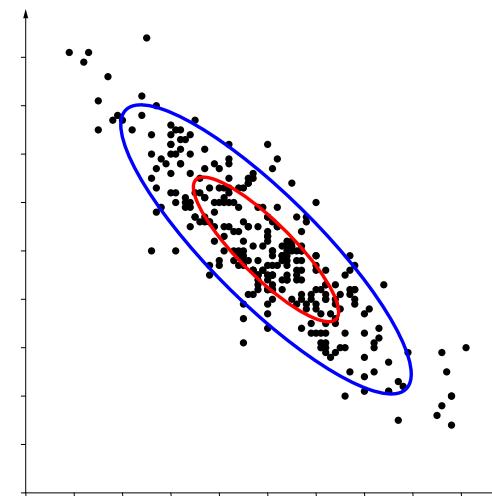
$$\Sigma \approx \begin{pmatrix} 2.33 & 1.44 \\ 1.44 & 2.41 \end{pmatrix}$$

$$L \approx \begin{pmatrix} 1.52 & 0 \\ 0.95 & 1.22 \end{pmatrix}$$



$$\Sigma \approx \begin{pmatrix} 1.88 & 1.62 \\ 1.62 & 2.03 \end{pmatrix}$$

$$L \approx \begin{pmatrix} 1.37 & 0 \\ 1.18 & 0.80 \end{pmatrix}$$



$$\Sigma \approx \begin{pmatrix} 2.25 & -1.93 \\ -1.93 & 2.23 \end{pmatrix}$$

$$L \approx \begin{pmatrix} 1.50 & 0 \\ -1.29 & 0.76 \end{pmatrix}$$

Covariance Matrix: Summary

- A covariance matrix provides information about the **height of the mode** and about the **spread/dispersion** of a multivariate normal distribution (or of a set of data points that are roughly normally distributed).
- A multivariate **analog of standard deviation** can be computed with Cholesky decomposition and eigenvalue decomposition.
The resulting matrix describes the distribution's shape and orientation.
- The shape and the orientation of a two-dimensional normal distribution can be visualized as an **ellipse** (curve of equal probability density; similar to a **contour line** — line of equal height — on a map.)
- The shape and the orientation of a three-dimensional normal distribution can be visualized as an **ellipsoid** (surface of equal probability density).
- The (square root of the) **determinant** of a covariance matrix describes the spread of a multivariate normal distribution with a single value.
It is a measure of the area or (hyper-)volume of the (hyper-)ellipsoid.

Correlation and Principal Component Analysis

Correlation Coefficient

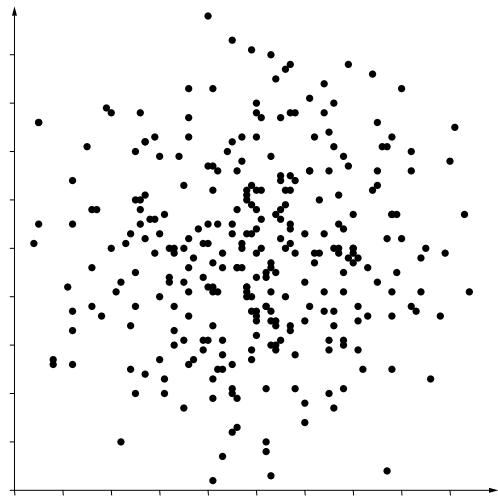
- The covariance is a measure of the strength of **linear dependence** of the two quantities of which it is computed.
- However, its value depends on the variances of the individual dimensions.
⇒ Normalize to unit variance in the individual dimensions.
- **Correlation Coefficient**
(more precisely: Pearson's Product Moment Correlation Coefficient)

$$r = \frac{s_{xy}}{s_x s_y}, \quad r \in [-1, +1].$$

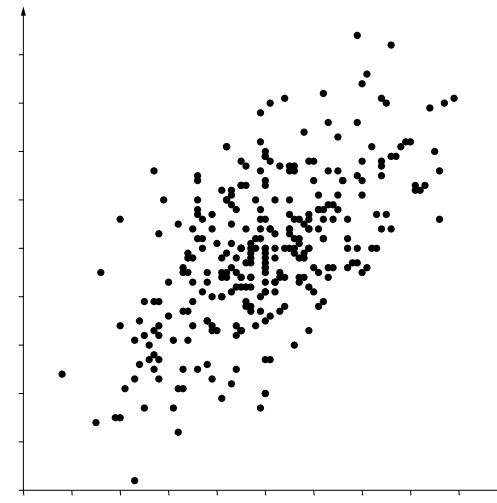
- r measures the strength of linear dependence:
 $r = -1$: the data points lie perfectly on a descending straight line.
 $r = +1$: the data points lie perfectly on an ascending straight line.
- $r = 0$: there is no **linear** dependence between the two attributes
(but there may be a non-linear dependence!).

Correlation Coefficients of Example Data Sets

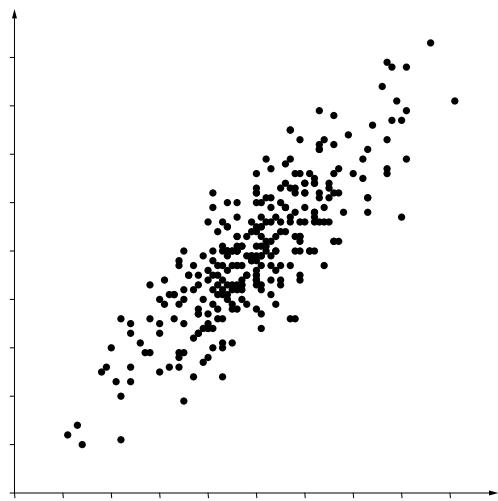
no
correlation
 $(r \approx 0.05)$



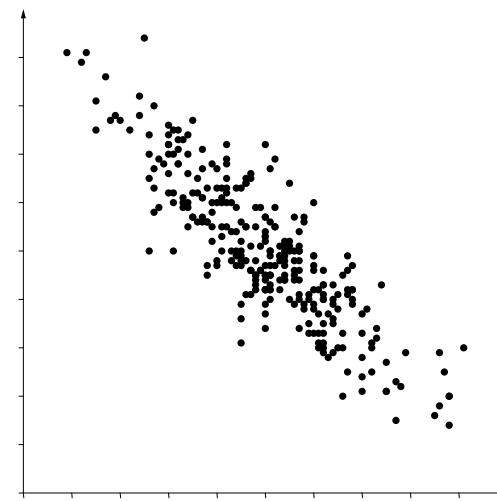
weak
positive
correlation
 $(r \approx 0.61)$



strong
positive
correlation
 $(r \approx 0.83)$



strong
negative
correlation
 $(r \approx -0.86)$



Correlation Matrix

- **Normalize Data** (z -score normalization)

Transform data to mean value 0 and variance/standard deviation 1:

$$\forall i; 1 \leq i \leq n : \quad x'_i = \frac{x_i - \bar{x}}{s_x}, \quad y'_i = \frac{y_i - \bar{y}}{s_y}.$$

- **Compute Covariance Matrix of Normalized Data**

Sum outer products of transformed data vectors:

$$\Sigma' = \frac{1}{n-1} \sum_{i=1}^n \begin{pmatrix} x'_i \\ y'_i \end{pmatrix} \begin{pmatrix} x'_i \\ y'_i \end{pmatrix}^\top = \begin{pmatrix} 1 & r \\ r & 1 \end{pmatrix}$$

Subtraction of mean vector is not necessary (because it is $(0, 0)^\top$).

Diagonal elements are always 1 (because of unit variance in each dimension).

- Normalizing the data and then computing the covariances or computing the covariances and then normalizing them has the same effect.

Correlation Matrix: Interpretation

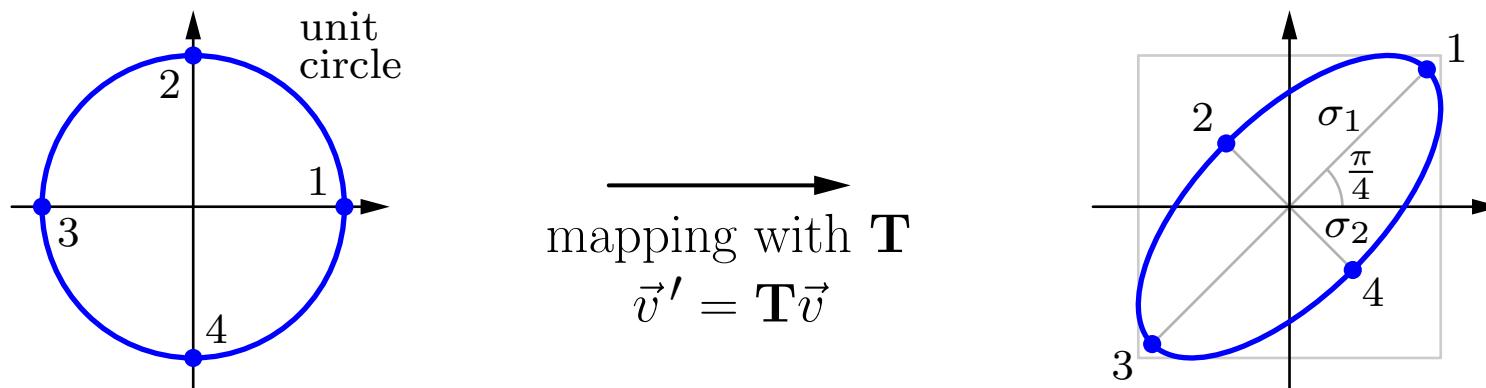
Special Case: Two Dimensions

- Correlation matrix

$$\Sigma' = \begin{pmatrix} 1 & r \\ r & 1 \end{pmatrix}, \quad \text{eigenvalues: } \sigma_1^2, \sigma_2^2$$
$$\text{correlation: } r = \frac{\sigma_1^2 - \sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

- Eigenvalue decomposition

$$\mathbf{T} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}, \quad s = \sin \frac{\pi}{4} = \frac{1}{\sqrt{2}}, \quad \sigma_1 = \sqrt{1+r},$$
$$c = \cos \frac{\pi}{4} = \frac{1}{\sqrt{2}}, \quad \sigma_2 = \sqrt{1-r}.$$



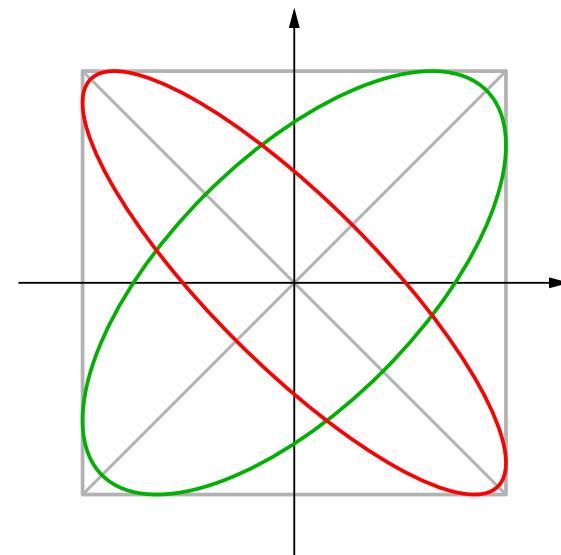
Correlation Matrix: Interpretation

- For two dimensions the eigenvectors of a correlation matrix are always

$$\vec{v}_1 = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right) \quad \text{and} \quad \vec{v}_2 = \left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right)$$

(or their opposites $-\vec{v}_1$ or $-\vec{v}_2$ or exchanged).

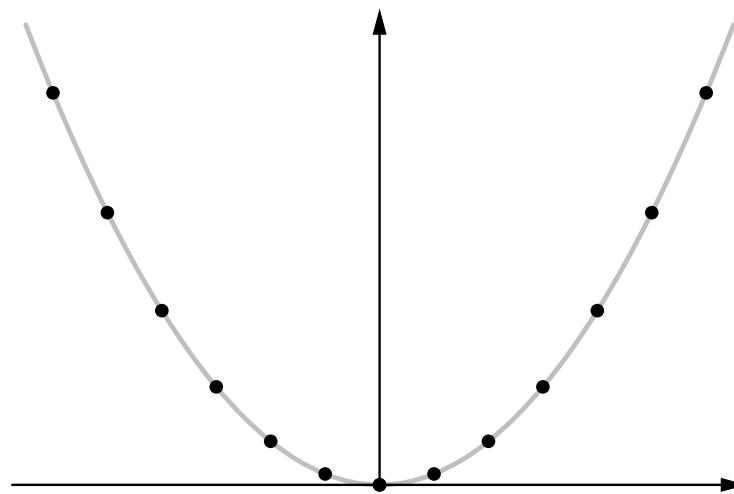
The reason is that the normalization transforms the data points in such a way that the ellipse, the unit circle is mapped to by the “square root” of the covariance matrix of the normalized data, is always inscribed into the square $[-1, 1] \times [-1, 1]$. Hence the ellipse’s major axes are the square’s diagonals.



- The situation is analogous in m -dimensional spaces:
the eigenvectors are always m of the 2^{m-1} diagonals
of the m -dimensional unit (hyper-)cube around the origin.

Correlation and Stochastic (In)Dependence

- Note: stochastic independence $\Rightarrow r = 0$,
but: $r = 0 \not\Rightarrow$ stochastic independence.
- Example: Suppose the data points lie symmetrically on a parabola.



- The correlation coefficient of this data set is $r = 0$,
because there is **no linear** dependence between the two attributes.
However, there is a perfect **quadratic** dependence,
and thus the two attributes are **not** stochastically independent.

Attention: Correlation $\not\Rightarrow$ Causation!

pictures not available in online version

- Always remember:
An observed correlation may be purely coincidental!
- This is especially the case if the data come from processes that show relatively steady growth or decline (these are always correlated).
- In order to claim a causal connection between quantities, the actual mechanism needs to be discovered and confirmed!

Attention: Correlation $\not\Rightarrow$ Causation!

Does high fat intake cause breast cancer?

picture not available in online version

- Data shows a clear correlation between breast cancer death rates and fat intake.
- Is this evidence for a causal connection? If at all, it is quite weak.
- Amount of fat in diet and amount of sugar are correlated.
- Plot of amount of sugar in diet and *colon* cancer death rates would look similar.
- How rich a country is influences the amount of fat and sugar in the diet, but also a lot of other factors (e.g. life expectancy).

Attention: Correlation $\not\Rightarrow$ Causation!

pictures not available in online version

Attention: Correlation $\not\Rightarrow$ Causation!

pictures not available in online version

Attention: Correlation $\not\Rightarrow$ Causation!

pictures not available in online version

Attention: Correlation $\not\Rightarrow$ Causation!

pictures not available in online version

Attention: Correlation $\not\Rightarrow$ Causation!

pictures not available in online version

Attention: Correlation $\not\Rightarrow$ Causation!

pictures not available in online version

Regression Line

- Since the covariance/correlation measures linear dependence, it is not surprising that it can be used to define a **regression line**:

$$(y - \bar{y}) = \frac{s_{xy}}{s_x^2}(x - \bar{x}) \quad \text{or} \quad y = \frac{s_{xy}}{s_x^2}(x - \bar{x}) + \bar{y}.$$

- The regression line can be seen as a conditional arithmetic mean: there is one arithmetic mean for the y -dimensions for each x -value.
- This interpretation is supported by the fact that the regression line minimizes the sum of squared differences in y -direction.
(Reminder: the arithmetic mean minimizes the sum of squared differences.)
- More information on **regression** and the **method of least squares** in the corresponding chapter (to be discussed later).

Principal Component Analysis

- Correlations between the attributes of a data set can be used to **reduce the number of dimensions**:
 - Of two strongly correlated features only one needs to be considered.
 - The other can be reconstructed approximately from the regression line.
 - However, the feature selection can be difficult.
- Better approach: **Principal Component Analysis** (PCA)
 - Find the direction in the data space that has the highest variance.
 - Find the direction in the data space that has the highest variance among those perpendicular to the first.
 - Find the direction in the data space that has the highest variance among those perpendicular to the first and second and so on.
 - Use first directions to describe the data.

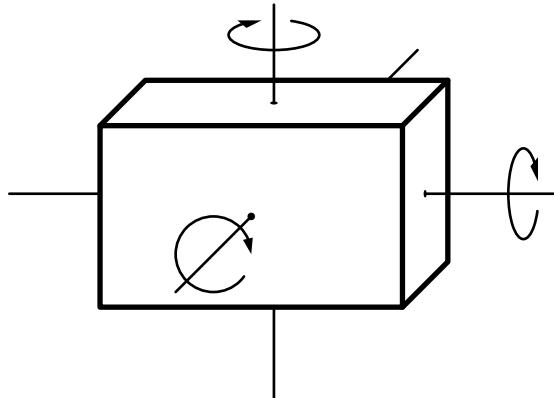
Principal Component Analysis: Physical Analog

- The rotation of a body around an axis through its center of gravity can be described by a so-called **inertia tensor**, which is a 3×3 -matrix

$$\Theta = \begin{pmatrix} \Theta_{xx} & \Theta_{xy} & \Theta_{xz} \\ \Theta_{xy} & \Theta_{yy} & \Theta_{yz} \\ \Theta_{xz} & \Theta_{yz} & \Theta_{zz} \end{pmatrix}.$$

- The diagonal elements of this tensor are called the **moments of inertia**. They describe the “resistance” of the body against being rotated.
- The off-diagonal elements are the so-called **deviation moments**. They describe forces vertical to the rotation axis.
- All bodies possess three perpendicular axes through their center of gravity, around which they can be rotated without forces perpendicular to the rotation axis. These axes are called **principal axes of inertia**.
There are bodies that possess more than 3 such axes (example: a homogeneous sphere), but all bodies have at least three such axes.

Principal Component Analysis: Physical Analog



The principal axes
of inertia of a box.

- The deviation moments cause “rattling” in the bearings of the rotation axis, which cause the bearings to wear out quickly.
- A car mechanic who balances a wheel carries out, in a way, a principal axes transformation. However, instead of changing the orientation of the axes, he/she adds small weights to minimize the deviation moments.
- A statistician who does a principal component analysis, finds, in a way, the axes through a weight distribution with unit weights at each data point, around which it can be rotated most easily.

Principal Component Analysis: Formal Approach

- Normalize all attributes to arithmetic mean 0 and standard deviation 1:

$$x' = \frac{x - \bar{x}}{s_x}$$

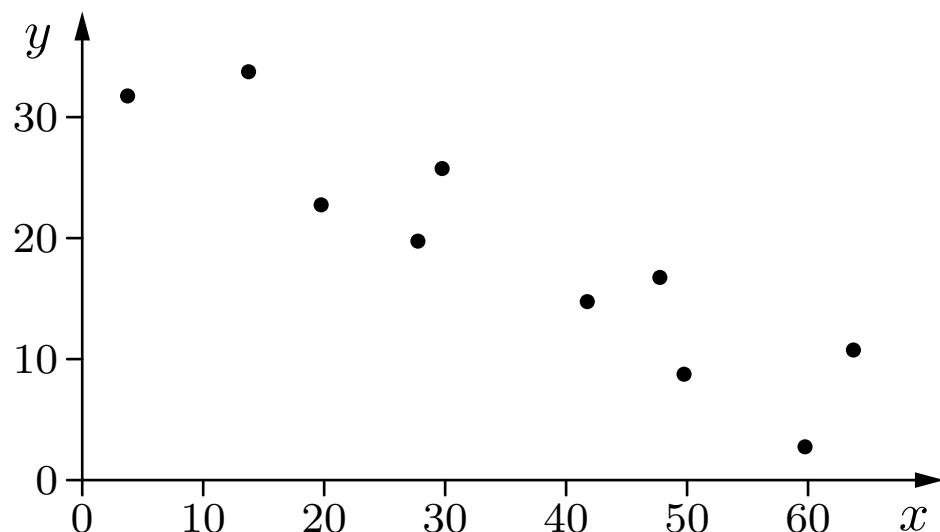
- Compute the **correlation matrix Σ**
(i.e., the covariance matrix of the normalized data)
- Carry out a **principal axes transformation** of the correlation matrix,
that is, find a matrix \mathbf{R} , such that $\mathbf{R}^\top \Sigma \mathbf{R}$ is a diagonal matrix.
- Formal procedure:
 - Find the **eigenvalues** and **eigenvectors** of the correlation matrix,
i.e., find the values λ_i and vectors \vec{v}_i , such that $\Sigma \vec{v}_i = \lambda_i \vec{v}_i$.
 - The eigenvectors indicate the desired directions.
 - The eigenvalues are the variances in these directions.

Principal Component Analysis: Formal Approach

- Select dimensions using the **percentage of explained variance**.
 - The eigenvalues λ_i are the variances σ_i^2 in the principal dimensions.
 - It can be shown that the sum of the eigenvalues of an $m \times m$ correlation matrix is m . Therefore it is plausible to define $\frac{\lambda_i}{m}$ as the share the i -th principal axis has in the total variance.
 - Sort the λ_i descendingly and find the smallest value k , such that
$$\sum_{i=1}^k \frac{\lambda_i}{m} \geq \alpha,$$
where α is a user-defined parameter (e.g. $\alpha = 0.9$).
- Select the corresponding k directions (given by the eigenvectors).
- Transform the data to the new data space by multiplying the data points with a matrix, the rows of which are the eigenvectors of the selected dimensions.

Principal Component Analysis: Example

x	5	15	21	29	31	43	49	51	61	65
y	33	35	24	21	27	16	18	10	4	12



- Strongly correlated features \Rightarrow Reduction to one dimension possible.
- Second dimension may be reconstructed from regression line.

Principal Component Analysis: Example

Normalize to arithmetic mean 0 and standard deviation 1:

$$\bar{x} = \frac{1}{10} \sum_{i=1}^{10} x_i = \frac{370}{10} = 37,$$

$$\bar{y} = \frac{1}{10} \sum_{i=1}^{10} y_i = \frac{200}{10} = 20,$$

$$s_x^2 = \frac{1}{9} \left(\sum_{i=1}^{10} x_i^2 - 10\bar{x}^2 \right) = \frac{17290 - 13690}{9} = 400 \Rightarrow s_x = 20,$$

$$s_y^2 = \frac{1}{9} \left(\sum_{i=1}^{10} y_i^2 - 10\bar{y}^2 \right) = \frac{4900 - 4000}{9} = 100 \Rightarrow s_y = 10.$$

x'	-1.6	-1.1	-0.8	-0.4	-0.3	0.3	0.6	0.7	1.2	1.4
y'	1.3	1.5	0.4	0.1	0.7	-0.4	-0.2	-1.0	-1.6	-0.8

Principal Component Analysis: Example

- Compute the correlation matrix (covariance matrix of normalized data).

$$\Sigma = \frac{1}{9} \begin{pmatrix} 9 & -8.28 \\ -8.28 & 9 \end{pmatrix} = \begin{pmatrix} 1 & -\frac{23}{25} \\ -\frac{23}{25} & 1 \end{pmatrix}.$$

- Find the eigenvalues and eigenvectors, i.e., the values λ_i and vectors \vec{v}_i , $i = 1, 2$, such that

$$\Sigma \vec{v}_i = \lambda_i \vec{v}_i \quad \text{or} \quad (\Sigma - \lambda_i \mathbf{1}) \vec{v}_i = \vec{0}.$$

where $\mathbf{1}$ is the unit matrix.

- Here: Find the eigenvalues as the roots of the characteristic polynomial.

$$c(\lambda) = |\Sigma - \lambda \mathbf{1}| = (1 - \lambda)^2 - \frac{529}{625}.$$

For more than 3 dimensions, this method is numerically unstable and should be replaced by some other method (Jacobi-Transformation, Householder Transformation to tridiagonal form followed by the QR algorithm etc.).

Principal Component Analysis: Example

- The roots of the characteristic polynomial $c(\lambda) = (1 - \lambda)^2 - \frac{529}{625}$ are

$$\lambda_{1/2} = 1 \pm \sqrt{\frac{529}{625}} = 1 \pm \frac{23}{25}, \quad \text{i.e.} \quad \lambda_1 = \frac{48}{25} \quad \text{and} \quad \lambda_2 = \frac{2}{25}$$

- The corresponding eigenvectors are determined by solving for $i = 1, 2$ the (underdetermined) linear equation system

$$(\Sigma - \lambda_i \mathbf{1}) \vec{v}_i = \vec{0}$$

- The resulting eigenvectors (normalized to length 1) are

$$\vec{v}_1 = \left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}} \right) \quad \text{and} \quad \vec{v}_2 = \left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \right),$$

(Note that for two dimensions always these two vectors result.
Reminder: directions of the eigenvectors of a correlation matrix.)

Principal Component Analysis: Example

- Therefore the transformation matrix for the principal axes transformation is

$$\mathbf{R} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}, \quad \text{for which it is} \quad \mathbf{R}^\top \Sigma \mathbf{R} = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

- However, instead of \mathbf{R}^\top we use $\sqrt{2}\mathbf{R}^\top$ to transform the data:

$$\begin{pmatrix} x'' \\ y'' \end{pmatrix} = \sqrt{2} \cdot \mathbf{R}^\top \cdot \begin{pmatrix} x' \\ y' \end{pmatrix}.$$

Resulting data set (additional factor $\sqrt{2}$ leads to nicer values):

x''	-2.9	-2.6	-1.2	-0.5	-1.0	0.7	0.8	1.7	2.8	2.2
y''	-0.3	0.4	-0.4	-0.3	0.4	-0.1	0.4	-0.3	-0.4	0.6

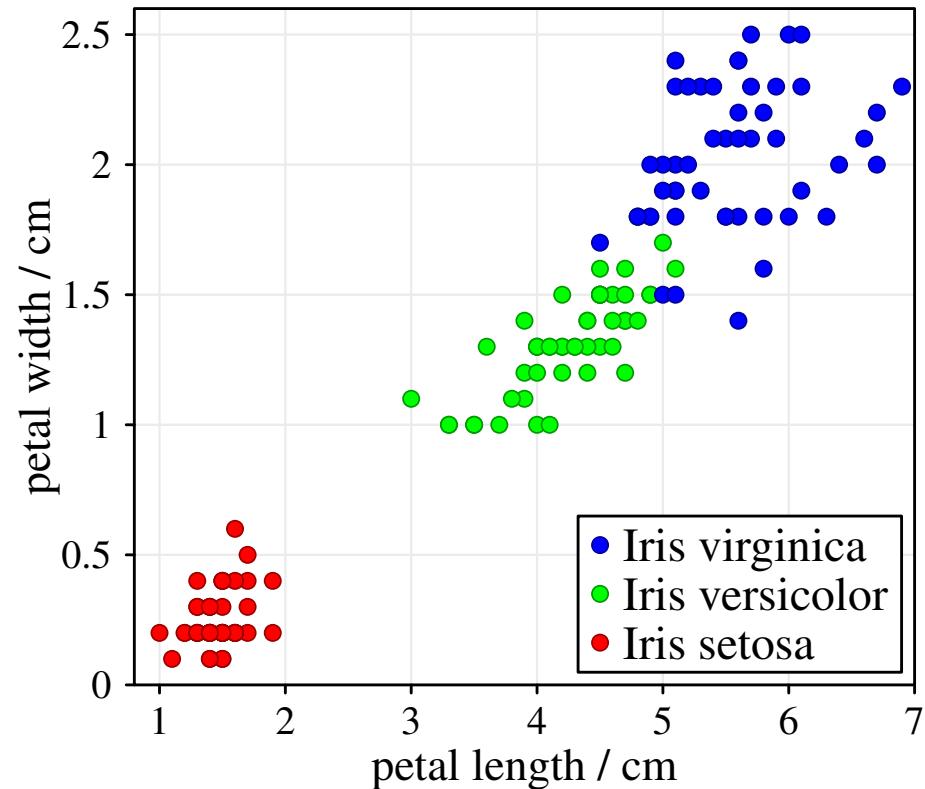
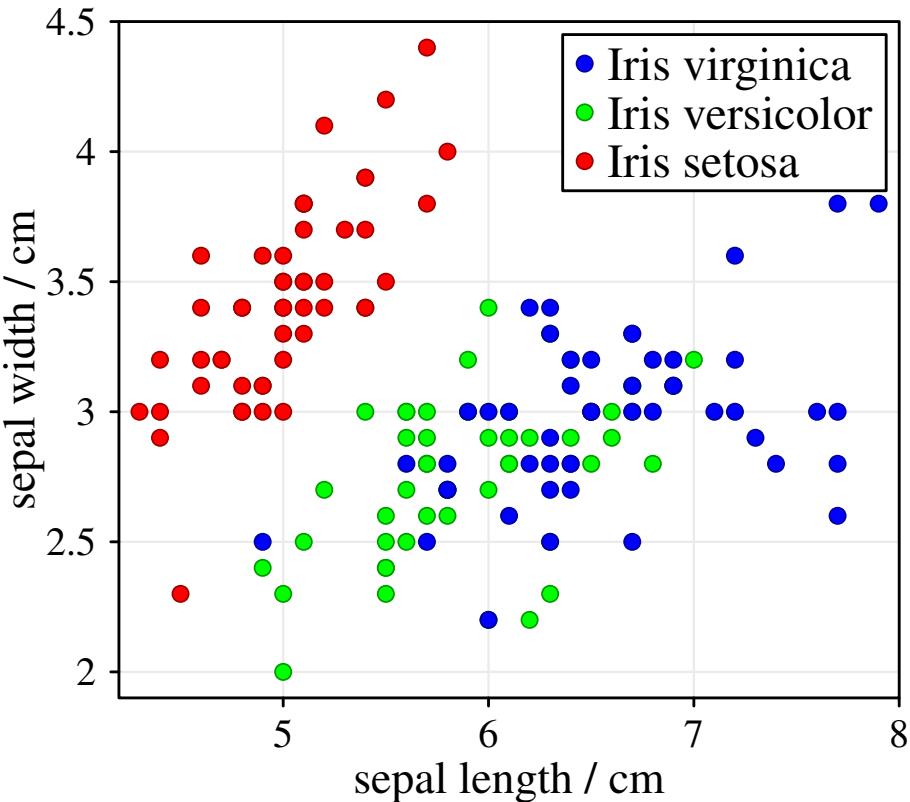
- y'' is discarded ($s_{y''}^2 = 2\lambda_2 = \frac{4}{25}$) and only x'' is kept ($s_{x''}^2 = 2\lambda_1 = \frac{96}{25}$).

The Iris Data

pictures not available in online version

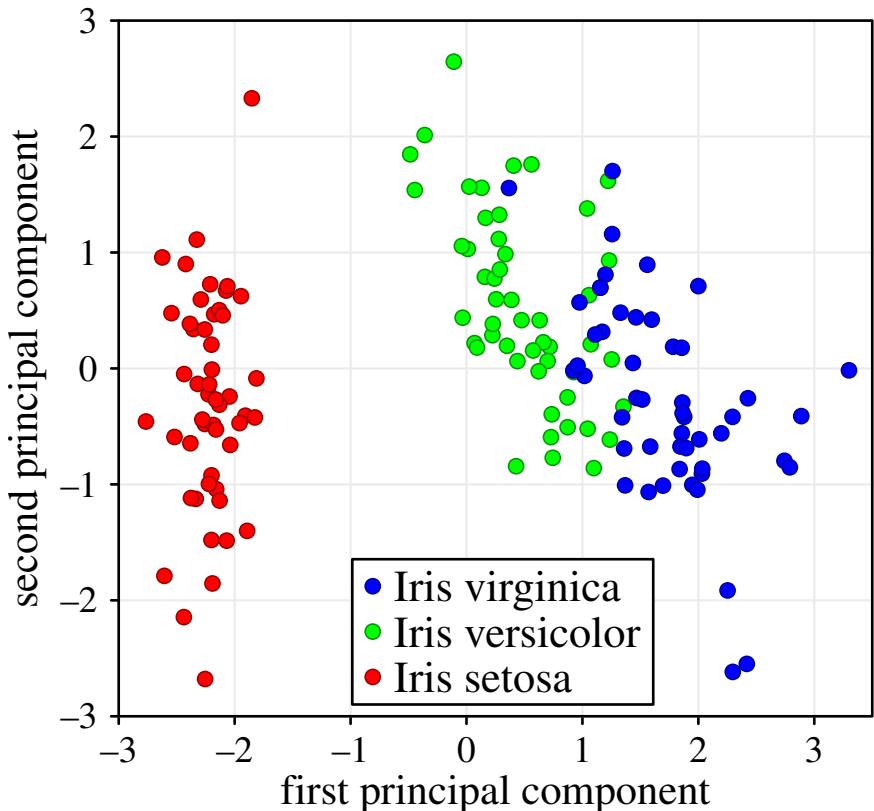
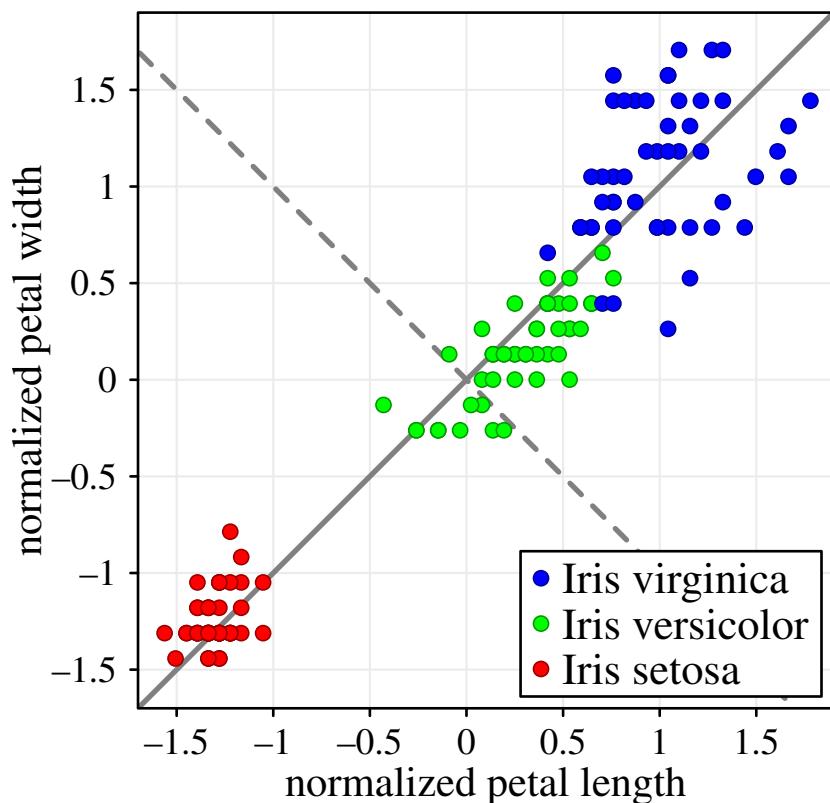
- Collected by Edgar Anderson on the Gaspé Peninsula (Canada).
- First analyzed by Ronald Aylmer Fisher (famous statistician).
- 150 cases in total, 50 cases per Iris flower type.
- Measurements of sepal length and width and petal length and width (in cm).
- Most famous data set in pattern recognition and data analysis.

The Iris Data



- Scatter plots of the iris data set for sepal length vs. sepal width (left) and for petal length vs. petal width (right). All quantities are measured in centimeters (cm).

Principal Component Analysis: Iris Data



- Left: the first (solid line) and the second principal component (dashed line).
- Right: the iris data projected to the space that is spanned by the first and the second principal component (resulting from a PCA involving all four attributes).

Inductive Statistics

Inductive Statistics: Main Tasks

- **Parameter Estimation**

Given an assumption about the type of distribution of the underlying random variable the parameter(s) of the distribution function is estimated.

- **Hypothesis Testing**

A hypothesis about the data generating process is tested by means of the data.

- *Parameter Test*

Test whether a parameter can have certain values.

- *Goodness-of-Fit Test*

Test whether a distribution assumption fits the data.

- *Dependence Test*

Test whether two attributes are dependent.

- **Model Selection**

Among different models that can be used to explain the data the best fitting is selected, taking the complexity of the model into account.

Inductive Statistics: Random Samples

- In inductive statistics probability theory is applied to make inferences about the process that generated the data. This presupposes that the sample is the result of a random experiment, a so-called **random sample**.
- The random variable yielding the sample value x_i is denoted X_i . x_i is called a **instantiation** of the random variable X_i .
- A random sample $x = (x_1, \dots, x_n)$ is an instantiation of the **random vector** $X = (X_1, \dots, X_n)$.
- A random sample is called **independent** if the random variables X_1, \dots, X_n are (stochastically) independent, i.e. if

$$\forall c_1, \dots, c_n \in \mathbb{R} : P\left(\bigwedge_{i=1}^n X_i \leq c_i\right) = \prod_{i=1}^n P(X_i \leq c_i).$$

- An independent random sample is called **simple** if the random variables X_1, \dots, X_n have the same distribution function.

Inductive Statistics: Parameter Estimation

Parameter Estimation

Given:

- A data set and
- a family of parameterized distributions functions of the same type, e.g.
 - the family of binomial distributions $b_X(x; p, n)$ with the parameters p , $0 \leq p \leq 1$, and $n \in \mathbb{N}$, where n is the sample size,
 - the family of normal distributions $N_X(x; \mu, \sigma^2)$ with the parameters μ (expected value) and σ^2 (variance).

Assumption:

- The process that generated the data can be described well by an element of the given family of distribution functions.

Desired:

- The element of the given family of distribution functions (determined by its parameters) that is the best model for the data.

Parameter Estimation

- Methods that yield an estimate for a parameter are called **estimators**.
- Estimators are **statistics**, i.e. functions of the values in a sample.

As a consequence they are functions of (instantiations of) random variables and thus (instantiations of) random variables themselves.

Therefore we can use all of probability theory to analyze estimators.
- There are two types of parameter estimation:
 - **Point Estimators**

Point estimators determine the best value of a parameter w.r.t. the data and certain quality criteria.
 - **Interval Estimators**

Interval estimators yield a region, a so-called **confidence interval**, in which the true value of the parameter lies with high certainty.

Inductive Statistics:

Point Estimation

Point Estimation

Not all statistics, that is, not all functions of the sample values are reasonable and useful estimator. Desirable properties are:

- **Consistency**

With growing data volume the estimated value should get closer and closer to the true value, at least with higher and higher probability.

Formally: If T is an estimator for the parameter θ , it should be

$$\forall \varepsilon > 0 : \lim_{n \rightarrow \infty} P(|T - \theta| < \varepsilon) = 1,$$

where n is the sample size.

- **Unbiasedness**

An estimator should not tend to over- or underestimate the parameter.

Rather it should yield, on average, the correct value.

Formally this means

$$E(T) = \theta.$$

Point Estimation

- **Efficiency**

The estimation should be as precise as possible, that is, the deviation from the true value should be as small as possible. Formally: If T and U are two estimators for the same parameter θ , then T is called *more efficient* than U if

$$D^2(T) < D^2(U).$$

- **Sufficiency**

An estimator should exploit all information about the parameter contained in the data. More precisely: two samples that yield the same estimate should have the same probability (otherwise there is unused information).

Formally: an estimator T for a parameter θ is called *sufficient* iff for all samples $x = (x_1, \dots, x_n)$ with $T(x) = t$ the expression

$$\frac{f_{X_1}(x_1; \theta) \cdots f_{X_n}(x_n; \theta)}{f_T(t; \theta)}$$

is independent of θ .

Point Estimation: Example

Given: a family of **uniform distributions** on the interval $[0, \theta]$, i.e.

$$f_X(x; \theta) = \begin{cases} \frac{1}{\theta}, & \text{if } 0 \leq x \leq \theta, \\ 0, & \text{otherwise.} \end{cases}$$

Desired: an estimate for the unknown parameter θ .

- We will now consider two estimators for the parameter θ and compare their properties.
 - $T = \max\{X_1, \dots, X_n\}$
 - $U = \frac{n+1}{n} \max\{X_1, \dots, X_n\}$
- **General approach:**
 - Find the probability density function of the estimator.
 - Check the desirable properties by exploiting this density function.

Point Estimation: Example

To analyze the estimator $T = \max\{X_1, \dots, X_n\}$, we compute its density function:

$$\begin{aligned} f_T(t; \theta) &= \frac{d}{dt} F_T(t; \theta) = \frac{d}{dt} P(T \leq t) \\ &= \frac{d}{dt} P(\max\{X_1, \dots, X_n\} \leq t) \\ &= \frac{d}{dt} P\left(\bigwedge_{i=1}^n X_i \leq t\right) = \frac{d}{dt} \prod_{i=1}^n P(X_i \leq t) \\ &= \frac{d}{dt} (F_X(t; \theta))^n = n \cdot (F_X(t; \theta))^{n-1} f_X(t; \theta) \end{aligned}$$

where

$$F_X(x; \theta) = \int_{-\infty}^x f_X(x; \theta) dx = \begin{cases} 0, & \text{if } x \leq 0, \\ \frac{x}{\theta}, & \text{if } 0 \leq x \leq \theta, \\ 1, & \text{if } x \geq \theta. \end{cases}$$

Therefore it is

$$f_T(t; \theta) = \frac{n \cdot t^{n-1}}{\theta^n} \quad \text{for } 0 \leq t \leq \theta, \quad \text{and 0 otherwise.}$$

Point Estimation: Example

- The estimator $T = \max\{X_1, \dots, X_n\}$ is **consistent**:

$$\begin{aligned}\lim_{n \rightarrow \infty} P(|T - \theta| < \epsilon) &= \lim_{n \rightarrow \infty} P(T > \theta - \epsilon) \\&= \lim_{n \rightarrow \infty} \int_{\theta-\epsilon}^{\theta} \frac{n \cdot t^{n-1}}{\theta^n} dt = \lim_{n \rightarrow \infty} \left[\frac{t^n}{\theta^n} \right]_{\theta-\epsilon}^{\theta} \\&= \lim_{n \rightarrow \infty} \left(\frac{\theta^n}{\theta^n} - \frac{(\theta - \epsilon)^n}{\theta^n} \right) \\&= \lim_{n \rightarrow \infty} \left(1 - \left(\frac{\theta - \epsilon}{\theta} \right)^n \right) = 1\end{aligned}$$

- It is **not unbiased**:

$$\begin{aligned}E(T) &= \int_{-\infty}^{\infty} t \cdot f_T(t; \theta) dt = \int_0^{\theta} t \cdot \frac{n \cdot t^{n-1}}{\theta^n} dt \\&= \left[\frac{n \cdot t^{n+1}}{(n+1)\theta^n} \right]_0^{\theta} = \frac{n}{n+1} \theta < \theta \quad \text{for } n < \infty.\end{aligned}$$

Point Estimation: Example

- The estimator $U = \frac{n+1}{n} \max\{X_1, \dots, X_n\}$ has the density function

$$f_U(u; \theta) = \frac{n^{n+1}}{(n+1)^n} \frac{u^{n-1}}{\theta^n} \quad \text{for } 0 \leq t \leq \frac{n+1}{n}\theta, \text{ and 0 otherwise.}$$

- The estimator U is **consistent** (without formal proof).
- It is **unbiased**:

$$\begin{aligned} E(U) &= \int_{-\infty}^{\infty} u \cdot f_U(u; \theta) du \\ &= \int_0^{\frac{n+1}{n}\theta} u \cdot \frac{n^{n+1}}{(n+1)^n} \frac{u^{n-1}}{\theta^n} du \\ &= \frac{n^{n+1}}{(n+1)^n \theta^n} \left[\frac{u^{n+1}}{n+1} \right]_0^{\frac{n+1}{n}\theta} \\ &= \frac{n^{n+1}}{(n+1)^n \theta^n} \cdot \frac{1}{n+1} \left(\frac{n+1}{n} \theta \right)^{n+1} = \theta \end{aligned}$$

Point Estimation: Example

Given: a family of **normal distributions** $N_X(x; \mu, \sigma^2)$

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

Desired: estimates for the unknown parameters μ and σ^2 .

- The median and the arithmetic mean of the sample are both consistent and unbiased estimators for the parameter μ .
The median is less efficient than the arithmetic mean.
- The function $V^2 = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})^2$ is a consistent, but **biased** estimator for the parameter σ^2 (it tends to underestimate the variance).
The function $S^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2$, however, is a consistent and **unbiased** estimator for σ^2 (this explains the definition of the empirical variance).

Point Estimation: Example

Given: a family of **polynomial distributions**

$$f_{X_1, \dots, X_k}(x_1, \dots, x_k; \theta_1, \dots, \theta_k, n) = \frac{n!}{\prod_{i=1}^k x_i!} \prod_{i=1}^k \theta_i^{x_i},$$

(n is the sample size, the x_i are the frequencies of the different values a_i , $i = 1, \dots, k$, and the θ_i are the probabilities with which the values a_i occur.)

Desired: estimates for the unknown parameters $\theta_1, \dots, \theta_k$

- The relative frequencies $R_i = \frac{X_i}{n}$ of the different values a_i , $i = 1, \dots, k$, are
 - consistent,
 - unbiased,
 - most efficient, and
 - sufficient estimators for the θ_i .

Inductive Statistics: Finding Point Estimators

How Can We Find Estimators?

- Up to now we analyzed given estimators,
now we consider the question how to find them.
- There are three main approaches to find estimators:
 - **Method of Moments**
Derive an estimator for a parameter from the moments of a distribution and its generator function.
(We do not consider this method here.)
 - **Maximum Likelihood Estimation**
Choose the (set of) parameter value(s) that makes the sample most likely.
 - **Maximum A-posteriori Estimation**
Choose a prior distribution on the range of parameter values, apply Bayes' rule to compute the posterior probability from the sample, and choose the (set of) parameter value(s) that maximizes this probability.

Maximum Likelihood Estimation

- General idea: **Choose the (set of) parameter value(s) that makes the sample most likely.**
- If the parameter value(s) were known, it would be possible to compute the probability of the sample. With unknown parameter value(s), however, it is still possible to state this probability as a function of the parameter(s).
- Formally this can be described as choosing the value θ that maximizes

$$L(D; \theta) = f(D | \theta),$$

where D are the sample data and L is called the **Likelihood Function**.

- Technically the estimator is determined by
 - setting up the likelihood function,
 - forming its partial derivative(s) w.r.t. the parameter(s), and
 - setting these derivatives equal to zero (necessary condition for a maximum).

Brief Excursion: Function Optimization

Task: Find values $\vec{x} = (x_1, \dots, x_m)$ such that $f(\vec{x}) = f(x_1, \dots, x_m)$ is optimal.

Often feasible approach:

- A necessary condition for a (local) optimum (maximum or minimum) is that the partial derivatives w.r.t. the parameters vanish (Pierre Fermat).
- Therefore: (Try to) solve the equation system that results from setting all partial derivatives w.r.t. the parameters equal to zero.

Example task: Minimize $f(x, y) = x^2 + y^2 + xy - 4x - 5y$.

Solution procedure:

1. Take the partial derivatives of the objective function and set them to zero:

$$\frac{\partial f}{\partial x} = 2x + y - 4 = 0, \quad \frac{\partial f}{\partial y} = 2y + x - 5 = 0.$$

2. Solve the resulting (here: linear) equation system: $x = 1, \quad y = 2$.

Maximum Likelihood Estimation: Example

Given: a family of **normal distributions** $N_X(x; \mu, \sigma^2)$

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

Desired: estimators for the unknown parameters μ and σ^2 .

The **Likelihood Function**, which describes the probability of the data, is

$$L(x_1, \dots, x_n; \mu, \sigma^2) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x_i - \mu)^2}{2\sigma^2}\right).$$

To simplify the technical task of forming the partial derivatives, we consider the natural logarithm of the likelihood function, i.e.

$$\ln L(x_1, \dots, x_n; \mu, \sigma^2) = -n \ln\left(\sqrt{2\pi\sigma^2}\right) - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2.$$

Maximum Likelihood Estimation: Example

- Estimator for the **expected value** μ :

$$\frac{\partial}{\partial \mu} \ln L(x_1, \dots, x_n; \mu, \sigma^2) = \frac{1}{\sigma^2} \sum_{i=1}^n (x_i - \mu) \stackrel{!}{=} 0$$
$$\Rightarrow \sum_{i=1}^n (x_i - \mu) = \left(\sum_{i=1}^n x_i \right) - n\mu \stackrel{!}{=} 0 \quad \Rightarrow \quad \hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i$$

- Estimator for the **variance** σ^2 :

$$\frac{\partial}{\partial \sigma^2} \ln L(x_1, \dots, x_n; \mu, \sigma^2) = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{i=1}^n (x_i - \mu)^2 \stackrel{!}{=} 0$$
$$\Rightarrow \hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2 = \frac{1}{n} \sum_{i=1}^n x_i^2 - \frac{1}{n^2} \left(\sum_{i=1}^n x_i \right)^2 \quad (\text{biased!})$$

Maximum A-posteriori Estimation: Motivation

Consider the following three situations:

- A drunkard claims to be able to predict the side on which a thrown coin will land (head or tails). On ten trials he always states the correct side beforehand.
- A tea lover claims that she is able to taste whether the tea or the milk was poured into the cup first. On ten trials she always identifies the correct order.
- An expert of classical music claims to be able to recognize from a single sheet of music whether the composer was Mozart or somebody else. On ten trials he is indeed correct every time.

Maximum likelihood estimation treats all situations alike, because formally the samples are the same. However, this is implausible:

- We do not believe the drunkard at all, despite the sample data.
- We highly doubt the tea drinker, but tend to consider the data as evidence.
- We tend to believe the music expert easily.

Maximum A-posteriori Estimation

- Background knowledge about the plausible values can be incorporated by
 - using a **prior distribution** on the domain of the parameter and
 - adapting this distribution with **Bayes' rule** and the data.
- Formally maximum a-posteriori estimation is defined as follows:
find the parameter value θ that maximizes

$$f(\theta \mid D) = \frac{f(D \mid \theta)f(\theta)}{f(D)} = \frac{f(D \mid \theta)f(\theta)}{\int_{-\infty}^{\infty} f(D \mid \theta)f(\theta) d\theta}$$

- As a comparison: maximum likelihood estimation maximizes

$$f(D \mid \theta)$$

- Note that $f(D)$ need not be computed: It is the same for all parameter values and since we are only interested in the value θ that maximizes $f(\theta \mid D)$ and not the *value of* $f(\theta \mid D)$, we can treat it as a normalization constant.

Maximum A-posteriori Estimation: Example

Given: a family of **binomial distributions**

$$f_X(x; \theta, n) = \binom{n}{x} \theta^x (1 - \theta)^{n-x}.$$

Desired: an estimator for the unknown parameter θ .

a) **Uniform prior:** $f(\theta) = 1, \quad 0 \leq \theta \leq 1.$

$$f(\theta \mid D) = \gamma \binom{n}{x} \theta^x (1 - \theta)^{n-x} \cdot 1 \quad \Rightarrow \quad \hat{\theta} = \frac{x}{n}$$

b) **Tendency towards $\frac{1}{2}$:** $f(\theta) = 6\theta(1 - \theta), \quad 0 \leq \theta \leq 1.$

$$\begin{aligned} f(\theta \mid D) &= \gamma \binom{n}{x} \theta^x (1 - \theta)^{n-x} \cdot \theta(1 - \theta) = \gamma \binom{n}{x} \theta^{x+1} (1 - \theta)^{n-x+1} \\ &\Rightarrow \quad \hat{\theta} = \frac{x + 1}{n + 2} \end{aligned}$$

Excursion: Dirichlet's Integral

- For computing the normalization factors of the probability density functions that occur with polynomial distributions, **Dirichlet's Integral** is helpful:

$$\int_{\theta_1} \dots \int_{\theta_k} \prod_{i=1}^k \theta_i^{x_i} d\theta_1 \dots d\theta_k = \frac{\prod_{i=1}^k \Gamma(x_i + 1)}{\Gamma(n + k)}, \quad \text{where } n = \sum_{i=1}^k x_i$$

and the Γ -function is the so-called **generalized factorial**:

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt, \quad x > 0,$$

which satisfies

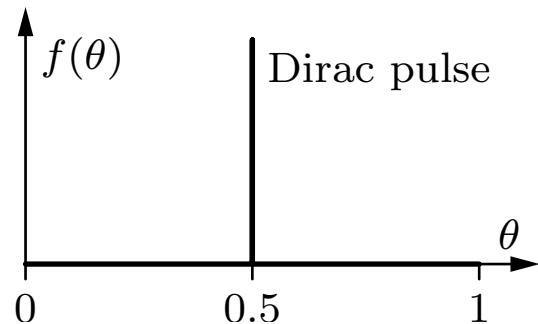
$$\Gamma(x + 1) = x \cdot \Gamma(x), \quad \Gamma(\frac{1}{2}) = \sqrt{\pi}, \quad \Gamma(1) = 1.$$

- **Example:** the normalization factor α for the binomial distribution prior $f(\theta) = \alpha \theta^2(1 - \theta)^3$ is

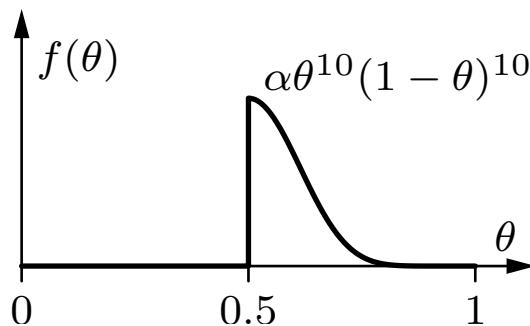
$$\alpha = \frac{1}{\int_\theta \theta^2(1 - \theta)^3 d\theta} = \frac{\Gamma(5 + 2)}{\Gamma(2 + 1) \Gamma(3 + 1)} = \frac{6!}{2! 3!} = \frac{720}{12} = 60.$$

Maximum A-posteriori Estimation: Example

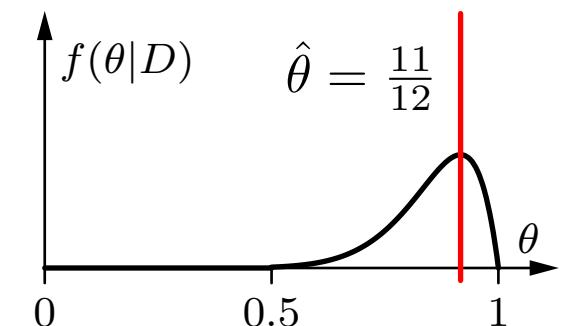
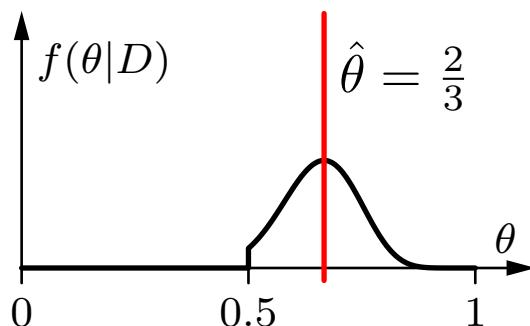
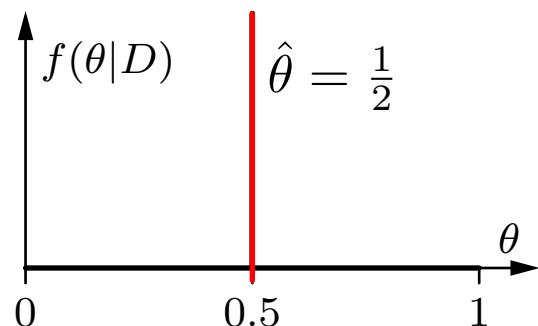
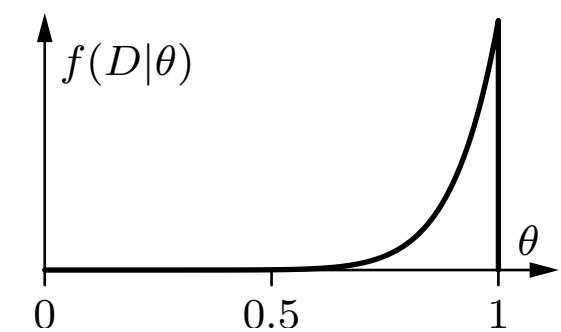
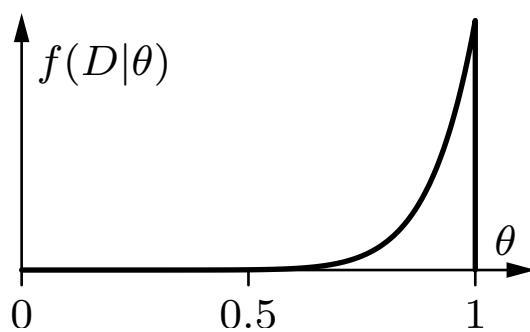
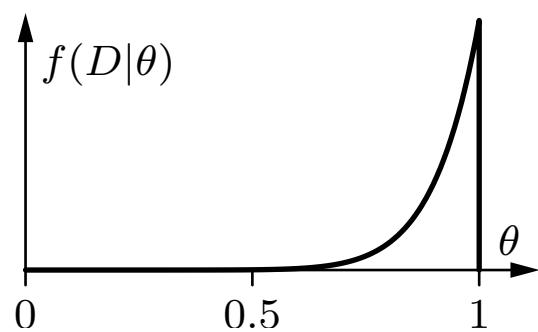
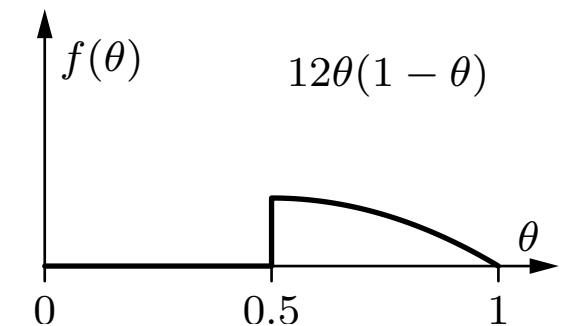
drunkard



tea lover



music expert



Inductive Statistics: Interval Estimation

Interval Estimation

- In general the estimated value of a parameter will differ from the true value.
- It is desirable to be able to make an assertion about the possible deviations.
- The simplest possibility is to state not only a point estimate, but also the standard deviation of the estimator:

$$t \pm D(T) = t \pm \sqrt{D^2(T)}.$$

- A better possibility is to find intervals that contain the true value with high probability. Formally they can be defined as follows:

Let $A = g_A(X_1, \dots, X_n)$ and $B = g_B(X_1, \dots, X_n)$ be two statistics with

$$P(A < \theta < B) = 1 - \alpha, \quad P(\theta \leq A) = \frac{\alpha}{2}, \quad P(\theta \geq B) = \frac{\alpha}{2}.$$

Then the random interval $[A, B]$ (or an instantiation $[a, b]$ of this interval) is called $(1 - \alpha) \cdot 100\%$ **confidence interval** for θ .

The value $1 - \alpha$ is called **confidence level**.

Interval Estimation

- This definition of a confidence interval is not specific enough:
 A and B are not uniquely determined.
- Common solution: Start from a point estimator T for the unknown parameter θ and define A and B as functions of T :

$$A = h_A(T) \quad \text{and} \quad B = h_B(T).$$

- Instead of $A \leq \theta \leq B$ consider the corresponding event w.r.t. the estimator T , that is, $A^* \leq T \leq B^*$.
- Determine $A = h_A(T)$ and $B = h_B(T)$ from the inverse functions $A^* = h_A^{-1}(\theta)$ and $B^* = h_B^{-1}(\theta)$.

$$\begin{aligned}\text{Procedure: } P(A^* < T < B^*) &= 1 - \alpha \\ \Rightarrow P(h_A^{-1}(\theta) < T < h_B^{-1}(\theta)) &= 1 - \alpha \\ \Rightarrow P(h_A(T) < \theta < h_B(T)) &= 1 - \alpha \\ \Rightarrow P(A < \theta < B) &= 1 - \alpha.\end{aligned}$$

Interval Estimation: Example

Given: a family of **uniform distributions** on the interval $[0, \theta]$, i.e.

$$f_X(x; \theta) = \begin{cases} \frac{1}{\theta}, & \text{if } 0 \leq x \leq \theta, \\ 0, & \text{otherwise.} \end{cases}$$

Desired: a confidence interval for the unknown parameter θ .

- Start from the unbiased point estimator $U = \frac{n+1}{n} \max\{X_1, \dots, X_n\}$:

$$P(U \leq B^*) = \int_0^{B^*} f_U(u; \theta) du = \frac{\alpha}{2}$$

$$P(U \geq A^*) = \int_{A^*}^{\frac{n+1}{n}\theta} f_U(u; \theta) du = \frac{\alpha}{2}$$

- From the study of point estimators we know

$$f_U(u; \theta) = \frac{n^{n+1}}{(n+1)^n} \frac{u^{n-1}}{\theta^n}.$$

Interval Estimation: Example

- Solving the integrals gives us

$$B^* = \sqrt[n]{\frac{\alpha}{2}} \frac{n+1}{n} \theta \quad \text{and} \quad A^* = \sqrt[n]{1 - \frac{\alpha}{2}} \frac{n+1}{n} \theta,$$

that is,

$$P\left(\sqrt[n]{\frac{\alpha}{2}} \frac{n+1}{n} \theta < U < \sqrt[n]{1 - \frac{\alpha}{2}} \frac{n+1}{n} \theta\right) = 1 - \alpha.$$

- Computing the inverse functions leads to

$$P\left(\frac{U}{\sqrt[n]{1 - \frac{\alpha}{2}} \frac{n+1}{n}} < \theta < \frac{U}{\sqrt[n]{\frac{\alpha}{2}} \frac{n+1}{n}}\right) = 1 - \alpha,$$

that is,

$$A = \frac{U}{\sqrt[n]{1 - \frac{\alpha}{2}} \frac{n+1}{n}} \quad \text{and} \quad B = \frac{U}{\sqrt[n]{\frac{\alpha}{2}} \frac{n+1}{n}}.$$

Inductive Statistics: Hypothesis Testing

Hypothesis Testing

- A **hypothesis test** is a statistical procedure with which a decision is made between two contrary hypothesis about the process that generated the data.
- The two hypotheses may refer to
 - the value of a parameter (**Parameter Test**),
 - a distribution assumption (**Goodness-of-Fit Test**),
 - the dependence of two attributes (**Dependence Test**).
- One of the two hypothesis is preferred, that is, in case of doubt the decision is made in its favor. (One says that it gets the “benefit of the doubt”.)
- The preferred hypothesis is called the **Null Hypothesis** H_0 ,
the other hypothesis is called the **Alternative Hypothesis** H_a .
- Intuitively: the null hypothesis H_0 is put on trial.
Only if the evidence is strong enough, it is convicted (that is, rejected).
If there is (sufficient) doubt, however, it is acquitted (that is, accepted).

Hypothesis Testing

- The test decision is based on a **test statistic**, that is, a function of the sample values.
- The null hypothesis is rejected if the value of the test statistic lies inside the so-called **critical region** C .
- Developing a hypothesis test consists in finding the critical region for a given test statistic and significance level (see below).
- The test decision may be wrong. There are two possible types of errors:
 - Type 1:** The null hypothesis H_0 is rejected, even though it is correct.
 - Type 2:** The null hypothesis H_0 is accepted, even though it is false.
- Type 1 errors are considered to be more severe, since the null hypothesis gets the “benefit of the doubt”.
- Hence it is tried to limit the probability of a type 1 error to a certain maximum α . This maximum value α is called **significance level**.

Parameter Test

- In a parameter test the contrary hypotheses refer to the value of a parameter, for example (one-sided test):

$$H_0 : \theta \geq \theta_0, \quad H_a : \theta < \theta_0.$$

- For such a test usually a point estimator T is chosen as the test statistic.
- The null hypothesis H_0 is rejected if the value t of the point estimator does not exceed a certain value c , the so-called **critical value** (that is, $C = (-\infty, c]$).
- Formally the critical value c is determined as follows: We consider

$$\beta(\theta) = P_\theta(H_0 \text{ is rejected}) = P_\theta(T \in C),$$

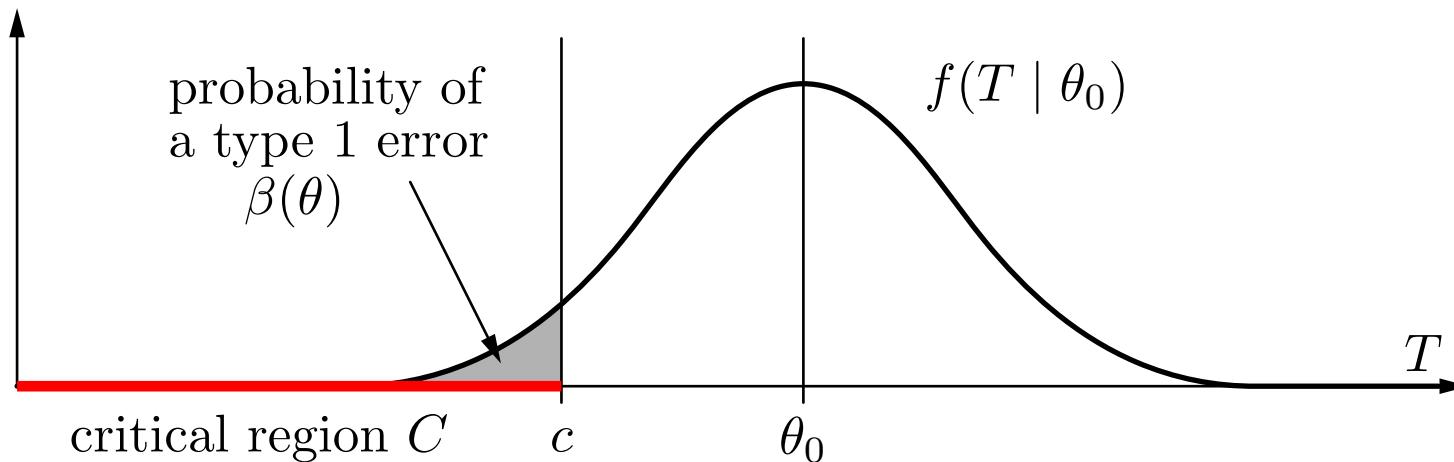
the so-called **power** β of the test.

- The power must be not exceed the significance level α for values θ satisfying H_0 :

$$\max_{\theta: \theta \text{ satisfies } H_0} \beta(\theta) \leq \alpha. \quad (\text{here: } \beta(\theta_0) \leq \alpha)$$

Parameter Test: Intuition

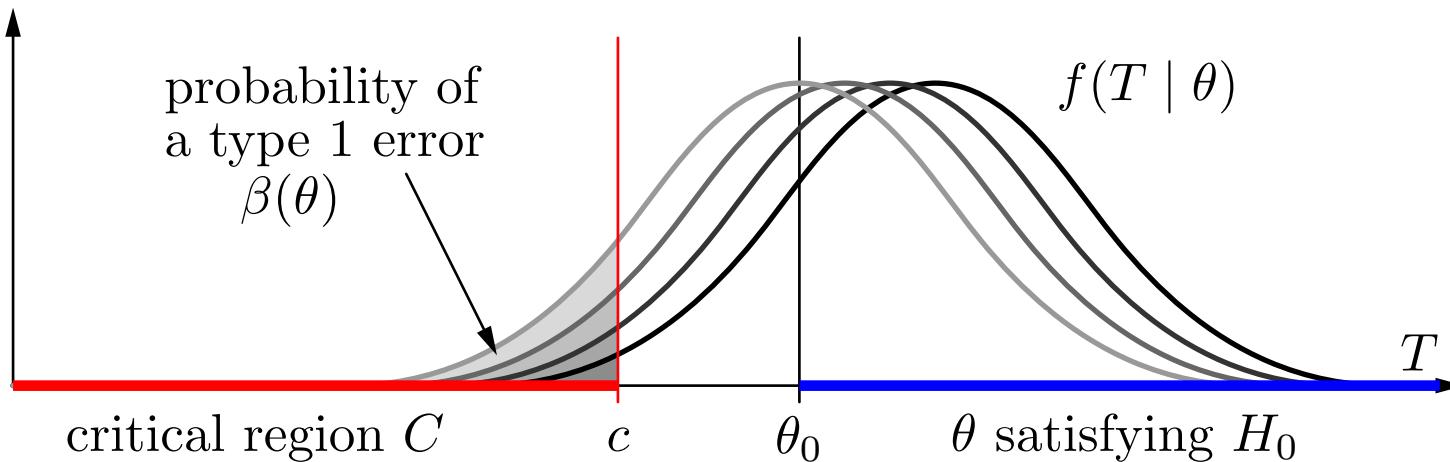
- The probability of a type 1 error is the area under the estimator's probability density function $f(T | \theta_0)$ to the left of the critical value c .
(Note: This example illustrates $H_0 : \theta \geq \theta_0$ and $H_a : \theta < \theta_0$.)



- Obviously the probability of a type 1 error depends on the location of the critical value c : higher values mean a higher error probability.
- Idea: Choose the location of the critical value so that the maximal probability of a type 1 error equals α , the chosen significance level.

Parameter Test: Intuition

- What is so special about θ_0 that we use $f(T \mid \theta_0)$?



- In principle, all θ satisfying H_0 have to be considered, that is, all density functions $f(T \mid \theta)$ with $\theta \geq \theta_0$.
- Among these values θ , the one with the highest probability of a type 1 error (that is, the one with the highest power $\beta(\theta)$) determines the critical value.
Intuitively: we consider the **worst possible case**.

Parameter Test: Example

- Consider a one-sided test of the expected value μ of a normal distribution $N(\mu, \sigma^2)$ with known variance σ^2 , that is, consider the hypotheses

$$H_0 : \mu \geq \mu_0, \quad H_a : \mu < \mu_0.$$

- As a test statistic we use the standard point estimator for the expected value

$$\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i.$$

This point estimator has the probability density

$$f_{\bar{X}}(x) = N\left(x; \mu, \frac{\sigma^2}{n}\right).$$

- Therefore it is (with the $N(0, 1)$ -distributed random variable Z)

$$\alpha = \beta(\mu_0) = P_{\mu_0}(\bar{X} \leq c) = P\left(\frac{\bar{X} - \mu_0}{\sigma/\sqrt{n}} \leq \frac{c - \mu_0}{\sigma/\sqrt{n}}\right) = P\left(Z \leq \frac{c - \mu_0}{\sigma/\sqrt{n}}\right).$$

Parameter Test: Example

- We have as a result that

$$\alpha = \Phi\left(\frac{c - \mu_0}{\sigma/\sqrt{n}}\right),$$

where Φ is the distribution function of the standard normal distribution.

- The distribution function Φ is tabulated, because it cannot be represented in closed form. From such a table we retrieve the value z_α satisfying $\alpha = \Phi(z_\alpha)$.
- Then the critical value is

$$c = \mu_0 + z_\alpha \frac{\sigma}{\sqrt{n}}.$$

(Note that the value of z_α is negative due to the usually small value of α . Typical values are $\alpha = 0.1$, $\alpha = 0.05$ or $\alpha = 0.01$.)

- H_0 is rejected if the value \bar{x} of the point estimator \bar{X} does not exceed c , otherwise it is accepted.

Parameter Test: Example

- Let $\sigma = 5.4$, $n = 25$ and $\bar{x} = 128$. We choose $\mu_0 = 130$ and $\alpha = 0.05$.
- From a standard normal distribution table we retrieve $z_{0.05} \approx -1.645$ and get

$$c_{0.05} \approx 130 - 1.645 \frac{5.4}{\sqrt{25}} \approx 128.22.$$

Since $\bar{x} = 128 < 128.22 = c$, we reject the null hypothesis H_0 .

- If, however, we had chosen $\alpha = 0.01$, it would have been (with $z_{0.01} \approx -2.326$):

$$c_{0.01} \approx 130 - 2.326 \frac{5.4}{\sqrt{25}} \approx 127.49$$

Since $\bar{x} = 128 > 127.49 = c$, we would have accepted the null hypothesis H_0 .

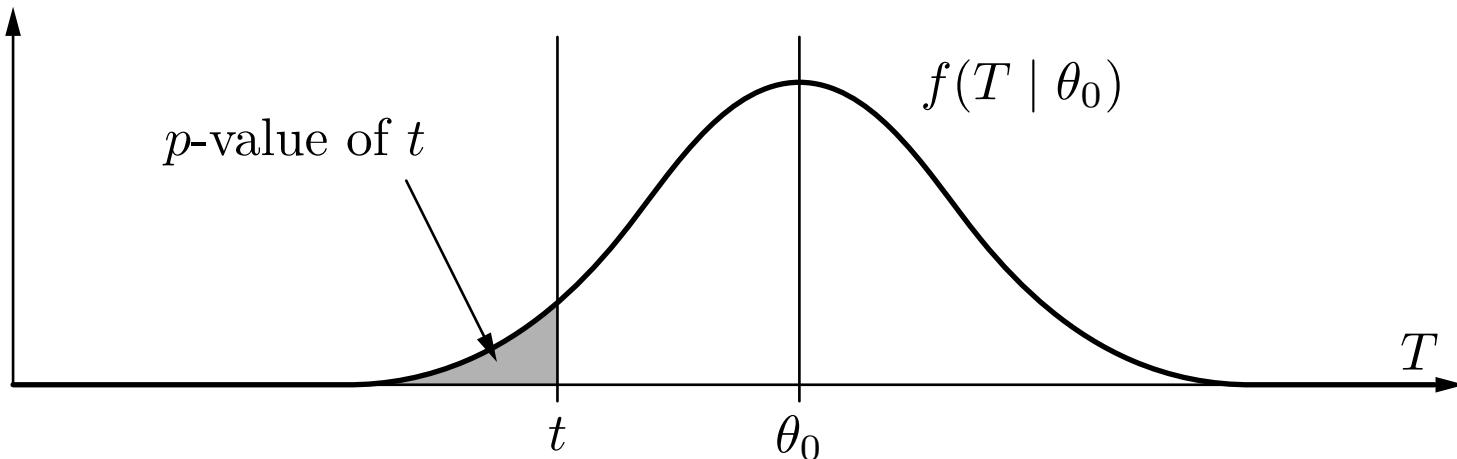
- Instead of fixing a significance level α one may state the so-called **p-value**

$$p = \Phi \left(\frac{128 - 130}{5.4/\sqrt{25}} \right) \approx 0.032.$$

For $\alpha \geq p = 0.032$ the null hypothesis is rejected, for $\alpha < p = 0.032$ accepted.

Parameter Test: p-value

- Let t be the value of the test statistic T that has been computed from a given data set.
(Note: This example illustrates $H_0 : \theta \geq \theta_0$ and $H_a : \theta < \theta_0$.)



- The **p-value** is the probability that a value of t or less can be observed for the chosen test statistic T .
- The p -value is a **lower limit for the significance level α** that may have been chosen if we wanted to reject the null hypothesis H_0 .

Parameter Test: p-value

Attention: p-values are often misused or misinterpreted!

- A low p -value does **not** mean that the result is very reliable!
All that matters for the test is whether the computed p -value
is **below the chosen significance level or not.**
(A low p -value could just be a chance event, an accident!)
- The significance level may **not** be chosen **after** computing the p -value,
since we tend to choose lower significance levels if we know that they are met.
Doing so would undermine the reliability of the procedure!
- Stating p -values is only a convenient way of avoiding a fixed significance level
(since significance levels are a matter of choice and thus user-dependent).

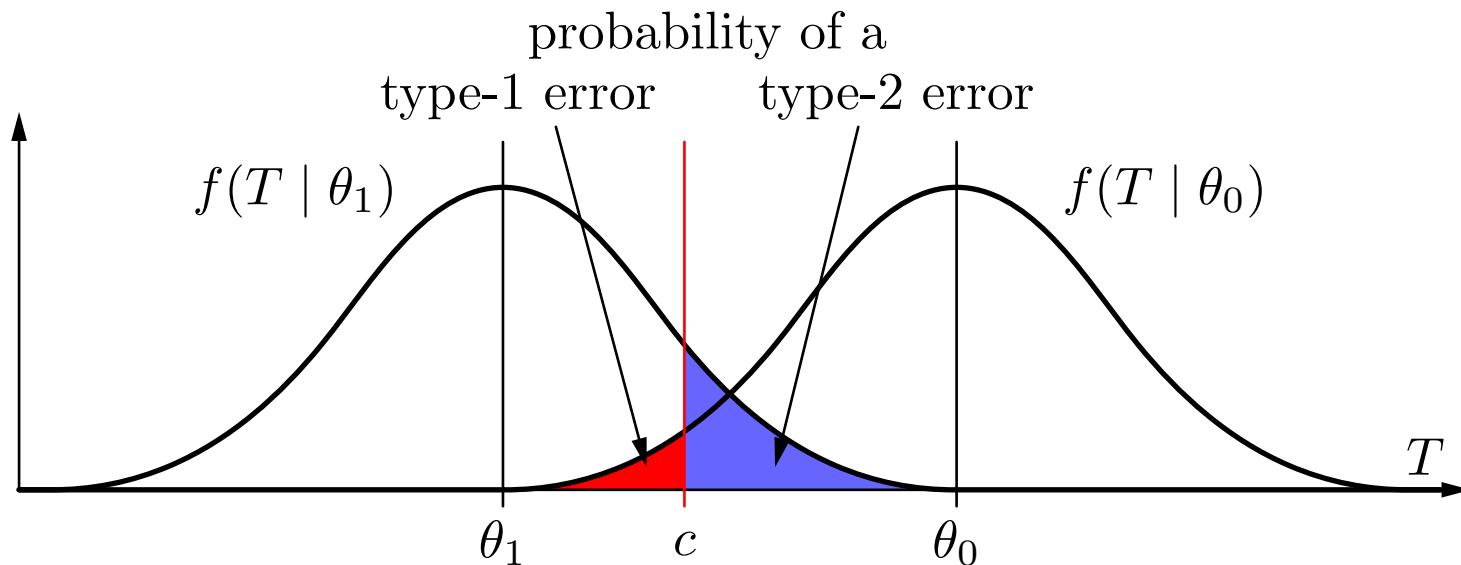
However: A significance level must still be chosen
before a reported p -value is looked at.

Relevance of the Type-2 Error

- Reminder: There are two possible types of errors:
 - Type 1:** The null hypothesis H_0 is rejected, even though it is correct.
 - Type 2:** The null hypothesis H_0 is accepted, even though it is false.
- Type-1 errors are considered to be more severe,
since the null hypothesis gets the “benefit of the doubt”.
- However, **type-2 errors should not be neglected** completely:
 - It is always possible to achieve a vanishing probability of a type-1 error:
Simply accept the null hypothesis in all instances, regardless of the data.
 - Unfortunately such an approach maximizes the type-2 error.
- Generally, **type-1 and type-2 errors are complementary quantities**:
The lower we require the type-1 error to be (the lower the significance level),
the higher will be the probability of a type-2 error.

Relationship between Type-1 and Type-2 Error

- Suppose there are only two possible parameter values θ_0 and θ_1 with $\theta_1 < \theta_0$.
(That is, we have $H_0 : \theta = \theta_0$ and $H_a : \theta = \theta_1$.)



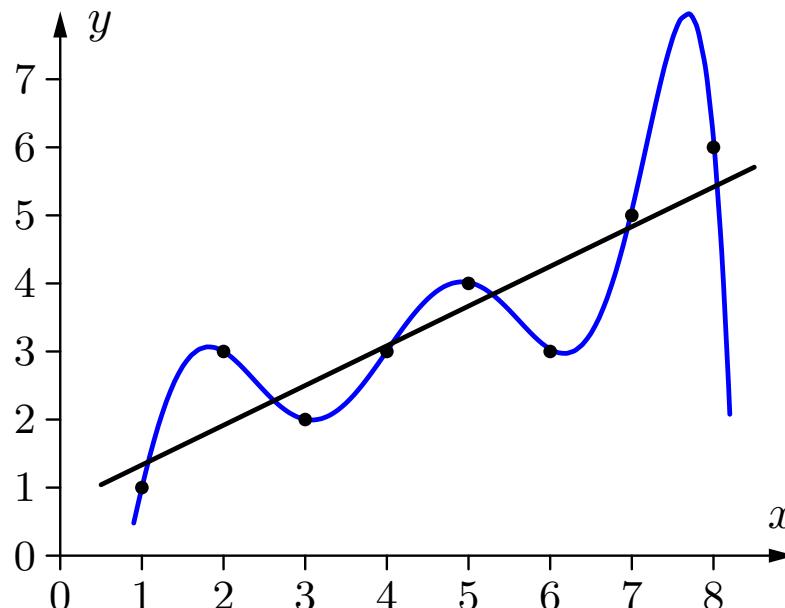
- Lowering the significance level α moves the critical value c to the left:
lower type-1 error (red), but higher type-2 error (blue).
- Increasing the significance level α moves the critical value c to the right:
higher type-1 error (red), but lower type-2 error (blue).

Inductive Statistics: Model Selection

Model Selection

- Objective: select the model that best fits the data,
taking the model complexity into account.

The more complex the model, the better it usually fits the data.



black line:
regression line
(2 free parameters)

blue curve:
7th order regression polynomial
(8 free parameters)

- The blue curve fits the data points perfectly, *but it is not a good model.*

Information Criteria

- There is a **tradeoff between model complexity and fit to the data.**
Question: How much better must a more complex model fit the data in order to justify the higher complexity?
- One approach to quantify the tradeoff: **Information Criteria**

Let M be a model and Θ the set of free parameters of M . Then:

$$\text{IC}_\kappa(M, \Theta \mid D) = -2 \ln P(D \mid M, \Theta) + \kappa |\Theta|,$$

where D are the sample data and κ is a parameter.

Special cases:

- **Akaike Information Criterion (AIC):** $\kappa = 2$
 - **Bayesian Information Criterion (BIC):** $\kappa = \ln n$,
where n is the sample size
- The lower the value of these measures, the better the model.

Minimum Description Length

- Idea: Consider the transmission of the data from a sender to a receiver.
Since the transmission of information is costly,
the length of the message to be transmitted should be minimized.
- A good model of the data can be used to transmit the data with fewer bits.
However, the receiver does not know the model the sender used
and thus cannot decode the message.
Therefore: if the sender uses a model, he/she has to transmit the model as well.
- **description length** = **length of model description**
+ **length of data description**
(A more complex model increases the length of the model description,
but reduces the length of the data description.)
- The model that leads to the smallest total description length is the best.

Minimum Description Length: Example

- Given: a one-dimensional sample from a polynomial distribution.
- Question: are the probabilities of the attribute values sufficiently different to justify a non-uniform distribution model?
- Coding using **no model** (equal probabilities for all values):

$$l_1 = n \log_2 k,$$

where n is the sample size and k the number of attribute values.

- Coding using a **polynomial distribution model**:

$$l_2 = \underbrace{\log_2 \frac{(n+k-1)!}{n!(k-1)!}}_{\text{model description}} + \underbrace{\log_2 \frac{n!}{x_1! \dots x_k!}}_{\text{data description}}$$

(Idea: Use a codebook with one page per configuration, that is, frequency distribution (model) and specific sequence (data), and transmit the page number.)

Minimum Description Length: Example

Some details about the codebook idea:

- **Model Description:**

There are n objects (the sample cases) that have to be partitioned into k groups (one for each attribute value). (Model: distribute n balls on k boxes.)

Number of possible distributions:
$$\frac{(n+k-1)!}{n!(k-1)!}$$

Idea: number of possible sequences of $n+k-1$ objects (n balls and $k-1$ box walls) of which n (the objects) and $k-1$ (the box walls) are indistinguishable.

- **Data Description:**

There are k groups of objects with n_i , $i = 1, \dots, k$, elements in them.
(The values of the n_k are known from the model description.)

Number of possible sequences:
$$\frac{n!}{n_1! \dots n_k!}$$

Summary Statistics

Statistics has two main areas:

- **Descriptive Statistics**

- Display the data in tables or charts.
- Summarize the data in characteristic measures.
- Reduce the dimensionality of the data with principal component analysis.

- **Inductive Statistics**

- Use probability theory to draw inferences about the process that generated the data.
- Parameter Estimation (point and interval)
- Hypothesis Testing (parameter, goodness-of-fit, dependence)
- Model Selection (tradeoff between fit and complexity)

Regression

Regression

- **General Idea of Regression**
 - Method of least squares
- **Linear Regression**
 - An illustrative example
- **Polynomial Regression**
 - Generalization to polynomial functional relationships
- **Multivariate Regression**
 - Generalization to more than one function argument
- **Logistic Regression**
 - Generalization to non-polynomial functional relationships
- **Logistic Classification**
 - Modelling 2-class problems with a logistic function
- **Summary**

Regression

Also known as: **Method of Least Squares** (Carl Friedrich Gauß)

Given:

- A dataset $((\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n))$ of n data tuples (one or more input values and one output value) and
- a hypothesis about the functional relationship between output and input values, e.g. $y = g(x) = a + bx$.

Desired:

- A parameterization of the conjectured function that minimizes the sum of squared errors (“best fit”),

Depending on

- the hypothesis about the functional relationship and
- the number of arguments to the conjectured function

different types of regression are distinguished.

Reminder: Function Optimization

Task: Find values $\vec{x} = (x_1, \dots, x_m)$ such that $f(\vec{x}) = f(x_1, \dots, x_m)$ is optimal.

Often feasible approach:

- A necessary condition for a (local) optimum (maximum or minimum) is that the partial derivatives w.r.t. the parameters vanish (Pierre Fermat, 1601–1665).
- Therefore: (Try to) solve the equation system that results from setting all partial derivatives w.r.t. the parameters equal to zero.

Example task: Minimize $f(x, y) = x^2 + y^2 + xy - 4x - 5y$.

Solution procedure:

1. Take the partial derivatives of the objective function and set them to zero:

$$\frac{\partial f}{\partial x} = 2x + y - 4 = 0, \quad \frac{\partial f}{\partial y} = 2y + x - 5 = 0.$$

2. Solve the resulting (here: linear) equation system: $x = 1, \quad y = 2$.

Linear Regression

- Given:
- A dataset $((x_1, y_1), \dots, (x_n, y_n))$ of n data tuples and
 - a hypothesis about the functional relationship, e.g. $y = g(x) = a + bx$.

Approach: Minimize the sum of squared errors, that is,

$$F(a, b) = \sum_{i=1}^n (g(x_i) - y_i)^2 = \sum_{i=1}^n (a + bx_i - y_i)^2.$$

Necessary conditions for a minimum

(a.k.a. Fermat's theorem, after Pierre de Fermat, 1601–1665):

$$\frac{\partial F}{\partial a} = \sum_{i=1}^n 2(a + bx_i - y_i) = 0 \quad \text{and}$$

$$\frac{\partial F}{\partial b} = \sum_{i=1}^n 2(a + bx_i - y_i)x_i = 0$$

Linear Regression

Result of necessary conditions: System of so-called **normal equations**, that is,

$$na + \left(\sum_{i=1}^n x_i \right) b = \sum_{i=1}^n y_i,$$

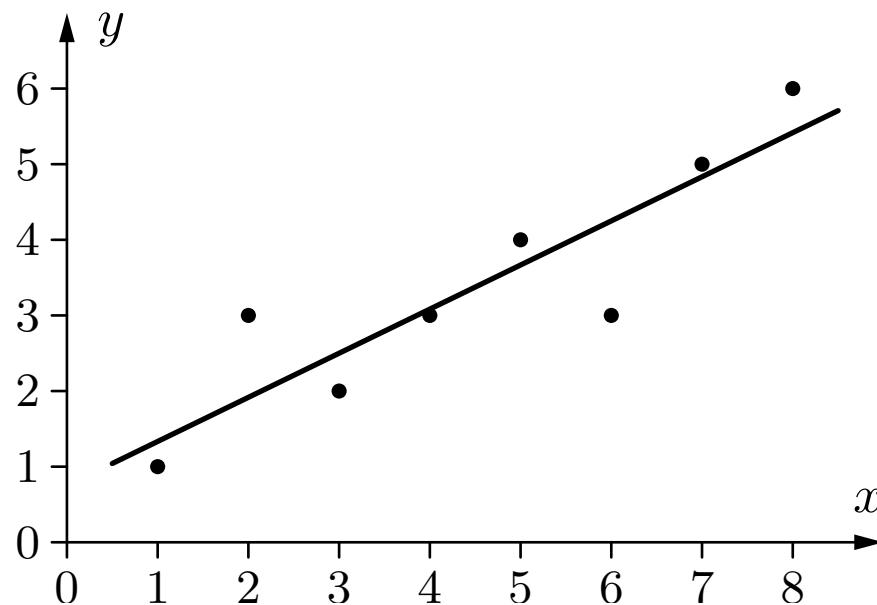
$$\left(\sum_{i=1}^n x_i \right) a + \left(\sum_{i=1}^n x_i^2 \right) b = \sum_{i=1}^n x_i y_i.$$

- Two linear equations for two unknowns a and b .
- System can be solved with standard methods from linear algebra.
- Solution is unique unless all x -values are identical.
- The resulting line is called a **regression line**.

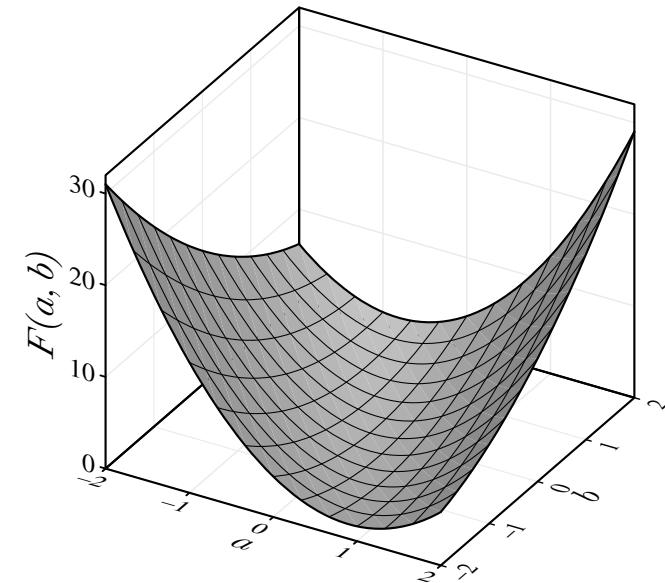
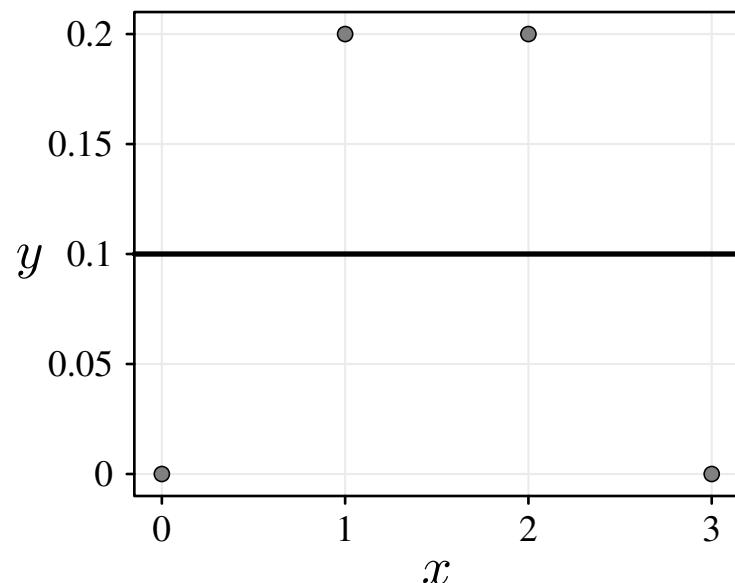
Linear Regression: Example

x	1	2	3	4	5	6	7	8
y	1	3	2	3	4	3	5	6

$$y = \frac{3}{4} + \frac{7}{12}x.$$



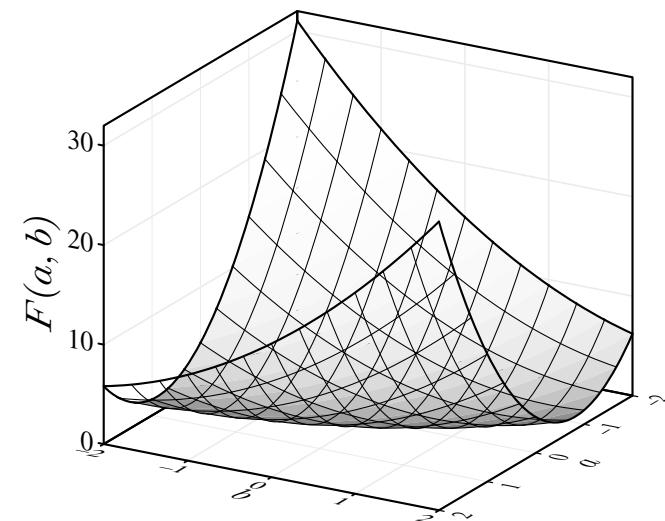
Linear Regression: Example



- A very simple data set (4 points), to which a line is to be fitted.
- The error functional for linear regression

$$F(a, b) = \sum_{i=1}^n (a + bx_i - y_i)^2$$

(same function, two different views).



Least Squares and Maximum Likelihood

A regression line can be interpreted as a **maximum likelihood estimator**:

Assumption: The data generation process can be described well by the model

$$y = a + bx + \xi,$$

where ξ is normally distributed with mean 0 and (unknown) variance σ^2 (σ^2 independent of x , that is, same dispersion of y for all x).

As a consequence we have

$$f(y \mid x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(y - (a + bx))^2}{2\sigma^2}\right).$$

With this expression we can set up the **likelihood function**

$$L((x_1, y_1), \dots, (x_n, y_n); a, b, \sigma^2)$$

$$= \prod_{i=1}^n f(x_i) f(y_i \mid x_i) = \prod_{i=1}^n f(x_i) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(y_i - (a + bx_i))^2}{2\sigma^2}\right).$$

Least Squares and Maximum Likelihood

To simplify taking the derivatives, we compute the natural logarithm:

$$\begin{aligned} & \ln L((x_1, y_1), \dots, (x_n, y_n); a, b, \sigma^2) \\ &= \ln \prod_{i=1}^n f(x_i) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(y_i - (a + bx_i))^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^n \ln f(x_i) + \sum_{i=1}^n \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - (a + bx_i))^2 \end{aligned}$$

From this expression it is clear that (provided $f(x)$ is independent of a , b , and σ^2) maximizing the likelihood function is equivalent to minimizing

$$F(a, b) = \sum_{i=1}^n (y_i - (a + bx_i))^2.$$

Interpreting the method of least squares as a maximum likelihood estimator works also for the generalizations to polynomials and multilinear functions discussed next.

Polynomial Regression

Generalization to polynomials

$$y = p(x) = a_0 + a_1x + \dots + a_mx^m$$

Approach: Minimize the sum of squared errors, that is,

$$F(a_0, a_1, \dots, a_m) = \sum_{i=1}^n (p(x_i) - y_i)^2 = \sum_{i=1}^n (a_0 + a_1x_i + \dots + a_mx_i^m - y_i)^2$$

Necessary conditions for a minimum: All partial derivatives vanish, that is,

$$\frac{\partial F}{\partial a_0} = 0, \quad \frac{\partial F}{\partial a_1} = 0, \quad \dots, \quad \frac{\partial F}{\partial a_m} = 0.$$

Polynomial Regression

System of normal equations for polynomials

$$\begin{aligned} na_0 + \left(\sum_{i=1}^n x_i \right) a_1 + \dots + \left(\sum_{i=1}^n x_i^m \right) a_m &= \sum_{i=1}^n y_i \\ \left(\sum_{i=1}^n x_i \right) a_0 + \left(\sum_{i=1}^n x_i^2 \right) a_1 + \dots + \left(\sum_{i=1}^n x_i^{m+1} \right) a_m &= \sum_{i=1}^n x_i y_i \\ \vdots &\quad \vdots \\ \left(\sum_{i=1}^n x_i^m \right) a_0 + \left(\sum_{i=1}^n x_i^{m+1} \right) a_1 + \dots + \left(\sum_{i=1}^n x_i^{2m} \right) a_m &= \sum_{i=1}^n x_i^m y_i, \end{aligned}$$

- $m + 1$ linear equations for $m + 1$ unknowns a_0, \dots, a_m .
- System can be solved with standard methods from linear algebra.
- Solution is unique unless the points lie exactly on a polynomial of lower degree.

Multilinear Regression

Generalization to more than one argument

$$z = f(x, y) = a + bx + cy$$

Approach: Minimize the sum of squared errors, that is,

$$F(a, b, c) = \sum_{i=1}^n (f(x_i, y_i) - z_i)^2 = \sum_{i=1}^n (a + bx_i + cy_i - z_i)^2$$

Necessary conditions for a minimum: All partial derivatives vanish, that is,

$$\begin{aligned}\frac{\partial F}{\partial a} &= \sum_{i=1}^n 2(a + bx_i + cy_i - z_i) = 0, \\ \frac{\partial F}{\partial b} &= \sum_{i=1}^n 2(a + bx_i + cy_i - z_i)x_i = 0, \\ \frac{\partial F}{\partial c} &= \sum_{i=1}^n 2(a + bx_i + cy_i - z_i)y_i = 0.\end{aligned}$$

Multilinear Regression

System of normal equations for several arguments

$$na + \left(\sum_{i=1}^n x_i \right) b + \left(\sum_{i=1}^n y_i \right) c = \sum_{i=1}^n z_i$$

$$\left(\sum_{i=1}^n x_i \right) a + \left(\sum_{i=1}^n x_i^2 \right) b + \left(\sum_{i=1}^n x_i y_i \right) c = \sum_{i=1}^n z_i x_i$$

$$\left(\sum_{i=1}^n y_i \right) a + \left(\sum_{i=1}^n x_i y_i \right) b + \left(\sum_{i=1}^n y_i^2 \right) c = \sum_{i=1}^n z_i y_i$$

- 3 linear equations for 3 unknowns a , b , and c .
- System can be solved with standard methods from linear algebra.
- Solution is unique unless all data points lie on a straight line.

Multilinear Regression

General multilinear case:

$$y = f(x_1, \dots, x_m) = a_0 + \sum_{k=1}^m a_k x_k$$

Approach: Minimize the sum of squared errors, that is,

$$F(\vec{a}) = (\mathbf{X}\vec{a} - \vec{y})^\top (\mathbf{X}\vec{a} - \vec{y}),$$

where

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{nm} \end{pmatrix}, \quad \vec{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad \text{and} \quad \vec{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{pmatrix}$$

Necessary condition for a minimum:

$$\vec{\nabla}_{\vec{a}} F(\vec{a}) = \vec{\nabla}_{\vec{a}} (\mathbf{X}\vec{a} - \vec{y})^\top (\mathbf{X}\vec{a} - \vec{y}) = \vec{0}$$

Multilinear Regression

- $\nabla_{\vec{a}} F(\vec{a})$ may easily be computed by remembering that the differential operator

$$\vec{\nabla}_{\vec{a}} = \left(\frac{\partial}{\partial a_0}, \dots, \frac{\partial}{\partial a_m} \right)$$

behaves formally like a vector that is “multiplied” to the sum of squared errors.

- Alternatively, one may write out the differentiation componentwise.

With the former method we obtain for the derivative:

$$\begin{aligned}\vec{\nabla}_{\vec{a}} F(\vec{a}) &= \vec{\nabla}_{\vec{a}} ((\mathbf{X}\vec{a} - \vec{y})^\top (\mathbf{X}\vec{a} - \vec{y})) \\ &= (\vec{\nabla}_{\vec{a}} (\mathbf{X}\vec{a} - \vec{y}))^\top (\mathbf{X}\vec{a} - \vec{y}) + ((\mathbf{X}\vec{a} - \vec{y})^\top (\vec{\nabla}_{\vec{a}} (\mathbf{X}\vec{a} - \vec{y})))^\top \\ &= (\vec{\nabla}_{\vec{a}} (\mathbf{X}\vec{a} - \vec{y}))^\top (\mathbf{X}\vec{a} - \vec{y}) + (\vec{\nabla}_{\vec{a}} (\mathbf{X}\vec{a} - \vec{y}))^\top (\mathbf{X}\vec{a} - \vec{y}) \\ &= 2\mathbf{X}^\top (\mathbf{X}\vec{a} - \vec{y}) \\ &= 2\mathbf{X}^\top \mathbf{X}\vec{a} - 2\mathbf{X}^\top \vec{y} = \vec{0}\end{aligned}$$

Multilinear Regression

Necessary condition for a minimum therefore:

$$\begin{aligned}\vec{\nabla}_{\vec{a}} F(\vec{a}) &= \vec{\nabla}_{\vec{a}} (\mathbf{X}\vec{a} - \vec{y})^\top (\mathbf{X}\vec{a} - \vec{y}) \\ &= 2\mathbf{X}^\top \mathbf{X}\vec{a} - 2\mathbf{X}^\top \vec{y} \stackrel{!}{=} \vec{0}\end{aligned}$$

As a consequence we obtain the system of **normal equations**:

$$\mathbf{X}^\top \mathbf{X}\vec{a} = \mathbf{X}^\top \vec{y}$$

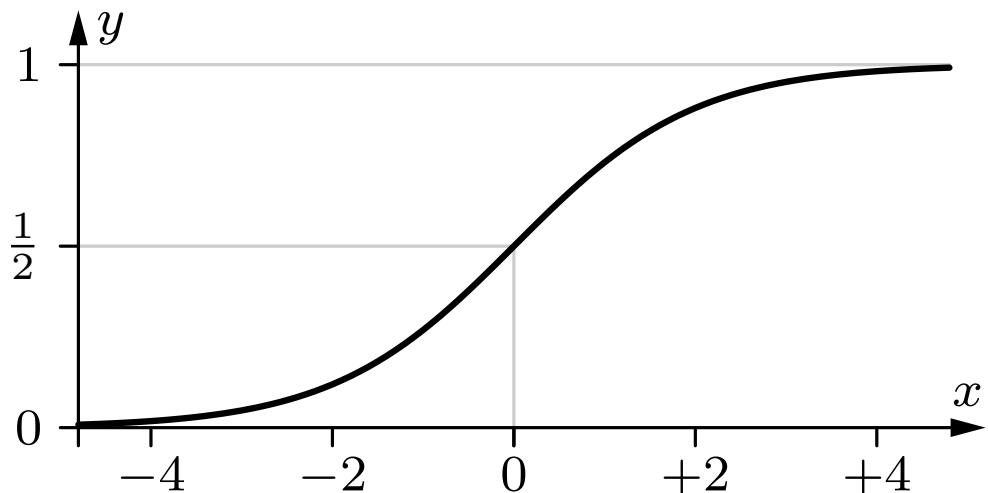
This system has a solution unless $\mathbf{X}^\top \mathbf{X}$ is singular. If it is regular, we have

$$\vec{a} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \vec{y}.$$

$(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ is called the (Moore-Penrose-) **Pseudoinverse** of the matrix \mathbf{X} .

With the matrix-vector representation of the regression problem
an extension to **multipolynomial regression** is straightforward:
Simply add the desired products of powers (monomials) to the matrix \mathbf{X} .

Mathematical Background: Logistic Function



Logistic Function:

$$y = f(x) = \frac{Y}{1 + e^{-a(x-x_0)}}$$

Special case $Y = a = 1, x_0 = 0$:

$$y = f(x) = \frac{1}{1 + e^{-x}}$$

Application areas of the logistic function:

- Can be used to describe **saturation processes** (growth processes with finite capacity/finite resources Y).

Derivation e.g. from a **Bernoulli differential equation**

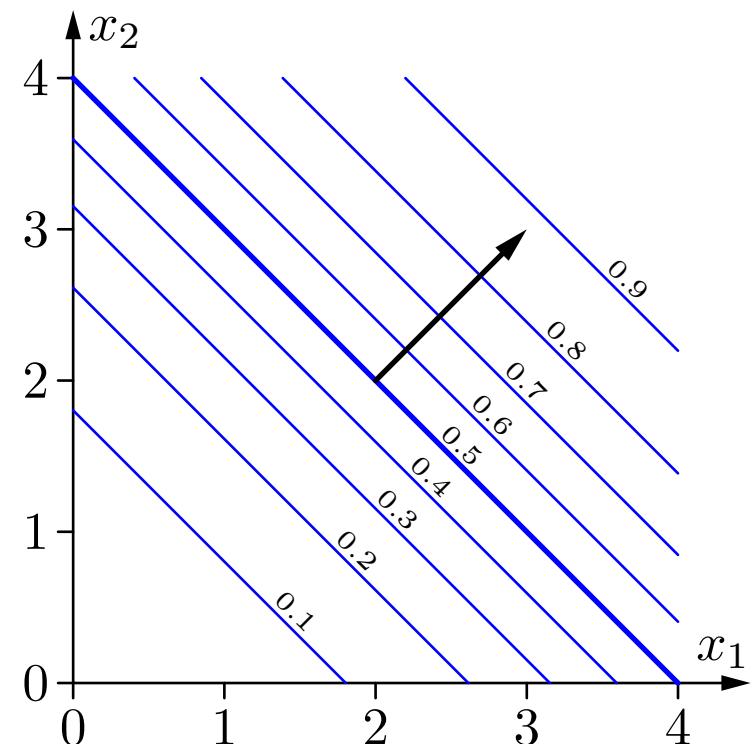
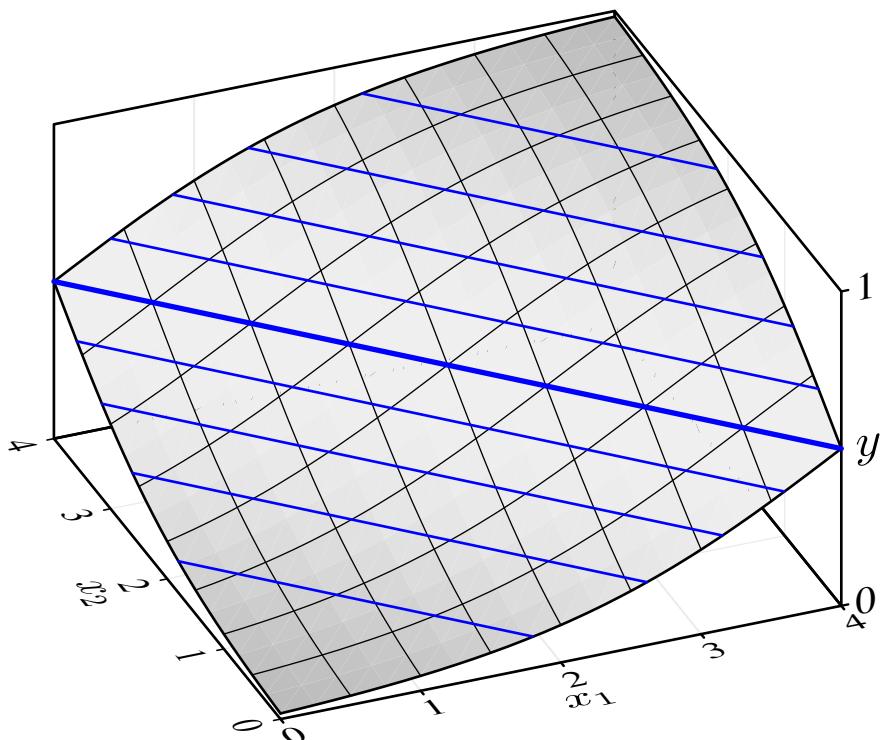
$$f'(x) = k \cdot f(x) \cdot (Y - f(x)) \quad (\text{yields } a = kY)$$

- Can be used to describe a **linear classifier** (especially for two-class problems, considered later).

Mathematical Background: Logistic Function

Example: two-dimensional logistic function

$$y = f(\vec{x}) = \frac{1}{1 + \exp(-(x_1 + x_2 - 4))} = \frac{1}{1 + \exp(-((1, 1)^\top (x_1, x_2) - 4))}$$

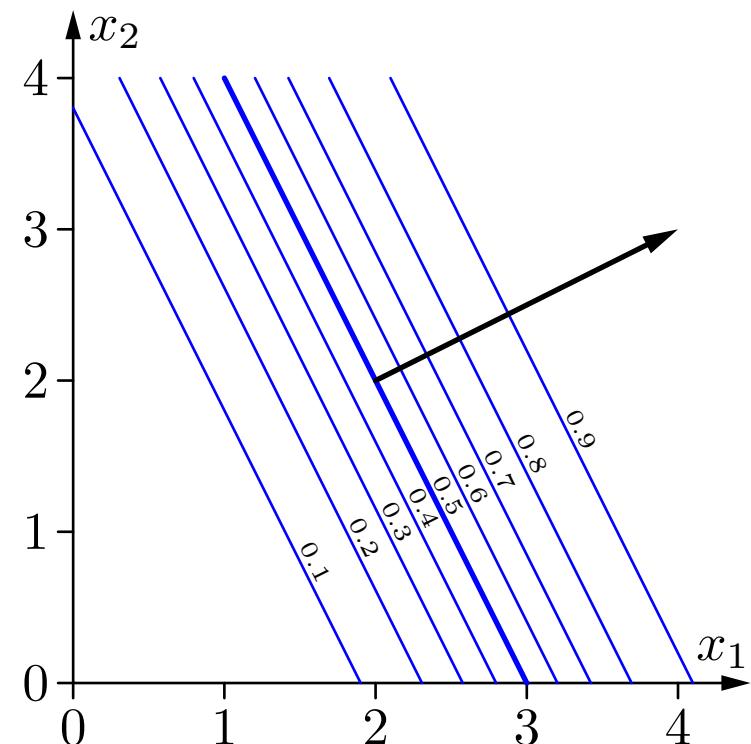
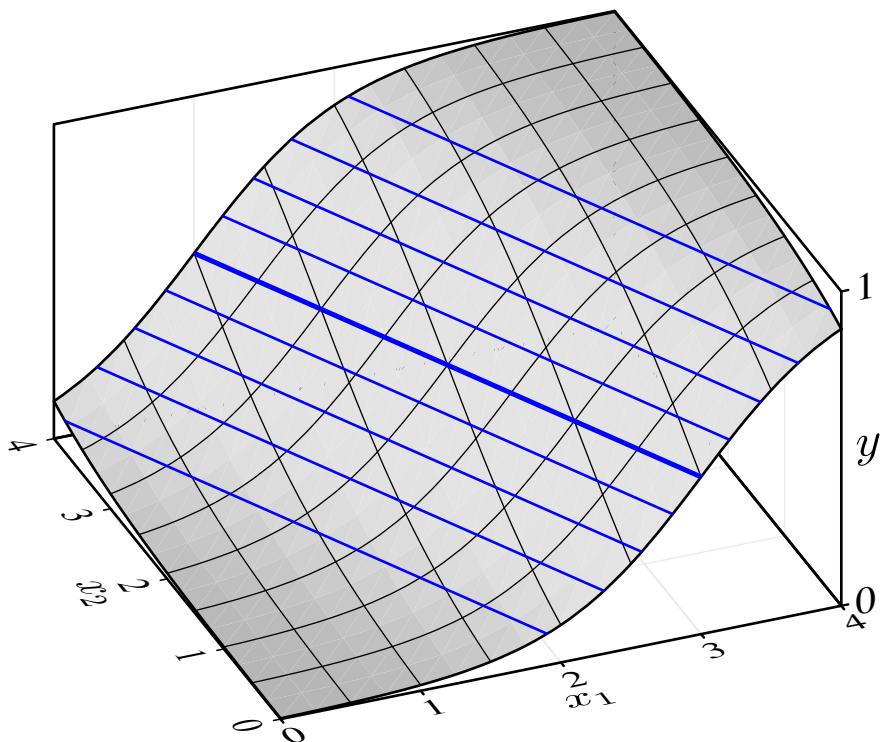


The “contour lines” of the logistic function are parallel lines/hyperplanes.

Mathematical Background: Logistic Function

Example: two-dimensional logistic function

$$y = f(\vec{x}) = \frac{1}{1 + \exp(-(2x_1 + x_2 - 6))} = \frac{1}{1 + \exp(-((2, 1)^\top (x_1, x_2) - 6))}$$



The “contour lines” of the logistic function are parallel lines/hyperplanes.

Mathematical Background: Logistic Regression

Generalization of regression to non-polynomial functions.

Simple example: $y = ax^b$

Idea: Find a **transformation to the linear/polynomial case**.

Transformation for the above example: $\ln y = \ln a + b \cdot \ln x$.

\Rightarrow Linear regression for the transformed data $y' = \ln y$ and $x' = \ln x$.

Special case: **Logistic Function** (mit $a_0 = \vec{a}^\top \vec{x}_0$)

$$y = \frac{Y}{1 + e^{-(\vec{a}^\top \vec{x} + a_0)}} \Leftrightarrow \frac{1}{y} = \frac{1 + e^{-(\vec{a}^\top \vec{x} + a_0)}}{Y} \Leftrightarrow \frac{Y - y}{y} = e^{-(\vec{a}^\top \vec{x} + a_0)}.$$

Result: Apply so-called **Logit Transform**

$$z = \ln \left(\frac{y}{Y - y} \right) = \vec{a}^\top \vec{x} + a_0.$$

Logistic Regression: Example

Data points:

x	1	2	3	4	5
y	0.4	1.0	3.0	5.0	5.6

Apply the logit transform

$$z = \ln\left(\frac{y}{Y-y}\right), \quad Y = 6.$$

Transformed data points:

(for linear regression)

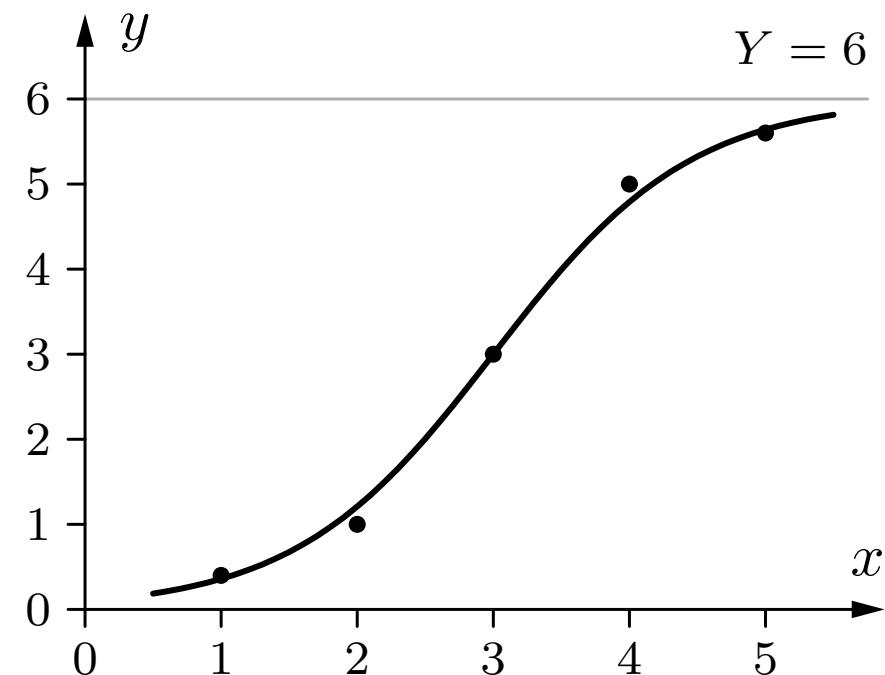
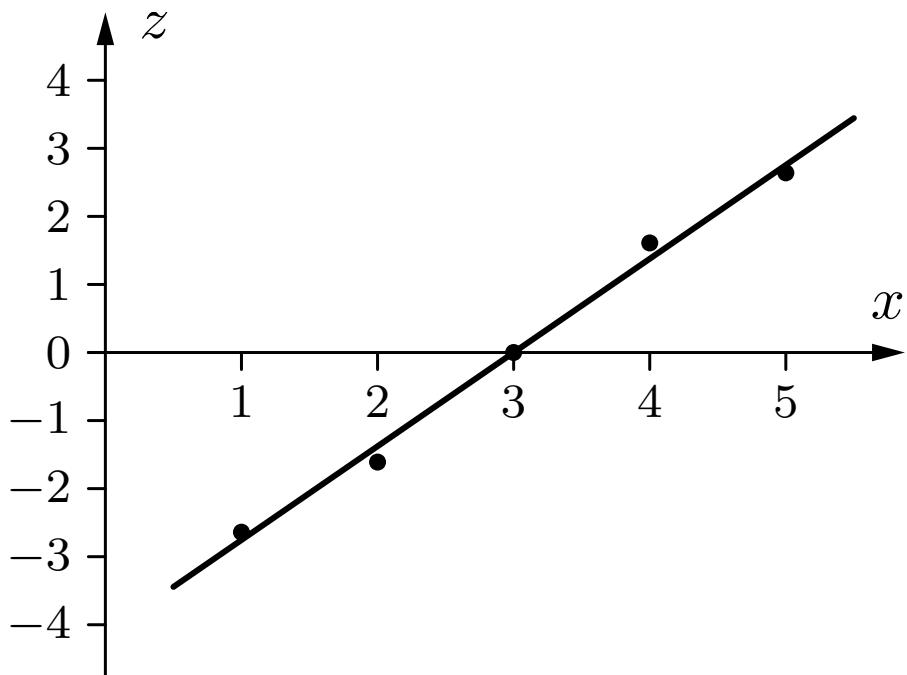
x	1	2	3	4	5
z	-2.64	-1.61	0.00	1.61	2.64

The resulting regression line and therefore the desired function are

$$z \approx 1.3775x - 4.133 \quad \text{and} \quad y \approx \frac{6}{1 + e^{-(1.3775x - 4.133)}} \approx \frac{6}{1 + e^{-1.3775(x-3)}}.$$

Attention: Note that the error is minimized only in the transformed space!
Therefore the function in the original space may not be optimal!

Logistic Regression: Example

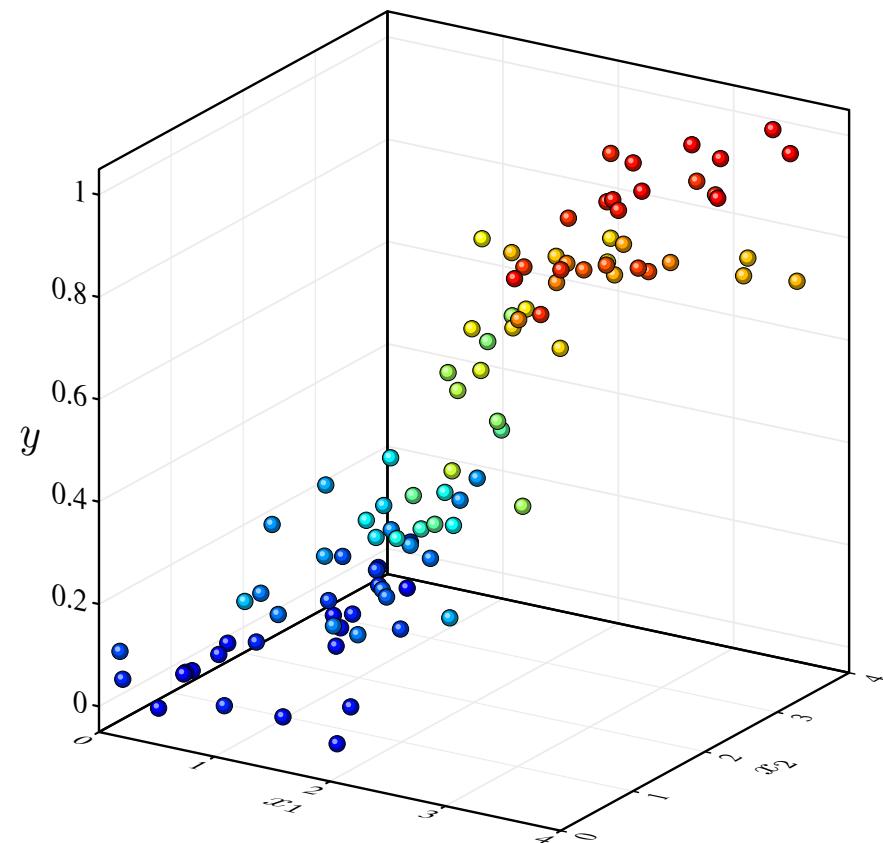
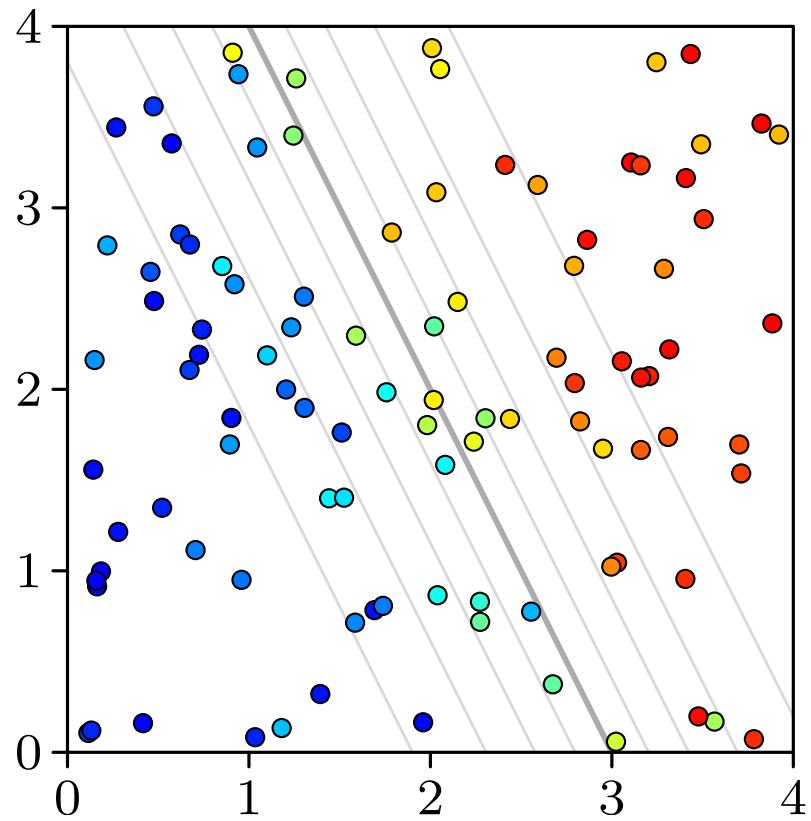


The resulting regression line and therefore the desired function are

$$z \approx 1.3775x - 4.133 \quad \text{and} \quad y \approx \frac{6}{1 + e^{-(1.3775x - 4.133)}} \approx \frac{6}{1 + e^{-1.3775(x-3)}}.$$

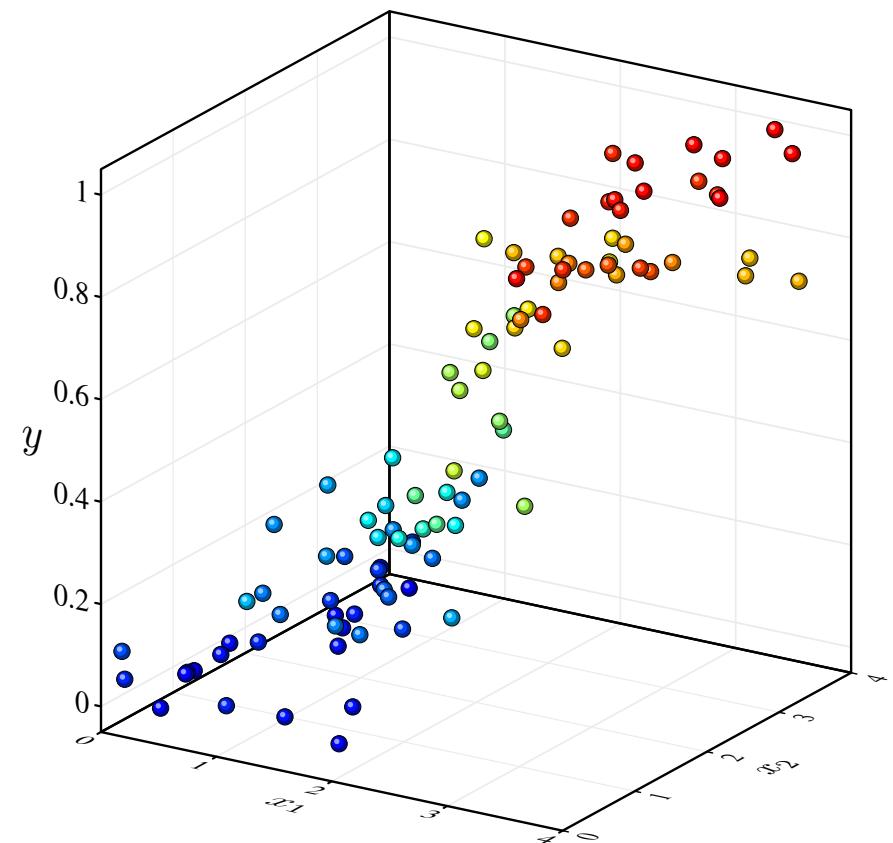
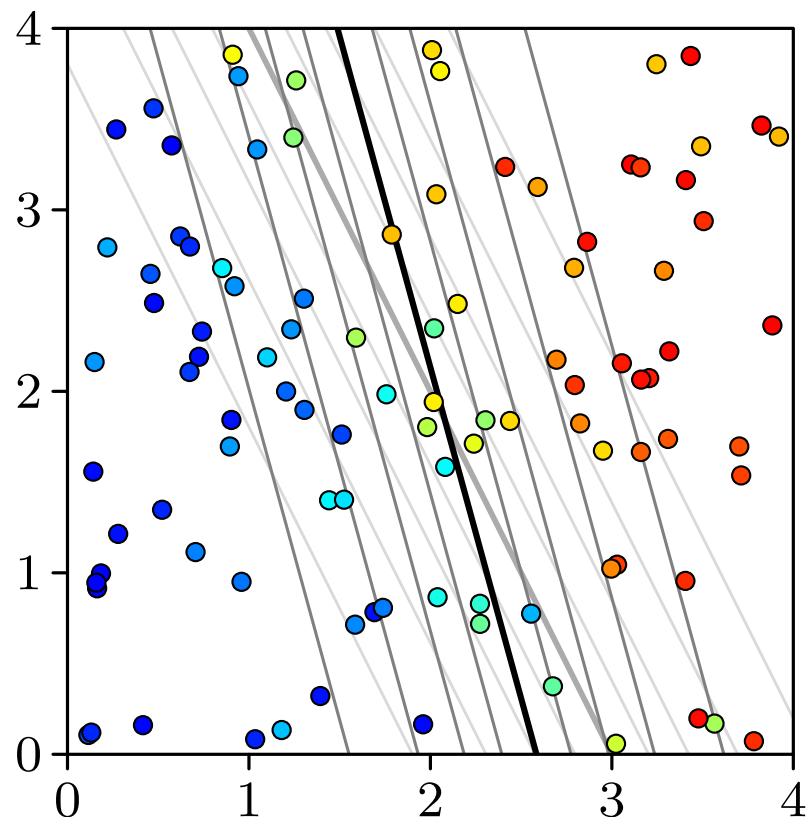
Attention: Note that the error is minimized only in the transformed space!
Therefore the function in the original space may not be optimal!

Multivariate Logistic Regression: Example



- Example data were drawn from a logistic function and noise was added.
(The gray “contour lines” show the ideal logistic function.)
- Reconstructing the logistic function can be reduced to a multilinear regression by applying a logit transform to the y -values of the data points.

Multivariate Logistic Regression: Example



- The black “contour lines” show the resulting logistic function.
Is the deviation from the ideal logistic function (gray) caused by the added noise?
- **Attention:** Note that the error is minimized only in the transformed space!
Therefore the function in the original space may not be optimal!

Logistic Regression: Optimization in Original Space

Approach analogous to linear/polynomial regression

Given: data set $\mathbf{D} = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$ with n data points, $y_i \in (0, 1)$.

Simplification: Use $\vec{x}_i^* = (1, x_{i1}, \dots, x_{im})^\top$ und $\vec{a} = (a_0, a_1, \dots, a_m)^\top$.
(By the leading 1 in \vec{x}_i^* the constant a_0 is captured.)

Minimize sum of squared errors / deviations:

$$F(\vec{a}) = \sum_{i=1}^n \left(y_i - \frac{1}{1 + e^{-\vec{a}^\top \vec{x}_i^*}} \right)^2 \stackrel{!}{=} \min.$$

Necessary condition for a minimum:

Gradient of the objective function $F(\vec{a})$ w.r.t. \vec{a} vanishes: $\vec{\nabla}_{\vec{a}} F(\vec{a}) \stackrel{!}{=} \vec{0}$

Problem: The resulting equation system is not linear.

Solution possibilities:

- Gradient descent on objective function $F(\vec{a})$.
- Root search on gradient $\vec{\nabla}_{\vec{a}} F(\vec{a})$. (e.g. Newton–Raphson method)

Reminder: Gradient Methods for Optimization

The **gradient** is a differential operator, that turns a scalar function into a vector field.

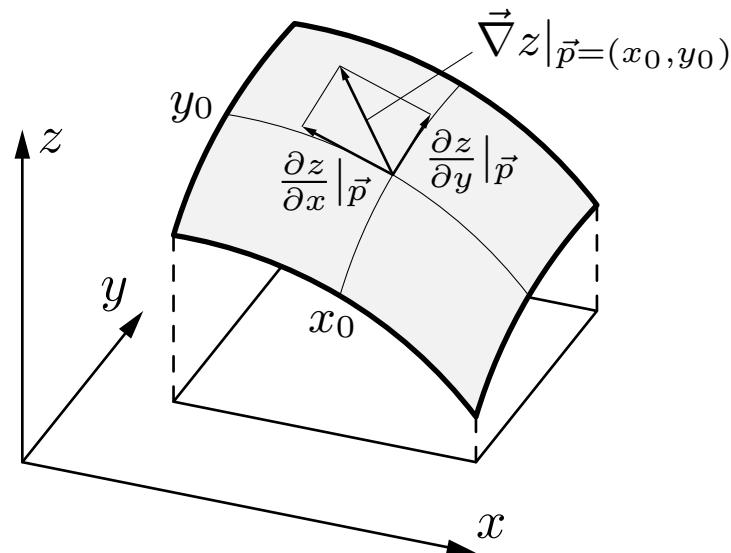


Illustration of the gradient of a real-valued function $z = f(x, y)$ at a point $\vec{p} = (x_0, y_0)$.

$$\text{It is } \vec{\nabla}z|_{(x_0,y_0)} = \left(\frac{\partial z}{\partial x}|_{x_0}, \frac{\partial z}{\partial y}|_{y_0} \right).$$

The gradient at a point shows the direction of the steepest ascent of the function at this point; its length describes the steepness of the ascent.

Principle of gradient methods:

Starting at a (possibly randomly chosen) initial point, make (small) steps in (or against) the direction of the gradient of the objective function at the current point, until a maximum (or a minimum) has been reached.

Gradient Methods: Cookbook Recipe

Idea: Starting from a randomly chosen point in the search space, make small steps in the search space, always in the direction of the steepest ascent (or descent) of the function to optimize, until a (local) maximum (or minimum) is reached.

1. Choose a (random) starting point $\vec{x}^{(0)} = (x_1^{(0)}, \dots, x_n^{(0)})^\top$
2. Compute the gradient of the objective function f at the current point $\vec{x}^{(i)}$:

$$\nabla_{\vec{x}} f(\vec{x}) \Big|_{\vec{x}^{(i)}} = \left(\frac{\partial}{\partial x_1} f(\vec{x}) \Big|_{x_1^{(i)}}, \dots, \frac{\partial}{\partial x_n} f(\vec{x}) \Big|_{x_n^{(i)}} \right)^\top$$

3. Make a small step in the direction (or against the direction) of the gradient:

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} \pm \eta \nabla_{\vec{x}} f(\vec{x}^{(i)}). \quad \begin{array}{ll} + & : \text{gradient ascent} \\ - & : \text{gradient descent} \end{array}$$

η is a step width parameter (“learning rate” in artificial neuronal networks)

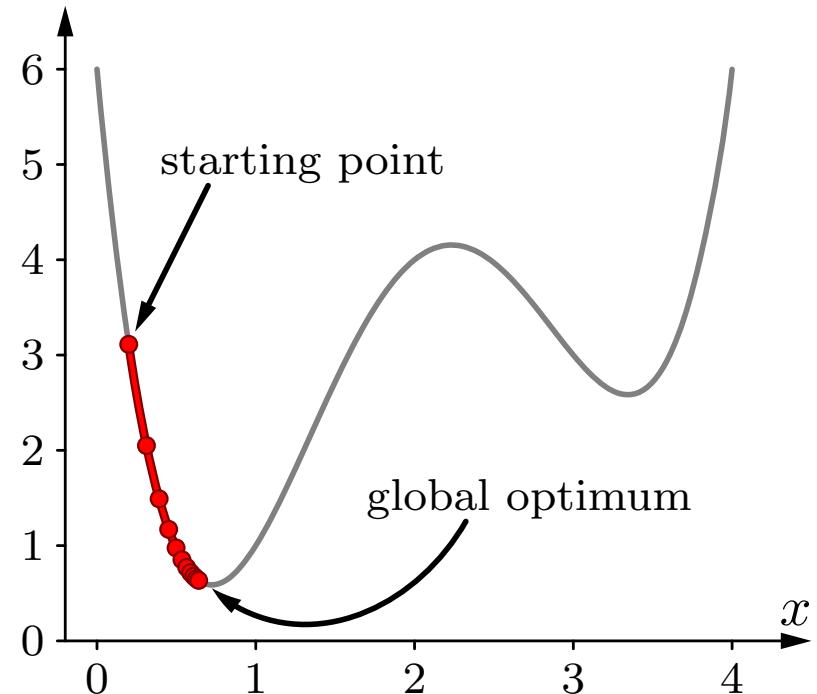
4. Repeat steps 2 and 3, until some termination criterion is satisfied.
(e.g., a certain number of steps has been executed, current gradient is small)

Gradient Descent: Simple Example

Example function:

$$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	0.200	3.112	-11.147	0.111
1	0.311	2.050	-7.999	0.080
2	0.391	1.491	-6.015	0.060
3	0.451	1.171	-4.667	0.047
4	0.498	0.976	-3.704	0.037
5	0.535	0.852	-2.990	0.030
6	0.565	0.771	-2.444	0.024
7	0.589	0.716	-2.019	0.020
8	0.610	0.679	-1.681	0.017
9	0.626	0.653	-1.409	0.014
10	0.640	0.635		



Gradient descent with initial value 0.2 and step width/learning rate 0.01.

Due to a proper width/learning rate, the minimum is approached fairly quickly.

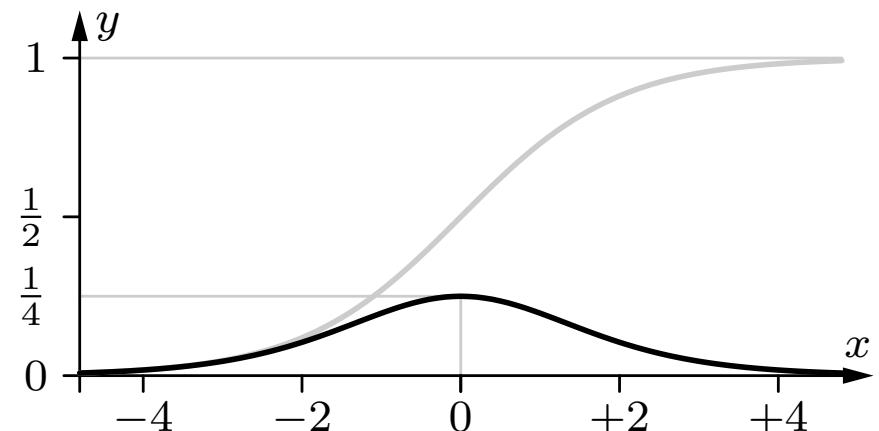
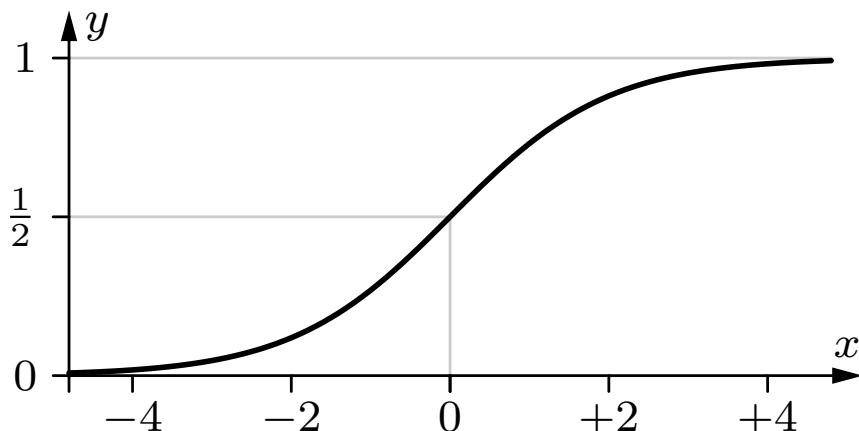
Logistic Regression: Gradient Descent

With the abbreviation $f(z) = \frac{1}{1+e^{-z}}$ for the logistic function it is

$$\vec{\nabla}_{\vec{a}} F(\vec{a}) = \vec{\nabla}_{\vec{a}} \sum_{i=1}^n (y_i - f(\vec{a}^\top \vec{x}_i^*))^2 = -2 \sum_{i=1}^n (y_i - f(\vec{a}^\top \vec{x}_i^*)) \cdot f'(\vec{a}^\top \vec{x}_i^*) \cdot \vec{x}_i^*.$$

Derivative of the logistic function: (cf. Bernoulli differential equation)

$$\begin{aligned} f'(z) &= \frac{d}{dz} (1 + e^{-z})^{-1} = -(1 + e^{-z})^{-2} (-e^{-z}) \\ &= \frac{1 + e^{-z} - 1}{(1 + e^{-z})^2} = \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}}\right) = f(z) \cdot (1 - f(z)), \end{aligned}$$



Logistic Regression: Gradient Descent

Given: data set $\mathbf{D} = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$ with n data points, $y_i \in (0, 1)$.

Simplification: Use $\vec{x}_i^* = (1, x_{i1}, \dots, x_{im})^\top$ and $\vec{a} = (a_0, a_1, \dots, a_m)^\top$.

Gradient descent on the objective function $F(\vec{a})$:

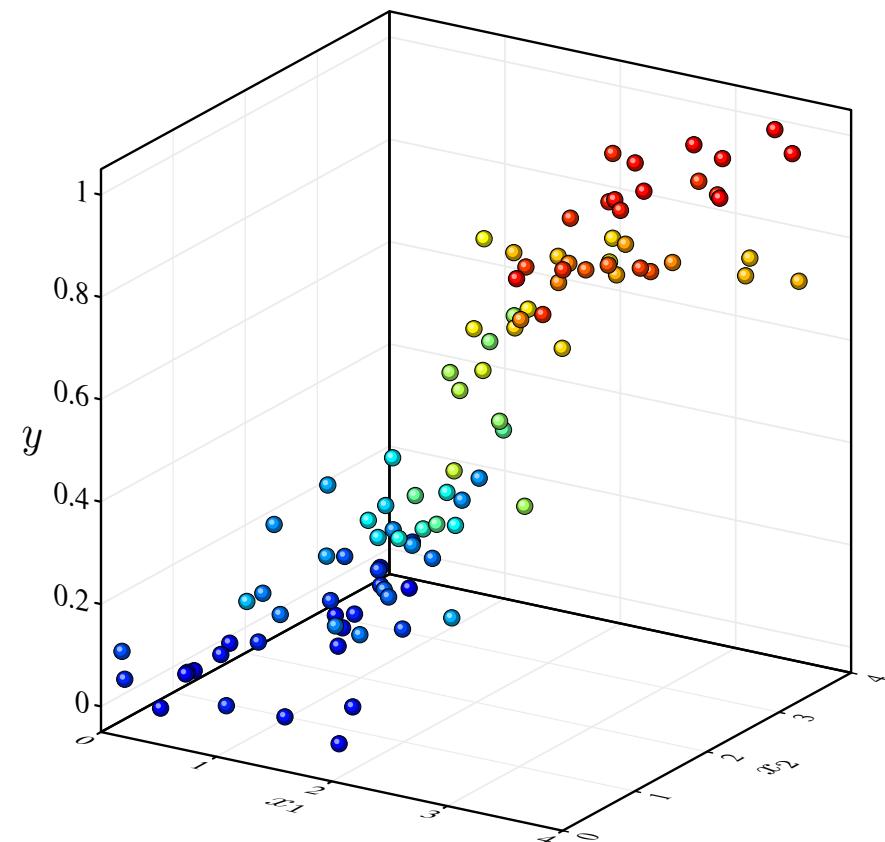
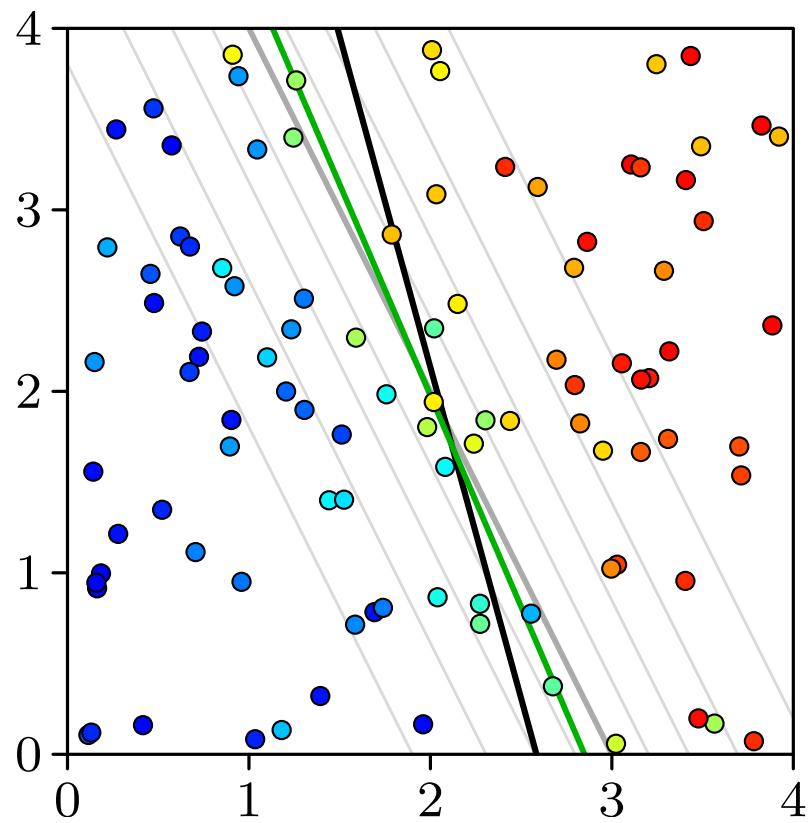
- Choose as the initial point \vec{a}_0 the result of a logit transform and a linear regression (or merely a linear regression).
- Update of the parameters \vec{a} :

$$\begin{aligned}\vec{a}_{t+1} &= \vec{a}_t - \frac{\eta}{2} \cdot \vec{\nabla}_{\vec{a}} F(\vec{a})|_{\vec{a}_t} \\ &= \vec{a}_t + \eta \cdot \sum_{i=1}^n (y_i - f(\vec{a}_t^\top \vec{x}_i^*)) \cdot f(\vec{a}_t^\top \vec{x}_i^*) \cdot (1 - f(\vec{a}_t^\top \vec{x}_i^*)) \cdot \vec{x}_i^*,\end{aligned}$$

where η is a step width parameter to be chosen by a user (e.g. $\eta = 0.05$) (in the area of artificial neural networks also called “learning rate”).

- Repeat the update step until convergence, e.g. until $\|\vec{a}_{t+1} - \vec{a}_t\| < \tau$ with a chosen threshold τ (z.B. $\tau = 10^{-6}$).

Multivariate Logistic Regression: Example



- Black “contour line”: logit transform and linear regression.
- Green “contour line”: gradient descent on error function in original space.

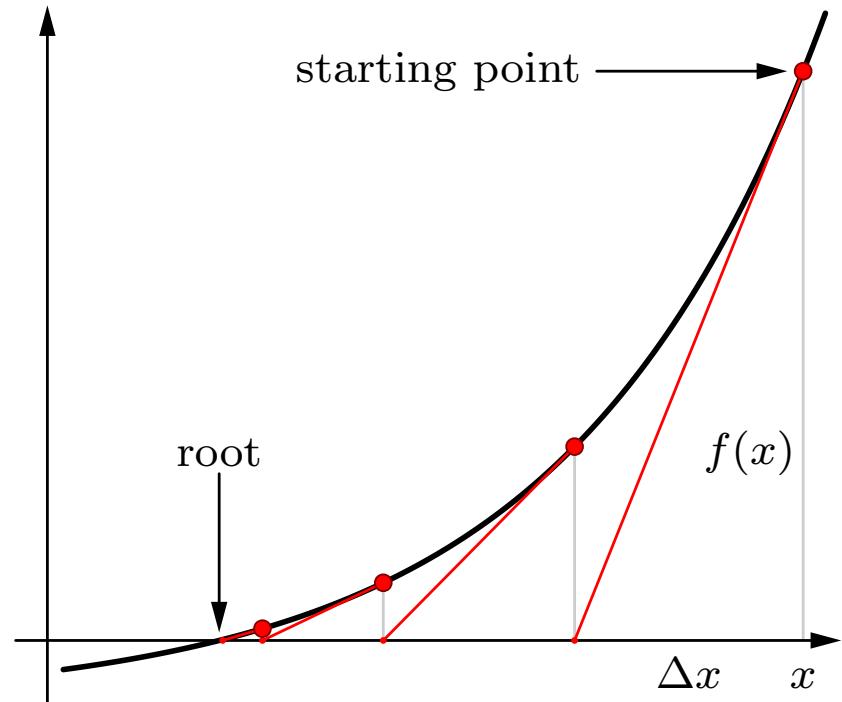
(For simplicity and clarity only the “contour lines” for $y = 0.5$ (inflection lines) are shown.)

Reminder: Newton–Raphson Method

- The Newton–Raphson method is an iterative numeric algorithm to approximate a root of a function.
- Idea: use slope/direction of a tangent to the function at a current point to find the next approximation.

- Formally:

$$\vec{x}_{t+1} = \vec{x}_t + \Delta \vec{x}_t; \quad \Delta \vec{x}_t = -\frac{f(\vec{x}_t)}{\nabla_{\vec{x}} f(\vec{x})|_{\vec{x}_t}}$$



- The gradient describes the direction of steepest ascent of tangent (hyper-)planes to the function (in one dimension: the slope of tangents to the function).
- In one dimension (see diagram):

Solve $(x_{t+1}, 0) = (x_t, f(x_t))^T + k \cdot (1, \frac{d}{dx} f(x)|_{x_t})^T$, $k \in \mathbb{R}$, for x_{t+1} .

Since $0 = f(x_t) + k \cdot \frac{d}{dx} f(x)|_{x_t}$, it is $k = -f(x_t)/\frac{d}{dx} f(x)|_{x_t}$.

Newton–Raphson Method for Finding Optima

- The standard Newton-Raphson method finds roots of functions.
- By applying it to the gradient of a function, it may be used to find optima (minima, maxima, or saddle points), because a vanishing gradient is a necessary condition for an optimum.
- In this case the update formula is

$$\vec{x}_{t+1} = \vec{x}_t + \Delta \vec{x}_t, \quad \Delta \vec{x}_t = - \left(\vec{\nabla}_{\vec{x}}^2 f(\vec{x}) \Big|_{\vec{x}_t} \right)^{-1} \cdot \vec{\nabla}_{\vec{x}} f(\vec{x}) \Big|_{\vec{x}_t},$$

where $\vec{\nabla}_{\vec{x}}^2 f(\vec{x})$ is the so-called **Hessian matrix**, that is, the matrix of second-order partial derivatives of the scalar-valued function f .

- In one dimension:

$$x_{t+1} = x_t + \Delta x_t, \quad \Delta x_t = - \frac{\frac{\partial f(x)}{\partial x} \Big|_{x_t}}{\frac{\partial^2 f(x)}{\partial x^2} \Big|_{x_t}}.$$

- The Newton–Raphson method usually converges much faster than gradient descent (... and needs no step width parameter!).

Logistic Regression: Newton–Raphson

With the abbreviation $f(z) = \frac{1}{1+e^{-z}}$ for the logistic function it is

$$\begin{aligned}\vec{\nabla}_{\vec{a}}^2 F(\vec{a}) &= \vec{\nabla}_{\vec{a}} \left(-2 \sum_{i=1}^n (y_i - f(\vec{a}^\top \vec{x}_i^*)) \cdot f(\vec{a}^\top \vec{x}_i^*) \cdot (1 - f(\vec{a}^\top \vec{x}_i^*)) \cdot \vec{x}_i^* \right) \\ &= -2 \vec{\nabla}_{\vec{a}} \sum_{i=1}^n \left(y_i f(\vec{a}^\top \vec{x}_i^*) - (y_i + 1) f^2(\vec{a}^\top \vec{x}_i^*) + f^3(\vec{a}^\top \vec{x}_i^*) \right) \cdot \vec{x}_i^* \\ &= -2 \sum_{i=1}^n \left(y_i - 2(y_i + 1)f(\vec{a}^\top \vec{x}_i^*) + 3f^2(\vec{a}^\top \vec{x}_i^*) \right) \cdot f'(\vec{a}^\top \vec{x}_i^*) \cdot \vec{x}_i^* \vec{x}_i^{*\top}\end{aligned}$$

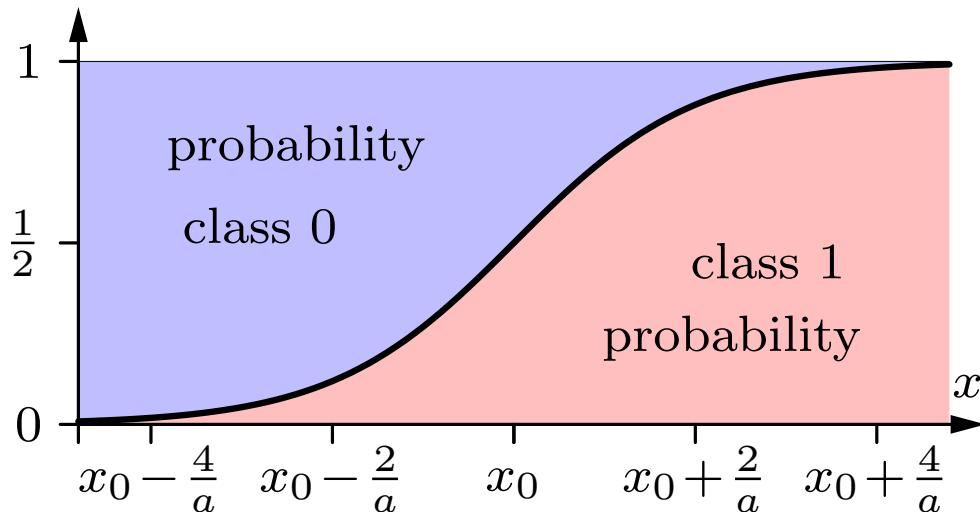
where again $f'(z) = f(z) \cdot (1 - f(z))$ (as derived above).

Thus we get for the update of the parameters \vec{a} : (note: no step width η)

$$\vec{a}_{t+1} = \vec{a}_t - \left(\vec{\nabla}_{\vec{a}}^2 F(\vec{a}) \Big|_{\vec{a}_t} \right)^{-1} \cdot \vec{\nabla}_{\vec{a}} F(\vec{a}) \Big|_{\vec{a}_t}$$

with $\vec{\nabla}_{\vec{a}}^2 F(\vec{a}) \Big|_{\vec{a}_t}$ as shown above and $\vec{\nabla}_{\vec{a}} F(\vec{a}) \Big|_{\vec{a}_t}$ as the expression in large parentheses.

Logistic Classification: Two Classes



Logistic function with $Y = 1$:

$$y = f(x) = \frac{1}{1 + e^{-a(x-x_0)}}$$

Interpret the logistic function as the probability of one class.

- Conditional class probability is logistic function:

$$P(C = c_1 \mid \vec{X} = \vec{x}) = p_1(\vec{x}) = p(\vec{x}; \vec{a}) = \frac{1}{1 + e^{-\vec{a}^\top \vec{x}^*}}.$$

$$\begin{aligned}\vec{a} &= (a_0, x_1, \dots, x_m)^\top \\ \vec{x}^* &= (1, x_1, \dots, x_m)^\top\end{aligned}$$

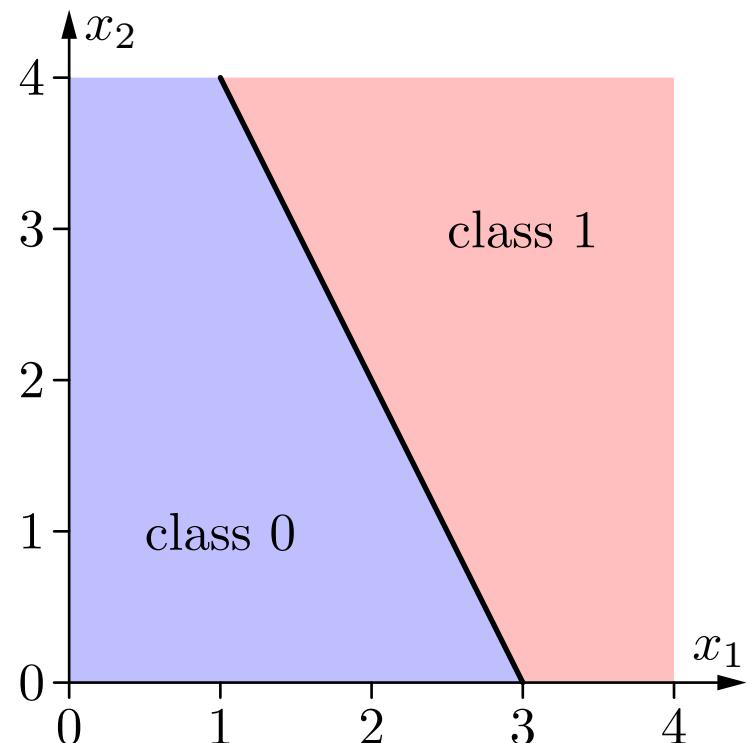
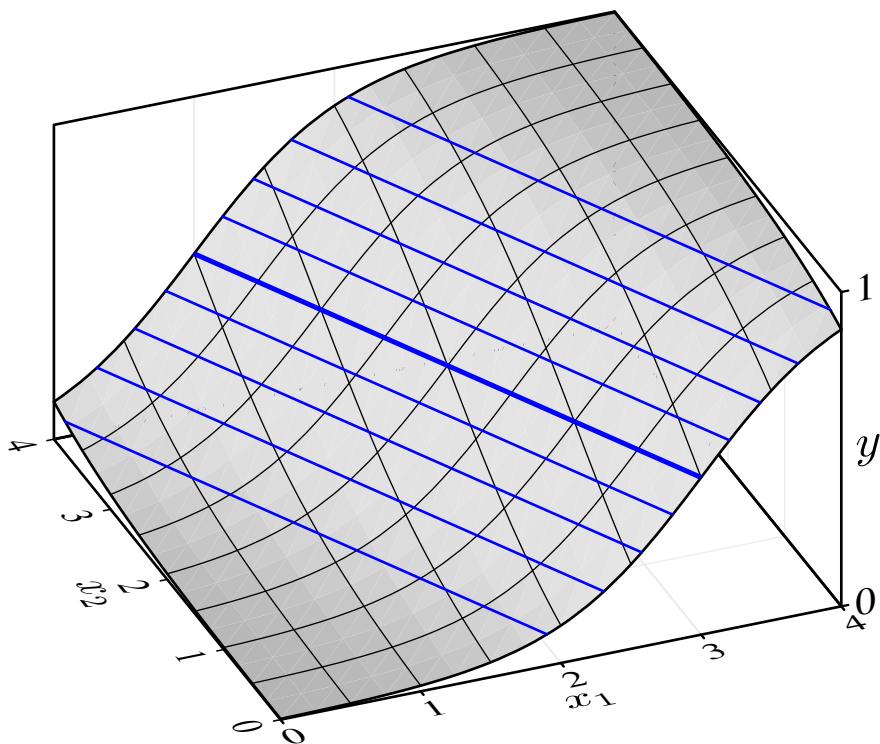
- With only two classes the conditional probability of the other class is:

$$P(C = c_0 \mid \vec{X} = \vec{x}) = p_1(\vec{x}) = 1 - p(\vec{x}; \vec{a}).$$

- Classification rule:

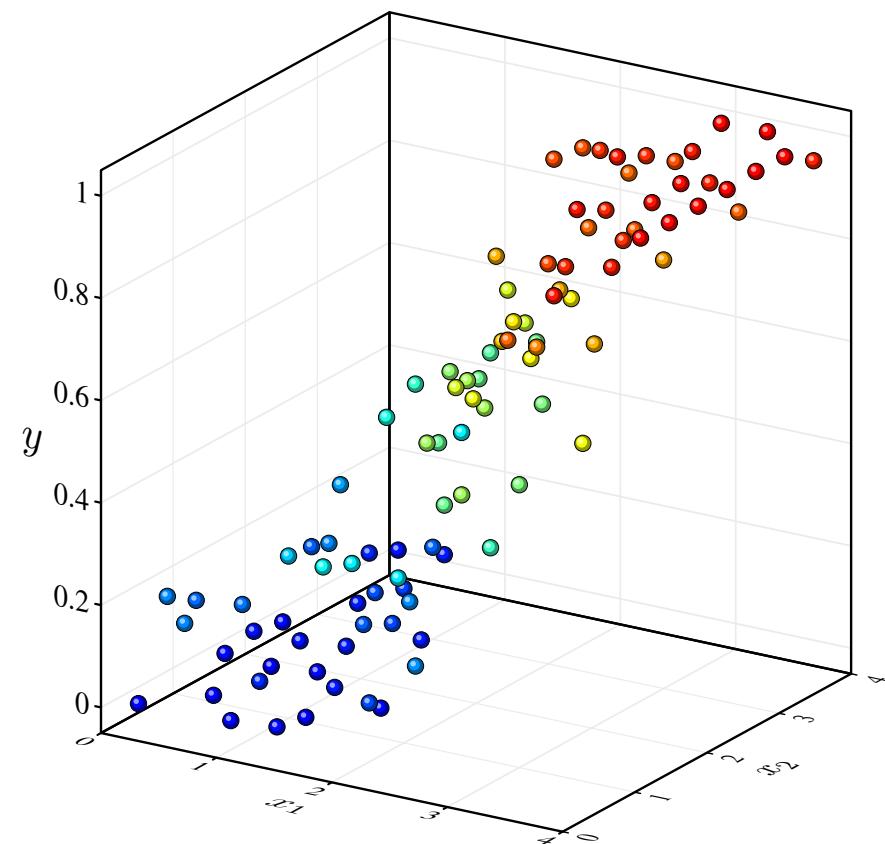
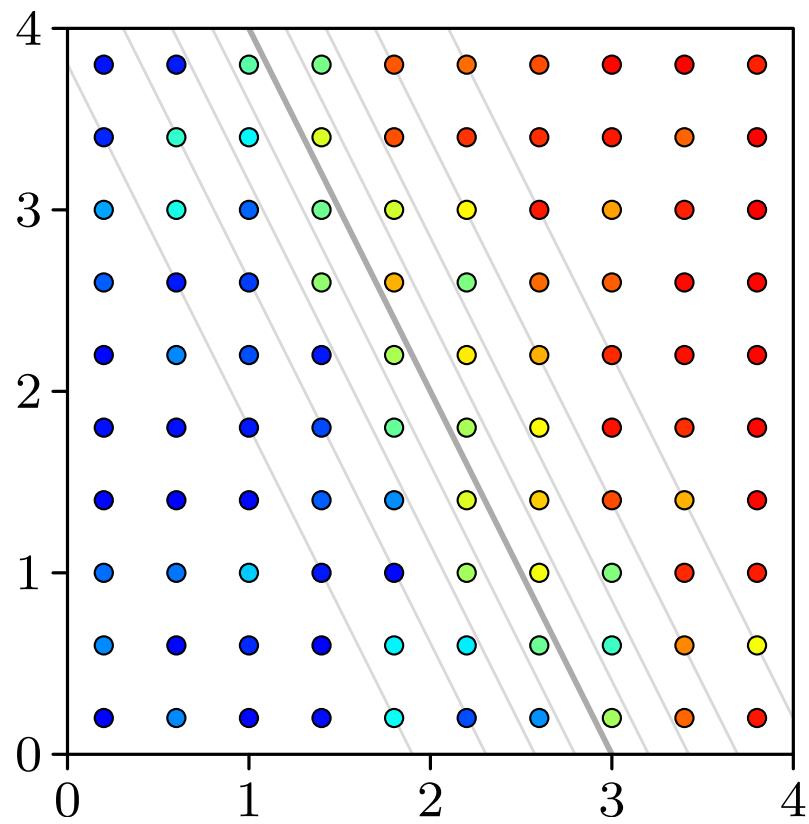
$$C = \begin{cases} c_1, & \text{if } p(\vec{x}; \vec{a}) \geq \theta, \\ c_0, & \text{if } p(\vec{x}; \vec{a}) < \theta, \end{cases} \quad \theta = 0.5.$$

Logistic Classification



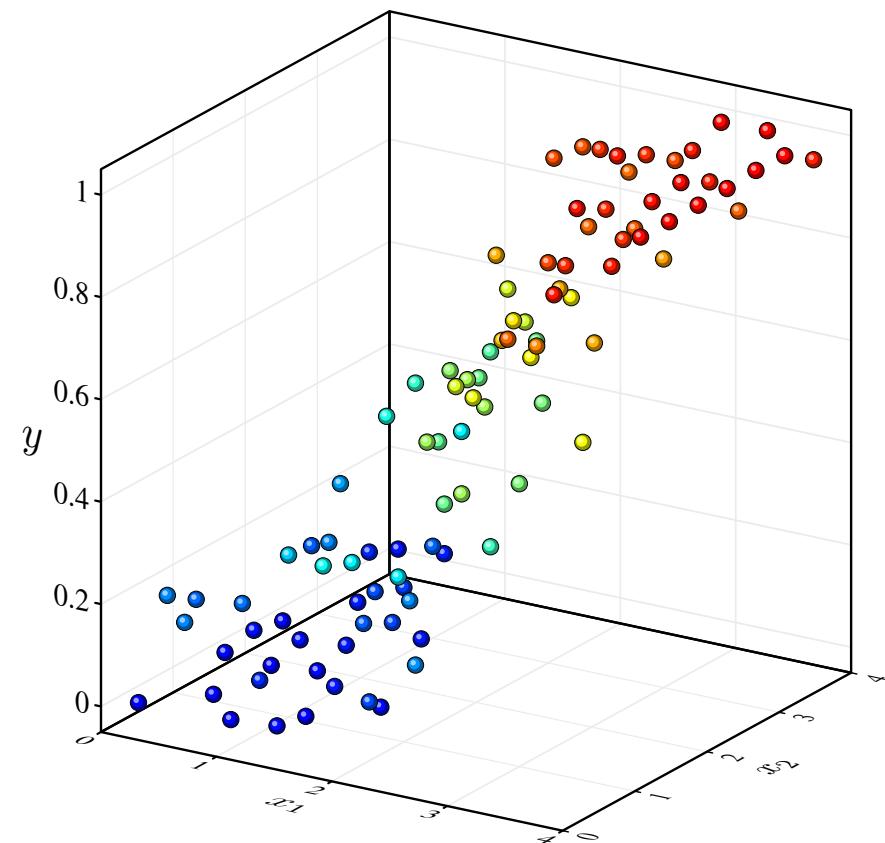
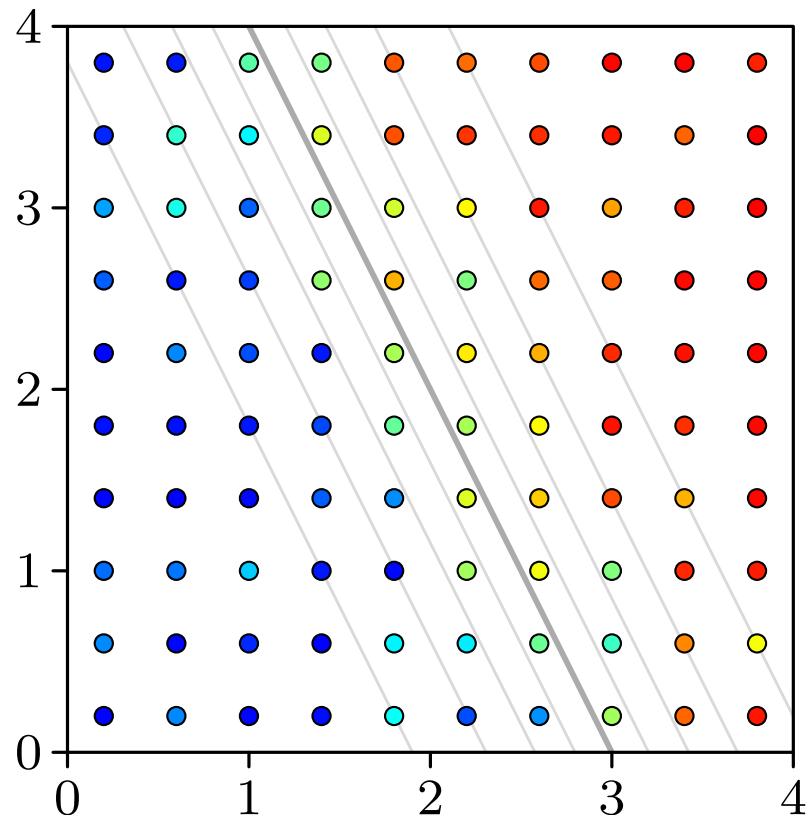
- The classes are separated at the “contour line” $p(\vec{x}; \vec{a}) = \theta = 0.5$ (inflection line). (The class boundary is linear, therefore **linear classification**.)
- Via the classification threshold θ , which need not be $\theta = 0.5$, misclassification costs may be incorporated.

Logistic Classification: Example



- In finance (e.g. when assessing the credit worthiness of businesses) logistic classification is often applied in discrete spaces, that are spanned e.g. by binary attributes and expert assessments.
(e.g. assessments of the range of products, market share, growth etc.)

Logistic Classification: Example

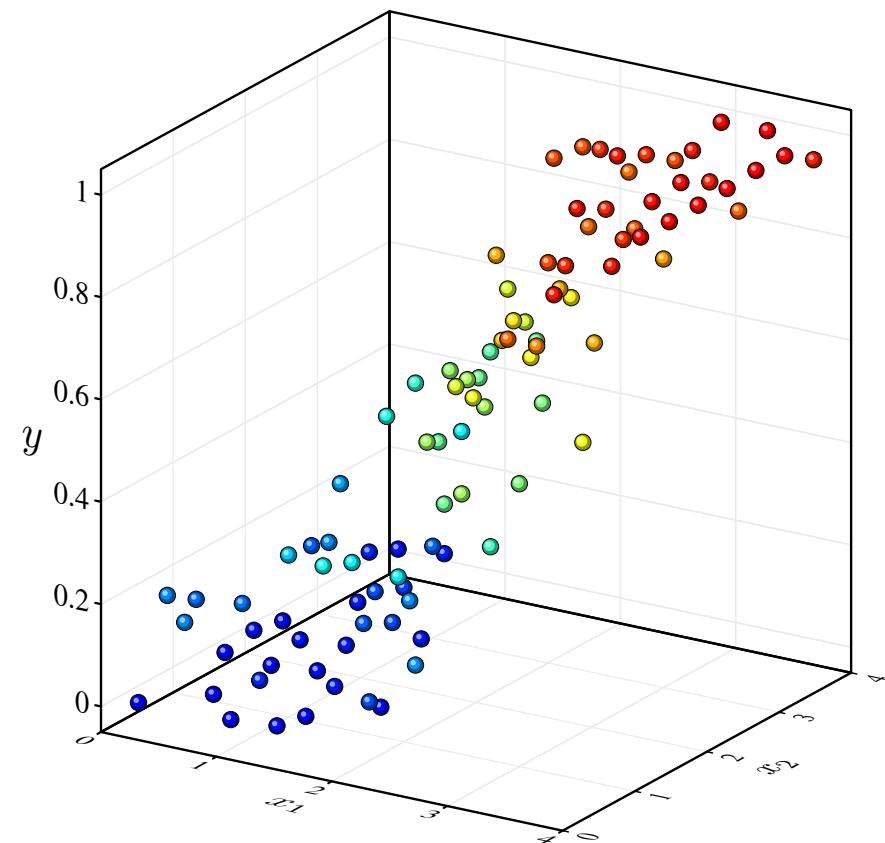
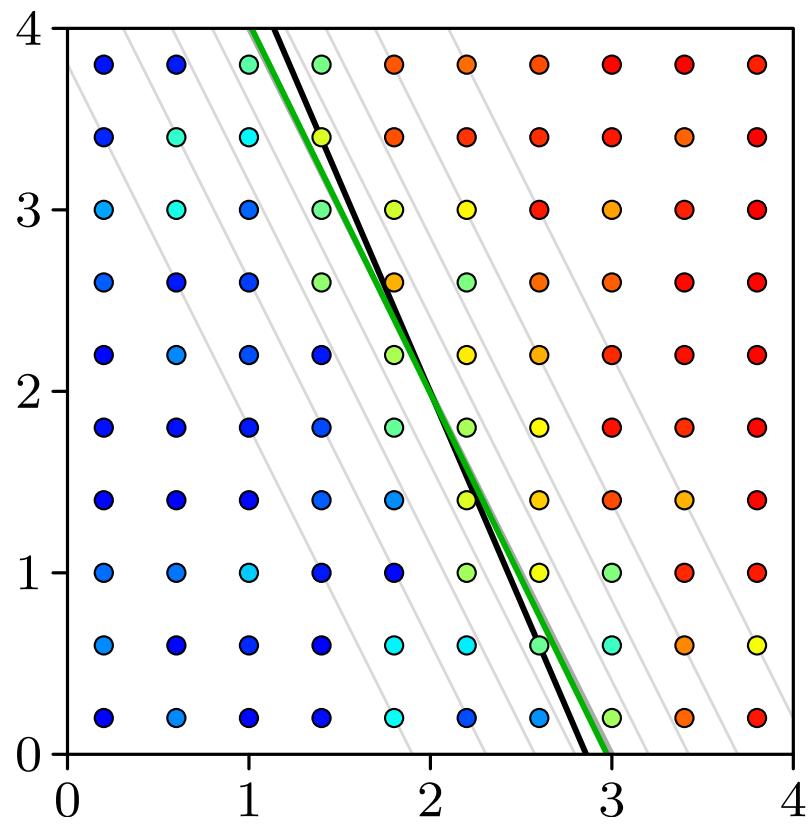


- In such a case multiple businesses may fall onto the same grid point.
- Then probabilities may be estimated from observed credit defaults:

$$\hat{p}_{\text{default}}(\vec{x}) = \frac{\#\text{defaults}(\vec{x}) + \gamma}{\#\text{loans}(\vec{x}) + 2\gamma}$$

(γ : Laplace correction, e.g. $\gamma \in \{\frac{1}{2}, 1\}$)

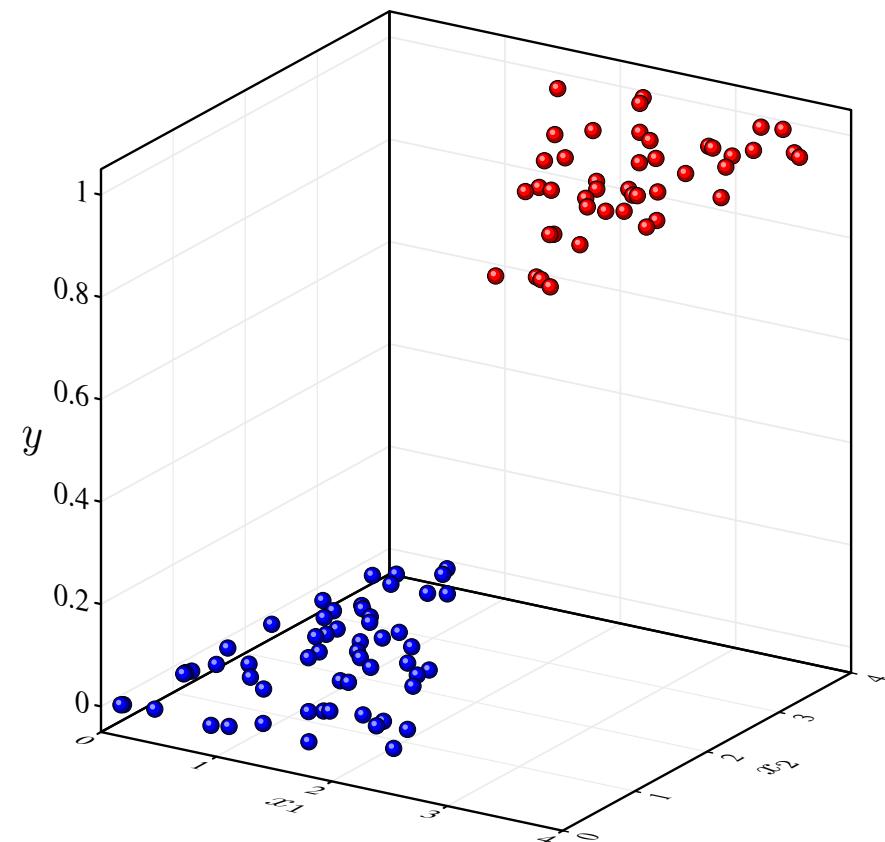
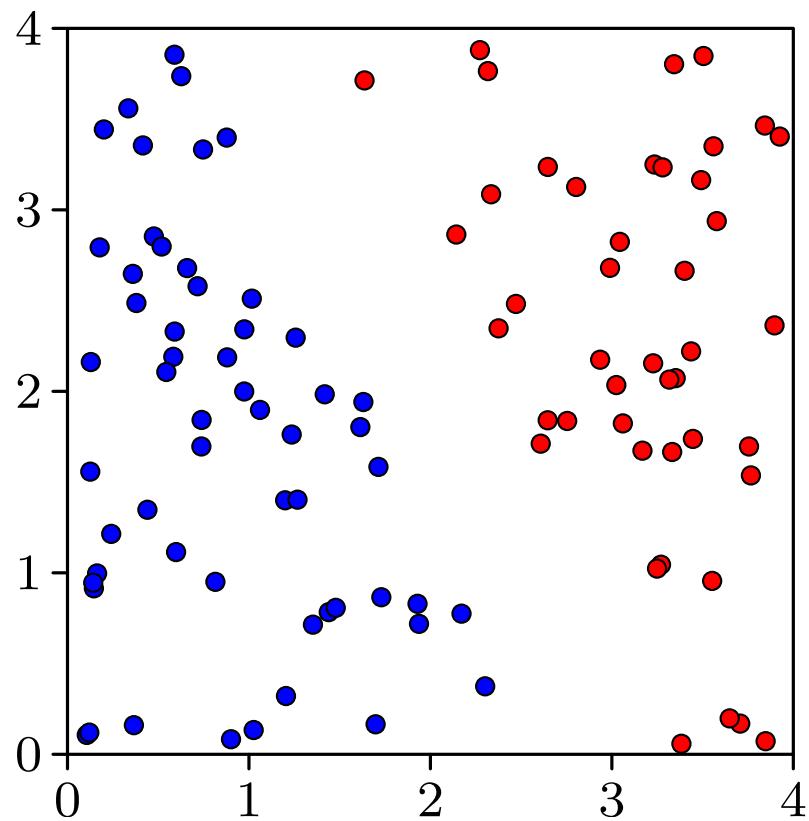
Logistic Classification: Example



- Black “contour line”: logit transform and linear regression.
- Green “contour line”: gradient descent on error function in original space.

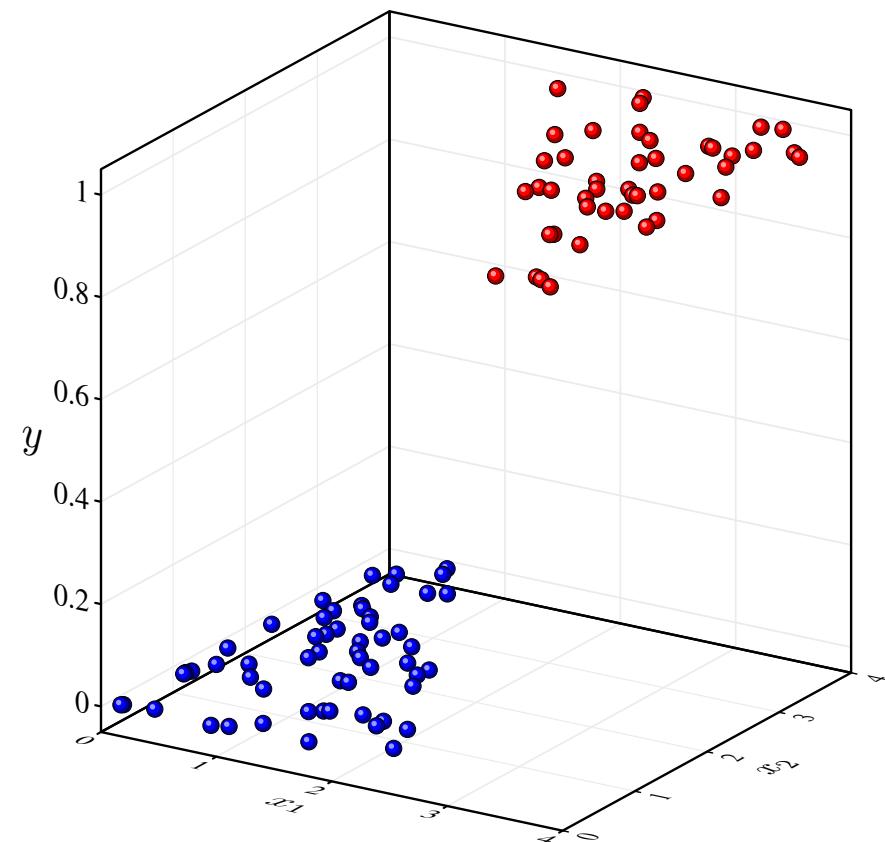
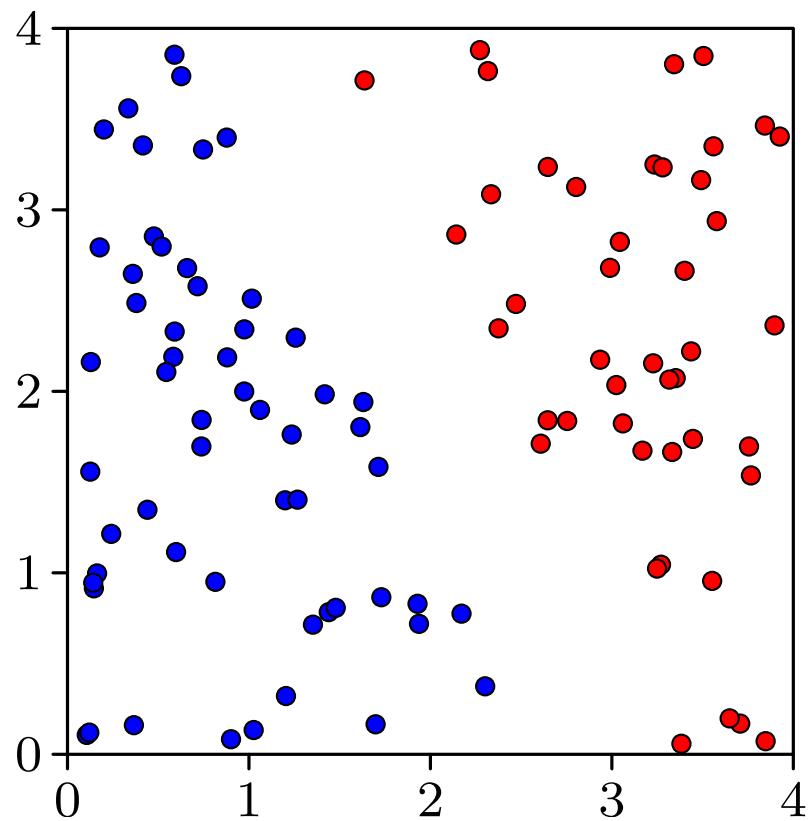
(For simplicity and clarity only the “contour lines” for $y = 0.5$ (inflection lines) are shown.)

Logistic Classification: Example



- More frequent is the case in which at least some attributes are metric and for each point a class, but no class probability is available.
- If we assign class 0: $c_0 \hat{=} y = 0$ and class 1: $c_1 \hat{=} y = 1$, the logit transform is not applicable.

Logistic Classification: Example

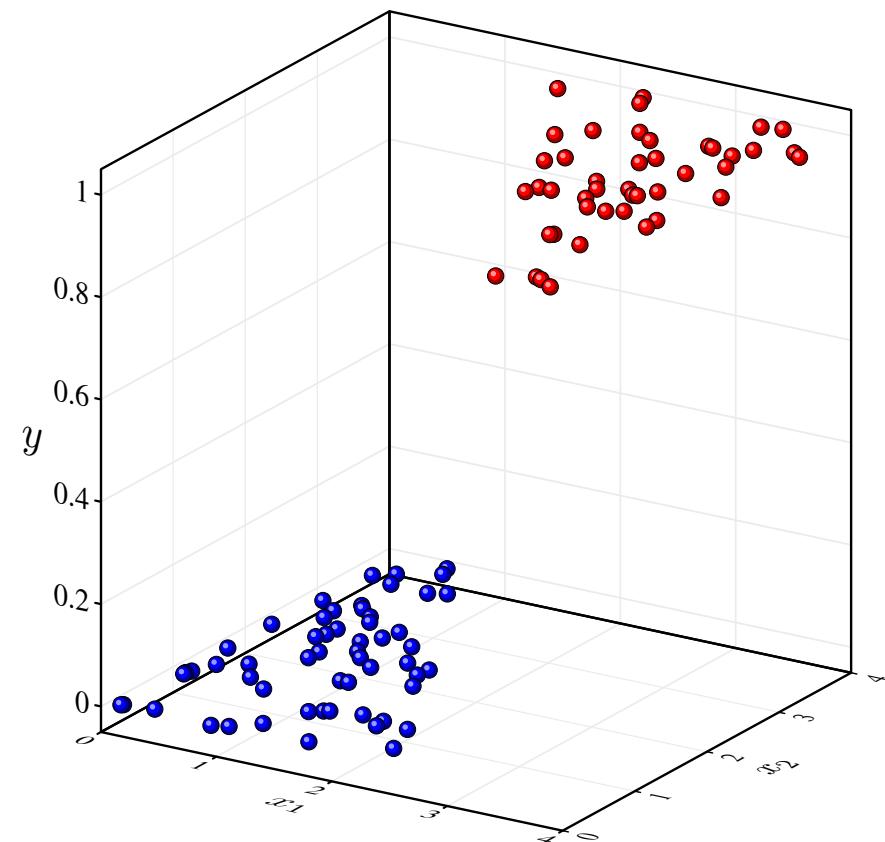
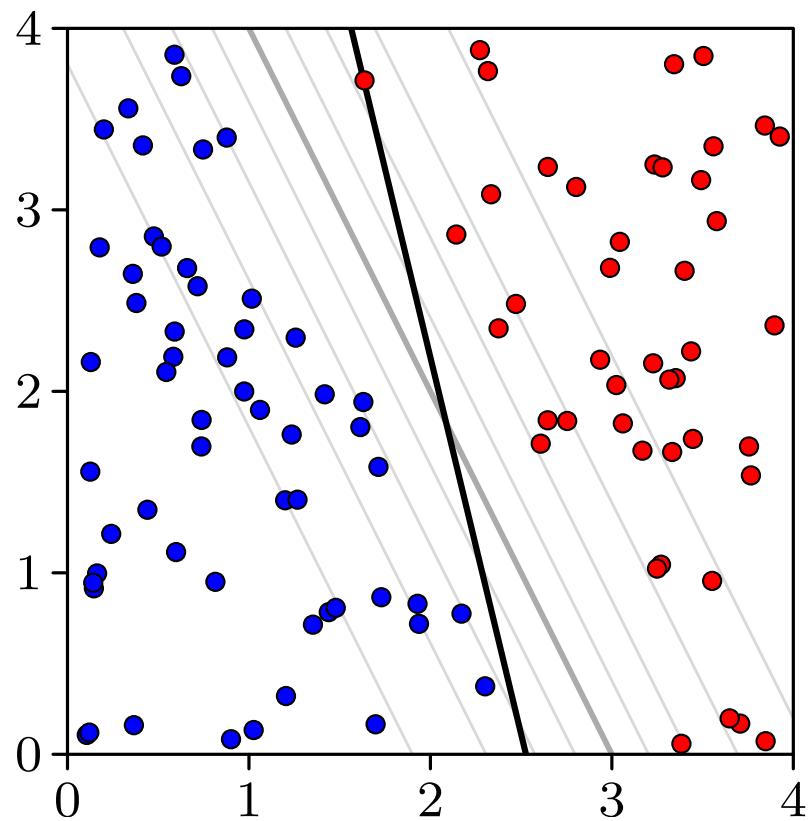


- The logit transform becomes applicable by mapping the classes to

$$c_1 \hat{=} y = \ln\left(\frac{\epsilon}{1-\epsilon}\right) \quad \text{and} \quad c_0 \hat{=} y = \ln\left(\frac{1-\epsilon}{\epsilon}\right) = -\ln\left(\frac{\epsilon}{1-\epsilon}\right).$$

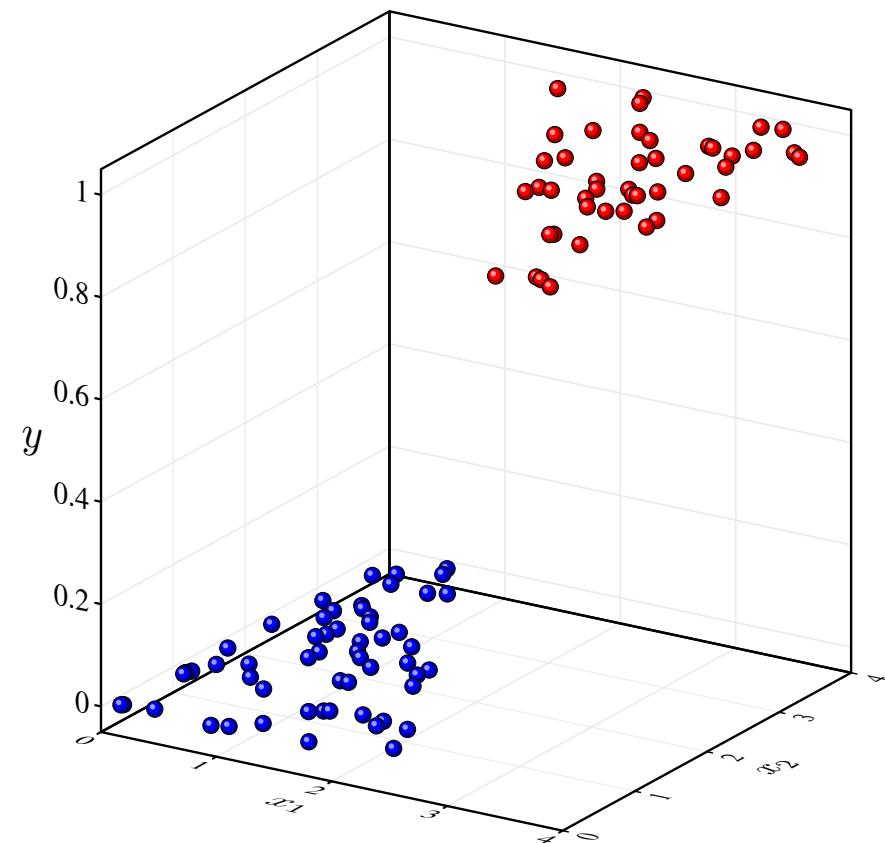
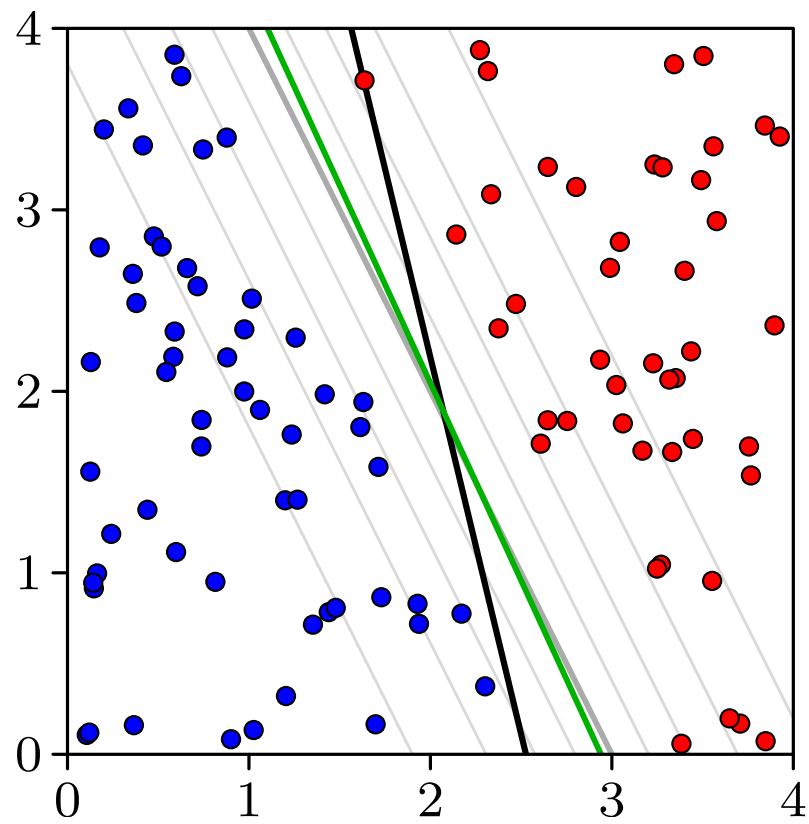
- The value of $\epsilon \in (0, \frac{1}{2})$ is irrelevant (i.e., the result is independent of ϵ and equivalent to a linear regression with $c_0 \hat{=} y = 0$ and $c_1 \hat{=} y = 1$).

Logistic Classification: Example



- Logit transform and linear regression often yield suboptimal results:
Depending on the distribution of the data points relativ to a(n optimal) separating hyperplane the computed separating hyperplane can be shifted and/or rotated.
- This can lead to (unnecessary) misclassifications!

Logistic Classification: Example



- Black “contour line”: logit transform and linear regression.
- Green “contour line”: gradient descent on error function in original space.

(For simplicity and clarity only the “contour lines” for $y = 0.5$ (inflection lines) are shown.)

Logistic Classification: Maximum Likelihood Approach

A **likelihood function** describes the probability of observed data depending on the parameters \vec{a} of the (conjectured) data generating process.

Here: **logistic function** to describe the class probabilities.

$$\begin{aligned} \text{class } y = 1 \text{ occurs with probability} & \quad p_1(\vec{x}) = f(\vec{a}^\top \vec{x}^*), \\ \text{class } y = 0 \text{ occurs with probability} & \quad p_0(\vec{x}) = 1 - f(\vec{a}^\top \vec{x}^*), \\ \text{with } f(z) = \frac{1}{1+e^{-z}} \text{ and } \vec{x}^* = (1, x_1, \dots, x_m)^\top \text{ and } \vec{a} = (a_0, a_1, \dots, a_m)^\top. \end{aligned}$$

Likelihood function for the data set $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$ with $y_i \in \{0, 1\}$:

$$\begin{aligned} L(\vec{a}) &= \prod_{i=1}^n p_1(\vec{x}_i)^{y_i} \cdot p_0(\vec{x}_i)^{1-y_i} \\ &= \prod_{i=1}^n f(\vec{a}^\top \vec{x}_i^*)^{y_i} \cdot (1 - f(\vec{a}^\top \vec{x}_i^*))^{1-y_i} \end{aligned}$$

Maximum Likelihood Approach: Find the set of parameters \vec{a} , which renders the occurrence of the (observed) data most likely.

Logistic Classification: Maximum Likelihood Approach

Simplification by taking the logarithm: **log likelihood function**

$$\begin{aligned}\ln L(\vec{a}) &= \sum_{i=1}^n \left(y_i \cdot \ln f(\vec{a}^\top \vec{x}_i^*) + (1 - y_i) \cdot \ln(1 - f(\vec{a}^\top \vec{x}_i^*)) \right) \\ &= \sum_{i=1}^n \left(y_i \cdot \ln \frac{1}{1 + e^{-(\vec{a}^\top \vec{x}_i^*)}} + (1 - y_i) \cdot \ln \frac{e^{-(\vec{a}^\top \vec{x}_i^*)}}{1 + e^{-(\vec{a}^\top \vec{x}_i^*)}} \right) \\ &= \sum_{i=1}^n \left((y_i - 1) \cdot \vec{a}^\top \vec{x}_i^* - \ln(1 + e^{-(\vec{a}^\top \vec{x}_i^*)}) \right)\end{aligned}$$

Necessary condition for a maximum:

Gradient of the objective function $\ln L(\vec{a})$ w.r.t. \vec{a} vanishes: $\vec{\nabla}_{\vec{a}} \ln L(\vec{a}) \stackrel{!}{=} \vec{0}$

Problem: The resulting equation system is not linear.

Solution possibilities:

- Gradient descent on objective function $\ln L(\vec{a})$.
- Root search on gradient $\vec{\nabla}_{\vec{a}} \ln L(\vec{a})$. (e.g. Newton–Raphson method)

Logistic Classification: Gradient Ascent

Gradient of the log likelihood function:

(mit $f(z) = \frac{1}{1+e^{-z}}$)

$$\begin{aligned}\vec{\nabla}_{\vec{a}} \ln L(\vec{a}) &= \vec{\nabla}_{\vec{a}} \sum_{i=1}^n \left((y_i - 1) \cdot \vec{a}^\top \vec{x}_i^* - \ln \left(1 + e^{-(\vec{a}^\top \vec{x}_i^*)} \right) \right) \\ &= \sum_{i=1}^n \left((y_i - 1) \cdot \vec{x}_i^* + \frac{e^{-(\vec{a}^\top \vec{x}_i^*)}}{1 + e^{-(\vec{a}^\top \vec{x}_i^*)}} \cdot \vec{x}_i^* \right) \\ &= \sum_{i=1}^n \left((y_i - 1) \cdot \vec{x}_i^* + (1 - f(\vec{a}^\top \vec{x}_i^*)) \cdot \vec{x}_i^* \right) \\ &= \sum_{i=1}^n \left((y_i - f(\vec{a}^\top \vec{x}_i^*)) \cdot \vec{x}_i^* \right)\end{aligned}$$

As a comparison:

Gradient of the sum of squared errors / deviations:

$$\vec{\nabla}_{\vec{a}} F(\vec{a}) = -2 \sum_{i=1}^n (y_i - f(\vec{a}^\top \vec{x}_i^*)) \cdot \underbrace{f(\vec{a}^\top \vec{x}_i^*) \cdot (1 - f(\vec{a}^\top \vec{x}_i^*))}_{\text{additional factor: derivative of the logistic function}} \cdot \vec{x}_i^*$$

Logistic Classification: Gradient Ascent

Given: data set $\mathbf{D} = \{(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)\}$ with n data points, $y \in \{0, 1\}$.

Simplification: Use $\vec{x}_i^* = (1, x_{i1}, \dots, x_{im})^\top$ and $\vec{a} = (a_0, a_1, \dots, a_m)^\top$.

Gradient ascent on the objective function $\ln L(\vec{a})$:

- Choose as the initial point \vec{a}_0 the result of a logit transform and a linear regression (or merely a linear regression).
- Update of the parameters \vec{a} :

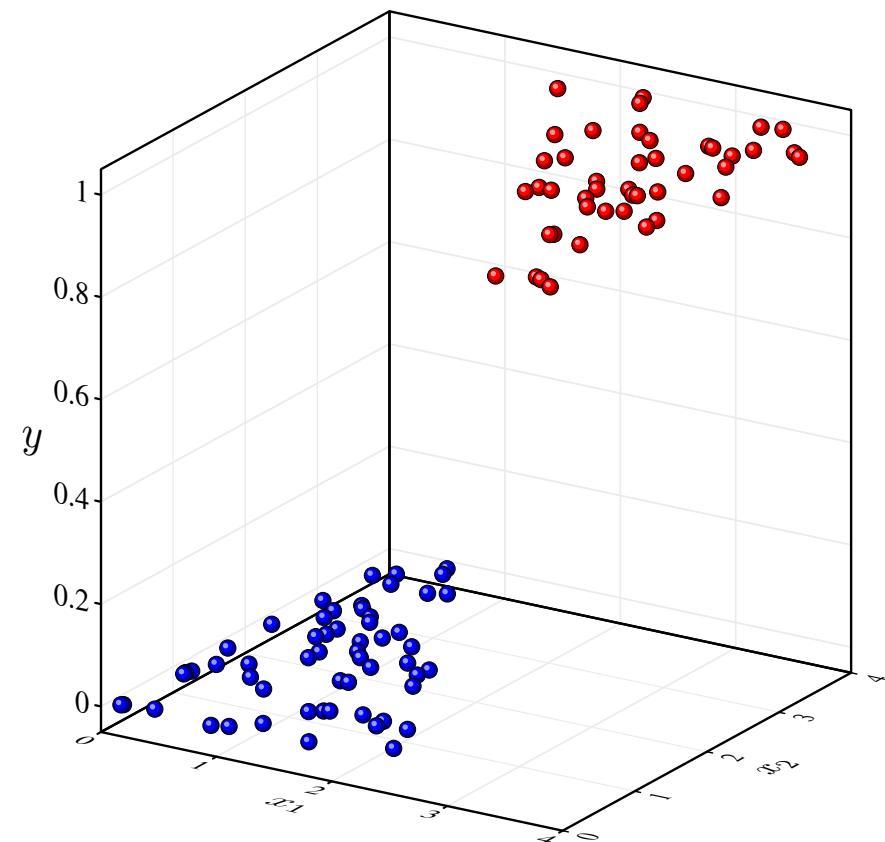
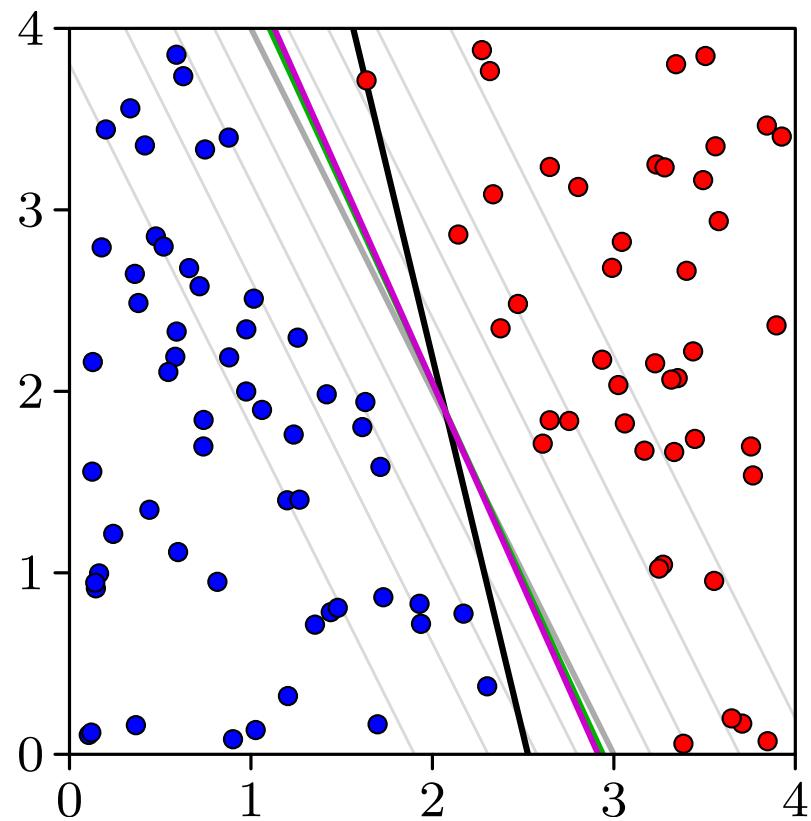
$$\begin{aligned}\vec{a}_{t+1} &= \vec{a}_t + \eta \cdot \vec{\nabla}_{\vec{a}} \ln L(\vec{a})|_{\vec{a}_t} \\ &= \vec{a}_t + \eta \cdot \sum_{i=1}^n (y_i - f(\vec{a}_t^\top \vec{x}_i^*)) \cdot \vec{x}_i^*,\end{aligned}$$

where η is a step width parameter to be chosen by a user (e.g. $\eta = 0.01$).

(Comparison with gradient descent: missing factor $f(\vec{a}_t^\top \vec{x}_i^*) \cdot (1 - f(\vec{a}_t^\top \vec{x}_i^*))$.)

- Repeat the update step until convergence, e.g. until $\|\vec{a}_{t+1} - \vec{a}_t\| < \tau$ with a chosen threshold τ (z.B. $\tau = 10^{-6}$).

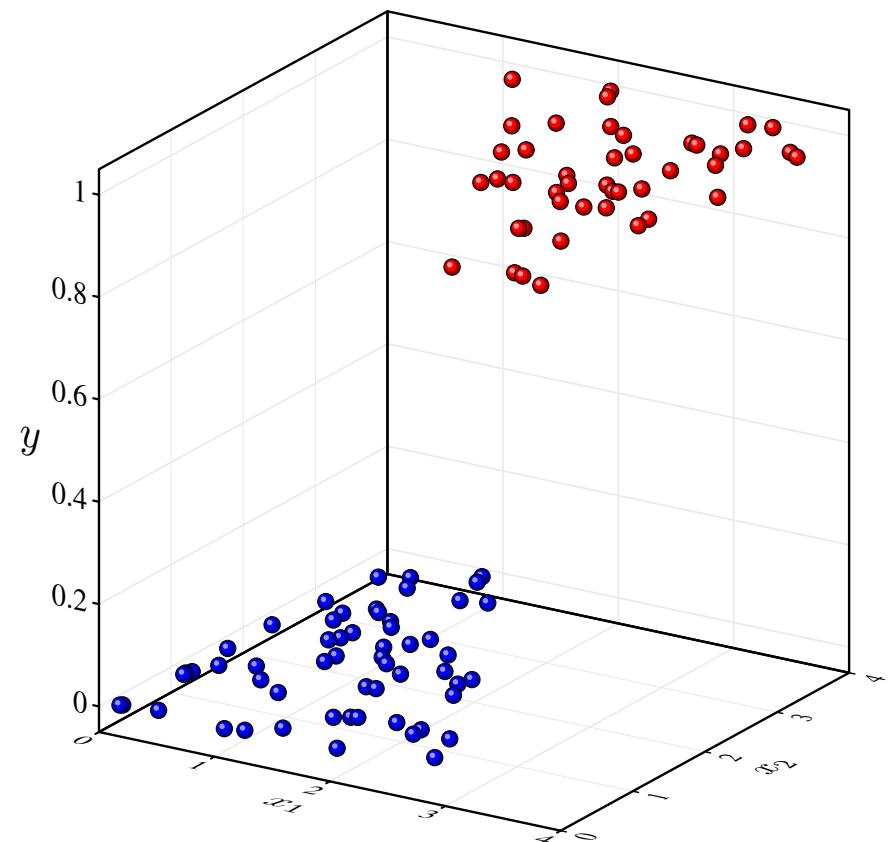
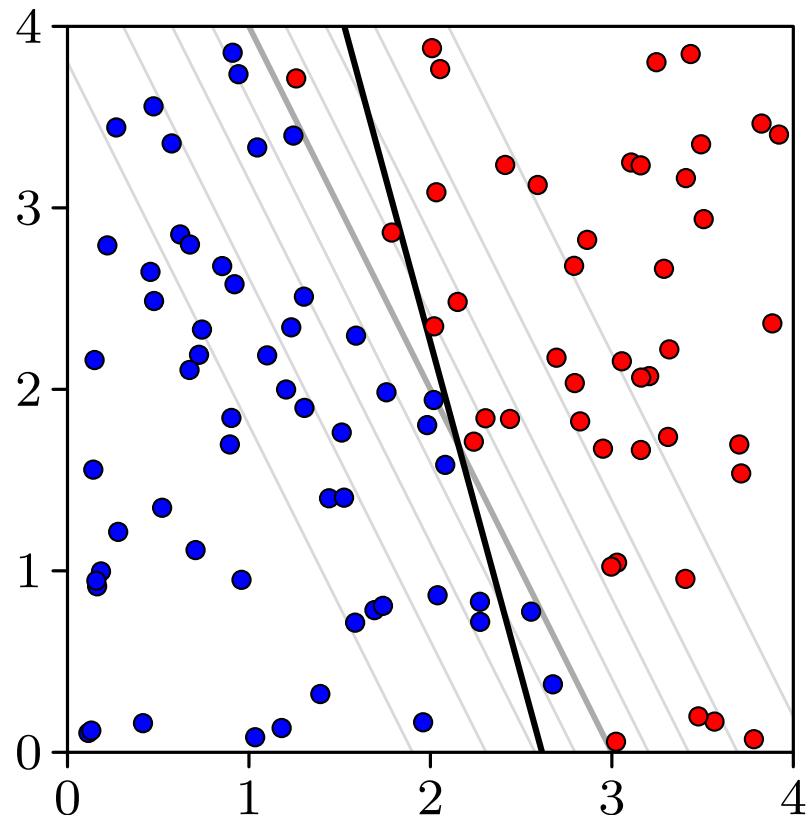
Logistic Classification: Example



- Black “contour line”: logit transform and linear regression.
- Green “contour line”: gradient descent on error function in the original space.
- Magenta “contour line”: gradient ascent on log likelihood function.

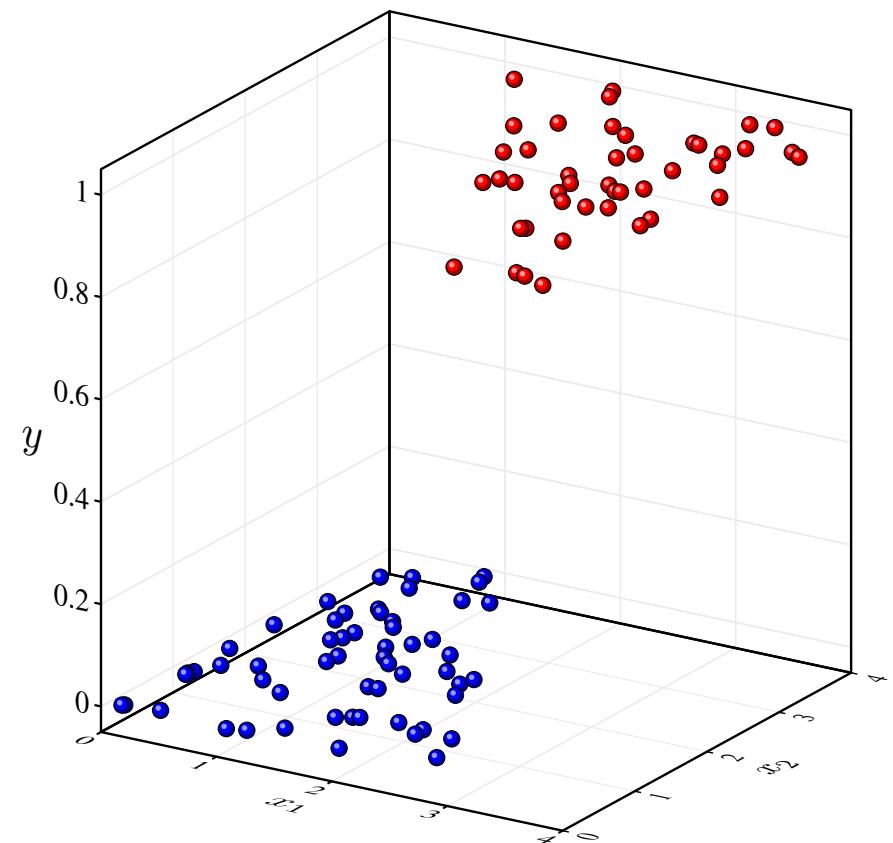
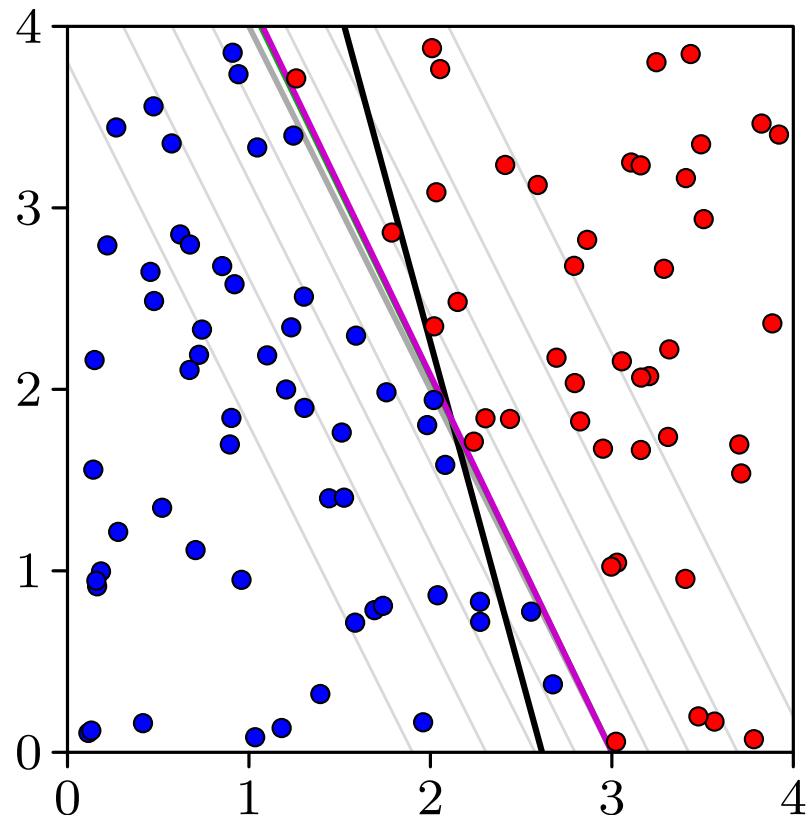
(For simplicity and clarity only the “contour lines” for $y = 0.5$ (inflection lines) are shown.)

Logistic Classification: No Gap Between Classes



- If there is no (clear) gap between the classes
a logit transform and subsequent linear regression
yields (unnecessary) misclassifications even more often.
- In such a case the alternative methods are clearly preferable!

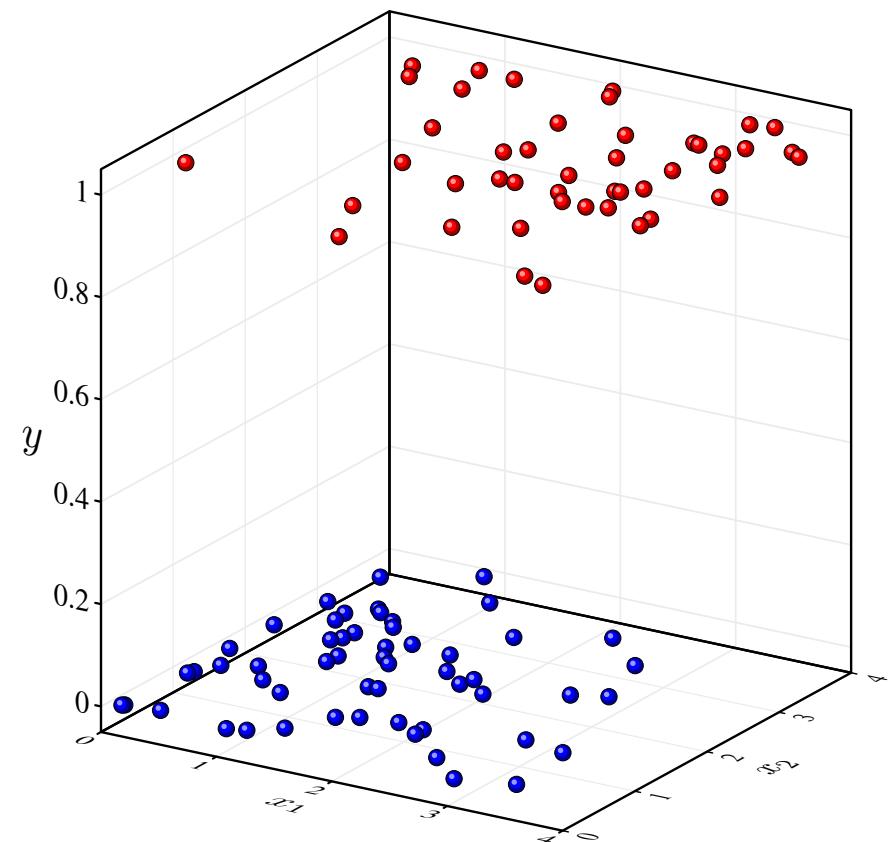
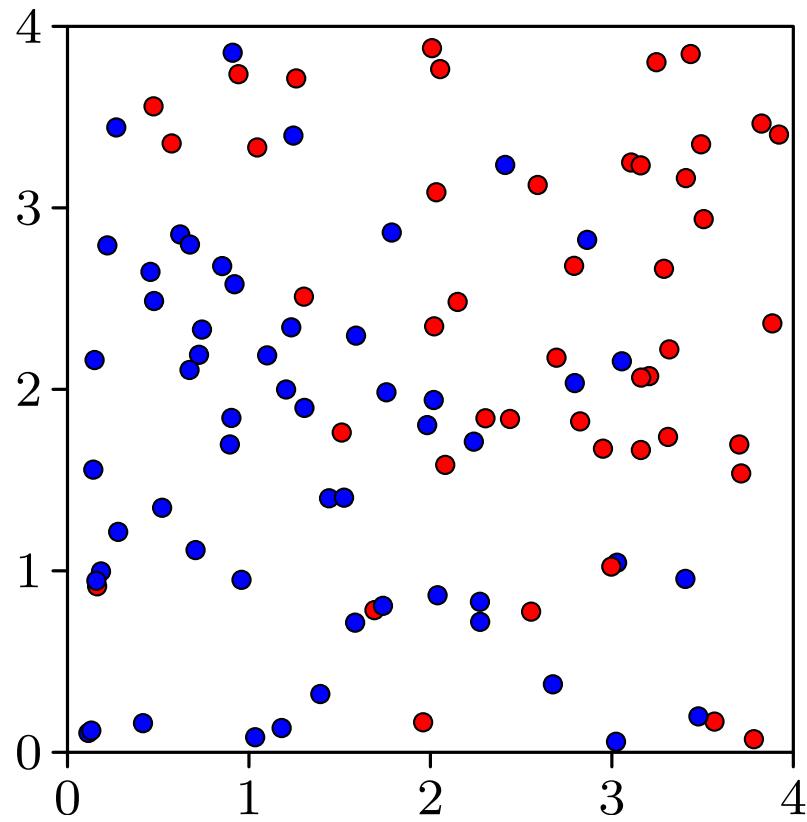
Logistic Classification: No Gap Between Classes



- Black “contour line”: logit transform and linear regression.
- Green “contour line”: gradient descent on error function in the original space.
- Magenta “contour line”: gradient ascent on log likelihood function.

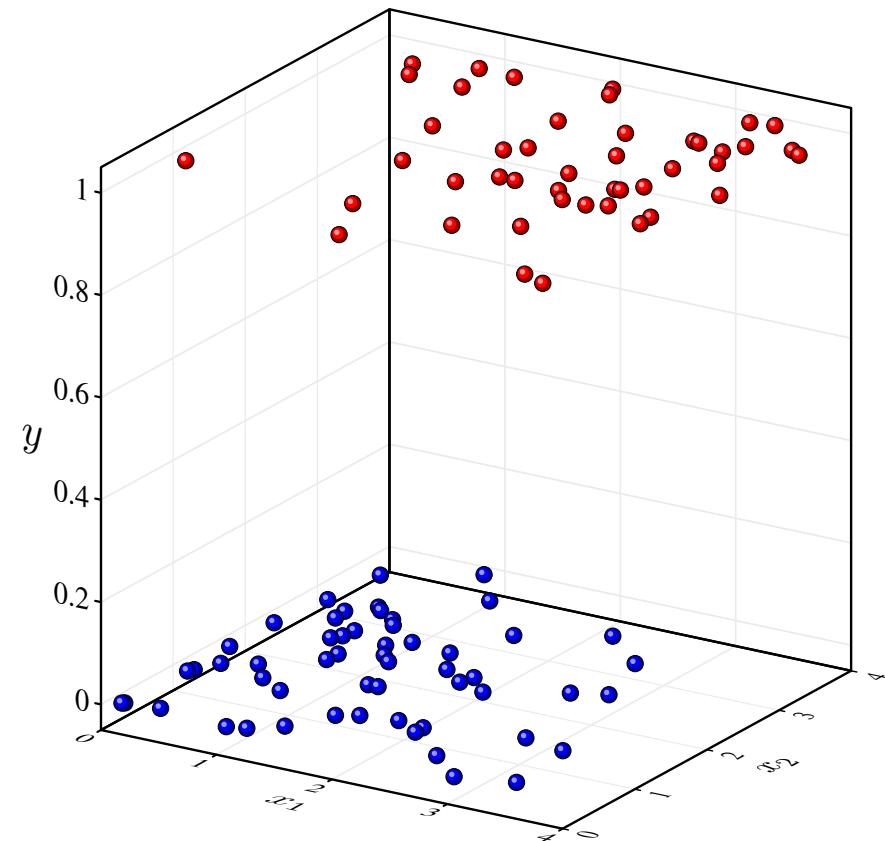
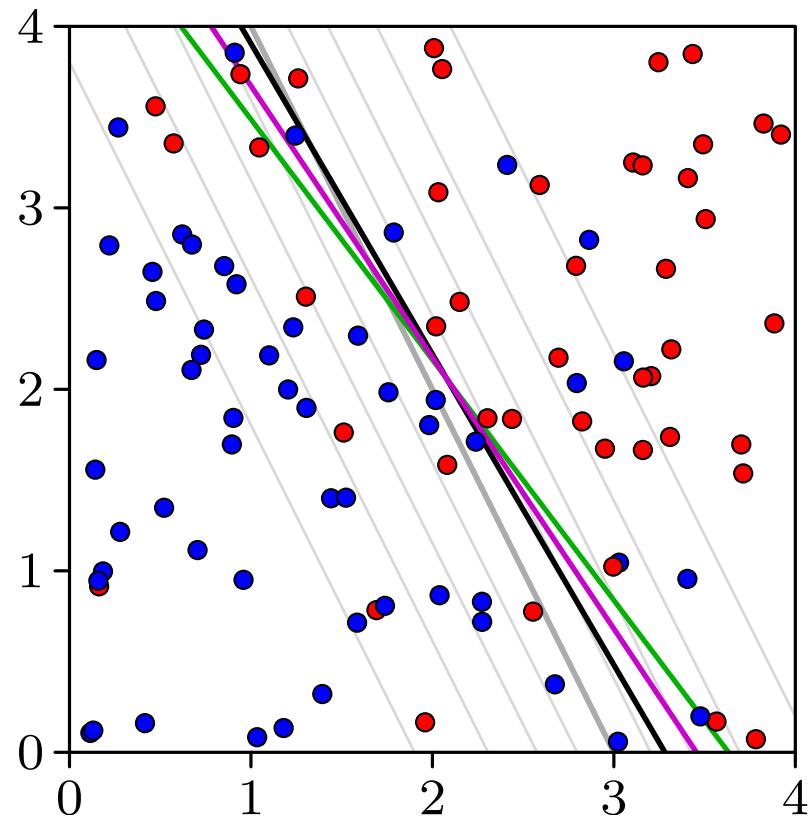
(For simplicity and clarity only the “contour lines” for $y = 0.5$ (inflection lines) are shown.)

Logistic Classification: Overlapping Classes



- Even more problematic is the situation if the classes overlap (i.e., there is no perfect separating line/hyperplane).
- In such a case even the other methods cannot avoid misclassifications.
(There is no way to be better than the pure or Bayes error.)

Logistic Classification: Overlapping Classes



- Black “contour line”: logit transform and linear regression.
- Green “contour line”: gradient descent on error function in the original space.
- Magenta “contour line”: gradient ascent on log likelihood function.

(For simplicity and clarity only the “contour lines” for $y = 0.5$ (inflection lines) are shown.)

Logistic Classification: Newton–Raphson

With the abbreviation $f(z) = \frac{1}{1+e^{-z}}$ for the logistic function it is

$$\begin{aligned}\vec{\nabla}_{\vec{a}}^2 \ln L(\vec{a}) &= \vec{\nabla}_{\vec{a}} \sum_{i=1}^n ((y_i - f(\vec{a}^\top \vec{x}_i^*)) \cdot \vec{x}_i^*) \\ &= \vec{\nabla}_{\vec{a}} \sum_{i=1}^n ((y_i \vec{x}_i^* - f(\vec{a}^\top \vec{x}_i^*) \cdot \vec{x}_i^*)) \\ &= - \sum_{i=1}^n f'(\vec{a}^\top \vec{x}_i^*) \cdot \vec{x}_i^* \vec{x}_i^{*\top} = - \sum_{i=1}^n f(\vec{a}^\top \vec{x}_i^*) \cdot (1 - f(\vec{a}^\top \vec{x}_i^*)) \cdot \vec{x}_i^* \vec{x}_i^{*\top}\end{aligned}$$

where again $f'(z) = f(z) \cdot (1 - f(z))$ (as derived above).

Thus we get for the update of the parameters \vec{a} : (note: no step width η)

$$\begin{aligned}\vec{a}_{t+1} &= \vec{a}_t - \left(\vec{\nabla}_{\vec{a}}^2 \ln L(\vec{a}) \Big|_{\vec{a}_t} \right)^{-1} \cdot \vec{\nabla}_{\vec{a}} \ln L(\vec{a}) \Big|_{\vec{a}_t} \\ &= \vec{a}_t + \left(\sum_{i=1}^n f(\vec{a}_t^\top \vec{x}_i^*) \cdot (1 - f(\vec{a}_t^\top \vec{x}_i^*)) \cdot \vec{x}_i^* \vec{x}_i^{*\top} \right)^{-1} \cdot \left(\sum_{i=1}^n (y_i - f(\vec{a}_t^\top \vec{x}_i^*)) \cdot \vec{x}_i^* \right).\end{aligned}$$

Robust Regression

- Solutions of (ordinary) least squares regression can be strongly affected by outliers.
The reason for this is obviously the squared error function, which weight outliers ver heavily (quadratically).
- More robust results can usually be obtained by minimizing the sum of absolute deviations (**least absolute deviations, LAD**).
- However, this approach has the disadvantage of not being analytically solvable (like least squares) and thus has to be addressed with iterative methods right from the start.
- In addition, least absolute deviation solutions can be unstable in the sense that small changes in the data can lead to “jumps” (discontinuous changes) of the solution parameters.
Instead, least squares solutions always changes “smoothly” (continuously).
- Finally, severe outliers can still have a distorting effect on the solution.

Robust Regression

- In order to improve the robustness of the procedure, more sophisticated regression methods have been developed: **robust regression**, which include:
 - **M-estimation** and **S-estimation** for regression and
 - **least trimmed squares (LTS)**, which simply uses a subset of at least half the size of the data set that yields the smallest sum of squared errors.

Here, we take a closer look at M-estimators.

- We rewrite the error functional (that is, the sum of squared errors) to be minimized in the form

$$F_\rho(a, b) = \sum_{i=1}^n \rho(e_i) = \sum_{i=1}^n \rho(\vec{x}_i^\top \vec{a} - y_i)$$

where $\rho(e_i) = e_i^2$ and e_i is the (signed) error of the regression function at the i th point, that is $e_i = e(x_i, y_i, \vec{a}) = f_{\vec{a}}(x_i) - y_i$, where f is the conjectured regression function family with parameters \vec{a} .

Robust Regression: M-Estimators

- Is $\rho(e_i) = e_i^2$ the only reasonable choice for the function ρ ? Certainly not.
- However, ρ should satisfy at least some reasonable restrictions.
 - The function ρ should always be positive, except for the case $e_i = 0$.
 - The sign of the error e_i should not matter for ρ .
 - ρ should be increasing when the absolute value of the error increases.
- These requirements can be formalized in the following way:

$$\begin{array}{lll} \rho(e) & \geq & 0, \\ \rho(e) & = & \rho(-e), \end{array} \quad \begin{array}{lll} \rho(0) & = & 0, \\ \rho(e_i) & \geq & \rho(e_j) \text{ if } |e_i| \geq |e_j|. \end{array}$$

- Parameter estimation (here the estimation of the parameter vector \vec{a}) is based on an objective function of the form

$$F_\rho(a, b) = \sum_{i=1}^n \rho(e_i) = \sum_{i=1}^n \rho(\vec{x}_i^\top \vec{a} - y_i)$$

and an error measure satisfying the above conditions is called an **M-estimator**.

Robust Regression: M-Estimators

- Parameter estimation (here the estimation of the parameter vector \vec{a}) based on an objective function of the form

$$F_\rho(a, b) = \sum_{i=1}^n \rho(e_i) = \sum_{i=1}^n \rho(\vec{x}_i^\top \vec{a} - y_i)$$

and an error measure satisfying the above conditions is called an **M-estimator**.

- Examples of such estimators are:

Method	$\rho(e)$
Least squares	e^2
Huber	$\begin{cases} \frac{1}{2}e^2 & \text{if } e \leq k, \\ k e - \frac{1}{2}k^2 & \text{if } e > k. \end{cases}$
Tukey's bisquare	$\begin{cases} \frac{k^2}{6} \left(1 - \left(1 - \left(\frac{e}{k} \right)^2 \right)^3 \right), & \text{if } e \leq k, \\ \frac{k^2}{6}, & \text{if } e > k. \end{cases}$

Robust Regression

- In order to understand the more general setting of an error measure ρ , it is useful to consider the derivative $\psi = \rho'$.
- Taking the derivatives of the objective function

$$F_\rho(a, b) = \sum_{i=1}^n \rho(e_i) = \sum_{i=1}^n \rho(\vec{x}_i^\top \vec{a} - y_i)$$

with respect to the parameters a_i , we obtain a system of $(m + 1)$ linear equations

$$\sum_{i=1}^n \psi_i(\vec{x}_i^\top \vec{a} - y_i) \vec{x}_i^\top = 0.$$

- Defining $w(e) = \psi(e)/e$ and $w_i = w(e_i)$, the system of linear equations can be rewritten in the form

$$\sum_{i=1}^n \frac{\psi_i(\vec{x}_i^\top \vec{a} - y_i)}{e_i} \cdot e_i \cdot \vec{x}_i^\top = \sum_{i=1}^n w_i \cdot (y_i - \vec{x}_i^\top b) \cdot \vec{x}_i^\top = 0.$$

Robust Regression

- Solving this system of linear equations corresponds to solving a standard least squares problem with (non-fixed) weights in the form $\sum_{i=1}^n w_i e_i^2$.
- However, the weights w_i depend on the residuals e_i , the residuals depend on the coefficients a_i , and the coefficients depend on the weights.
- Therefore, it is in general not possible to provide an explicit solution.
- Instead of an analytical solution, the following iteration scheme is applied:
 1. Choose an initial solution $\vec{a}^{(0)}$,
for instance the standard least squares solution setting all weights to $w_i = 1$.
 2. In each iteration step t , calculate the residuals $e^{(t-1)}$ and the corresponding weights $w^{(t-1)} = w(e^{(t-1)})$ determined by the previous step.
 3. Solve the weighted least squares problem $\sum_{i=1}^n w_i e_i^2$ which leads to

$$\vec{a}^{(t)} = (\mathbf{X}^\top \mathbf{W}^{(t-1)} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{W}^{(t-1)} \vec{y},$$

where \mathbf{W} stands for a diagonal matrix with weights w_i on the diagonal.

Robust Regression

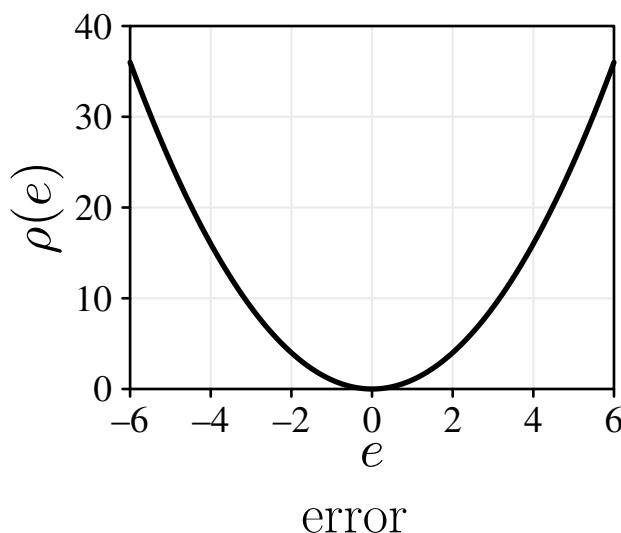
- The error measures and the weights are related as follows:

Method	$w(e)$
Least squares	1
Huber	$\begin{cases} 1, & \text{if } e \leq k, \\ k/ e , & \text{if } e > k. \end{cases}$
Tukey's bisquare	$\begin{cases} \left(1 - \left(\frac{e}{k}\right)^2\right)^2, & \text{if } e \leq k, \\ 0, & \text{if } e > k. \end{cases}$

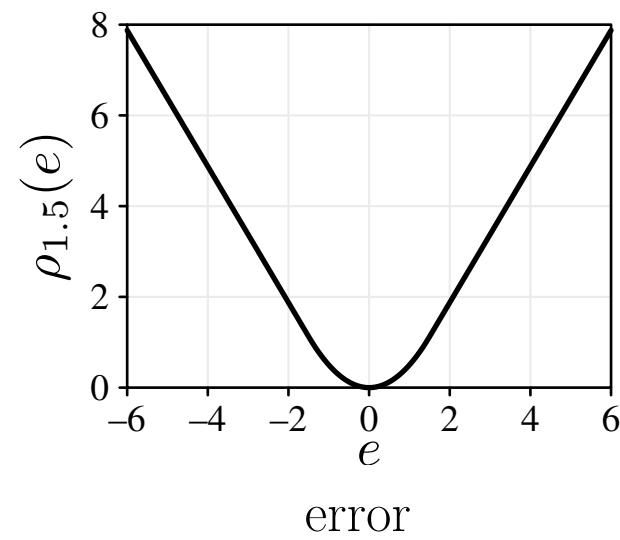
- Note that the weights are an additional result of the procedure (beyond the actually desired regression function).
- They provide information which data points may be considered as outliers (those with low weights, as this indicates that they have not been fitted well).
- Note also that the weights may be plotted even for high-dimensional data sets (using some suitable arrangement of the data points, e.g., sorted by weight).

Robust Regression

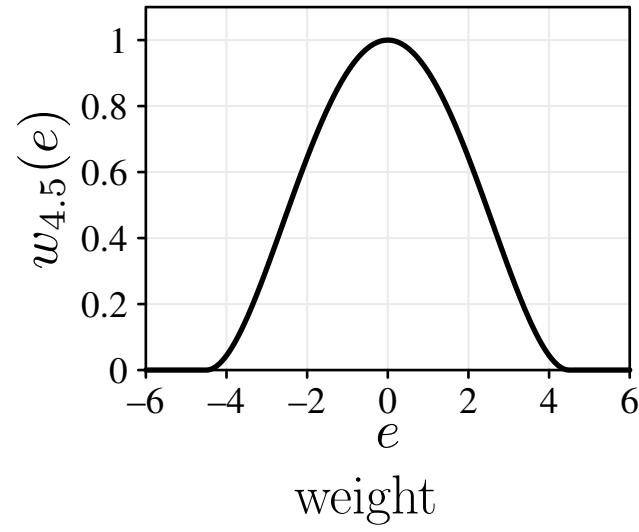
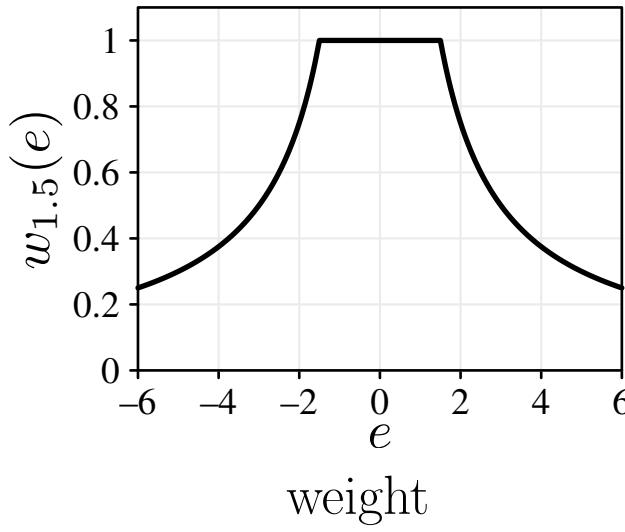
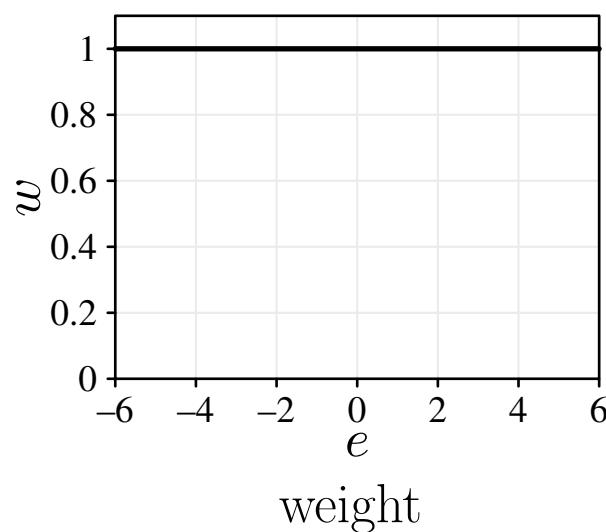
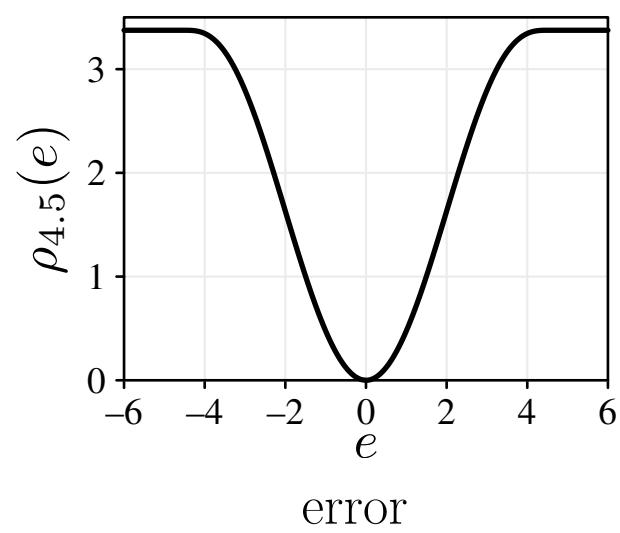
(ordinary) least squares



Huber



Tukey's bisquare



Robust Regression

- The **(ordinary) least squares** error increases in a quadratic manner with increasing distance. The weights are always constant.

This means that extreme outliers will have full influence on the regression coefficients and can corrupt the result completely.

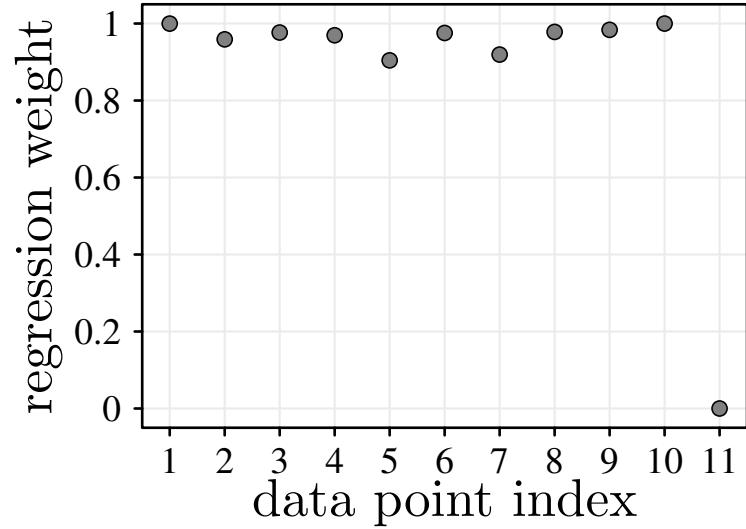
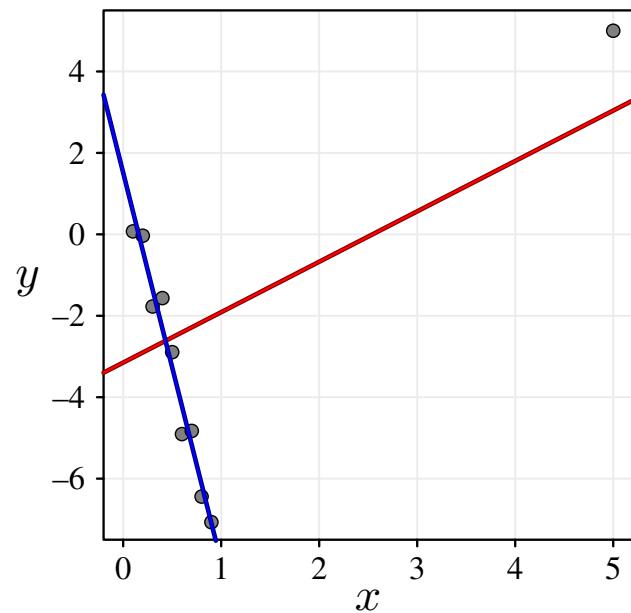
- In the more robust approach by **Huber** the change of the error measure ρ switches from a quadratic increase for small errors to a linear increase for larger errors.

As a result, only data points with small errors will have full influence on the regression coefficients. For extreme outliers the weights tend to zero.

- **Tukey's bisquare** approach is even more drastic than Huber's. For larger errors the error measure ρ does not increase at all, but remains constant. As a consequence, the weights for outliers drop to zero if they are too far away from the regression curve.

This means that extreme outliers have no influence on the regression curve at all.

Robust Regression: Example



- There is one outlier that leads to the red regression line that neither fits the outlier nor the other points.
- With robust regression, for instance based on Huber's ρ -function, we obtain the blue regression line that simply ignores the outlier.
- An additional result are the computed weights for the data points. In this way, outlier can be identified by robust regression.

Summary Regression

- **Minimize the Sum of Squared Errors**
 - Write the sum of squared errors as a function of the parameters to be determined.
- **Exploit Necessary Conditions for a Minimum**
 - Partial derivatives w.r.t. the parameters to determine must vanish.
- **Solve the System of Normal Equations**
 - The best fit parameters are the solution of the system of normal equations.
- **Non-polynomial Regression Functions**
 - Find a transformation to the multipolynomial case.
 - Logistic regression can be used to solve two class classification problems.
- **Robust Regression**
 - Reduce the influence of outliers by using different error measures.

Bayes Classifiers

Bayes Classifiers

- **Probabilistic Classification and Bayes' Rule**
- **Naive Bayes Classifiers**
 - Derivation of the classification formula
 - Probability estimation and Laplace correction
 - Simple examples of naive Bayes classifiers
 - A naive Bayes classifier for the Iris data
- **Full Bayes Classifiers**
 - Derivation of the classification formula
 - Comparison to naive Bayes classifiers
 - A simple example of a full Bayes classifier
 - A full Bayes classifier for the Iris data
- **Summary**

Probabilistic Classification

- A classifier is an algorithm that assigns a class from a predefined set to a case or object, based on the values of descriptive attributes.
- An optimal classifier maximizes the probability of a correct class assignment.
 - Let C be a class attribute with $\text{dom}(C) = \{c_1, \dots, c_{n_C}\}$, which occur with probabilities p_i , $1 \leq i \leq n_C$.
 - Let q_i be the probability with which a classifier assigns class c_i . ($q_i \in \{0, 1\}$ for a deterministic classifier)
 - The probability of a correct assignment is

$$P(\text{correct assignment}) = \sum_{i=1}^{n_C} p_i q_i.$$

- Therefore the best choice for the q_i is

$$q_i = \begin{cases} 1, & \text{if } p_i = \max_{k=1}^{n_C} p_k, \\ 0, & \text{otherwise.} \end{cases}$$

Probabilistic Classification

- Consequence: An optimal classifier should assign the **most probable class**.
- This argument does not change if we take descriptive attributes into account.
 - Let $U = \{A_1, \dots, A_m\}$ be a set of descriptive attributes with domains $\text{dom}(A_k)$, $1 \leq k \leq m$.
 - Let $A_1 = a_1, \dots, A_m = a_m$ be an instantiation of the attributes.
 - An optimal classifier should assign the class c_i for which

$$P(C = c_i \mid A_1 = a_1, \dots, A_m = a_m) = \\ \max_{j=1}^{n_C} P(C = c_j \mid A_1 = a_1, \dots, A_m = a_m)$$

- **Problem:** We cannot store a class (or the class probabilities) for every possible instantiation $A_1 = a_1, \dots, A_m = a_m$ of the descriptive attributes. (The table size grows exponentially with the number of attributes.)
- Therefore: **Simplifying assumptions are necessary.**

Bayes' Rule and Bayes' Classifiers

- Bayes' rule is a formula that can be used to “invert” conditional probabilities:
Let X and Y be events, $P(X) > 0$. Then

$$P(Y \mid X) = \frac{P(X \mid Y) \cdot P(Y)}{P(X)}.$$

- Bayes' rule follows directly from the definition of conditional probability:

$$P(Y \mid X) = \frac{P(X \cap Y)}{P(X)} \quad \text{and} \quad P(X \mid Y) = \frac{P(X \cap Y)}{P(Y)}.$$

- Bayes' classifiers: Compute the class probabilities as

$$P(C = c_i \mid A_1 = a_1, \dots, A_m = a_m) = \frac{P(A_1 = a_1, \dots, A_m = a_m \mid C = c_i) \cdot P(C = c_i)}{P(A_1 = a_1, \dots, A_m = a_m)}.$$

- Looks unreasonable at first sight: Even more probabilities to store.

Naive Bayes Classifiers

Naive Assumption:

The descriptive attributes are conditionally independent given the class.

Bayes' Rule:

$$P(C = c_i \mid \omega) = \frac{P(A_1 = a_1, \dots, A_m = a_m \mid C = c_i) \cdot P(C = c_i)}{P(A_1 = a_1, \dots, A_m = a_m)} \leftarrow p_0$$

Chain Rule of Probability:

$$P(C = c_i \mid \omega) = \frac{P(C = c_i)}{p_0} \cdot \prod_{k=1}^m P(A_k = a_k \mid A_1 = a_1, \dots, A_{k-1} = a_{k-1}, C = c_i)$$

Conditional Independence Assumption:

$$P(C = c_i \mid \omega) = \frac{P(C = c_i)}{p_0} \cdot \prod_{k=1}^m P(A_k = a_k \mid C = c_i)$$

Reminder: Chain Rule of Probability

- Based on the **product rule** of probability:

$$P(A \wedge B) = P(A \mid B) \cdot P(B)$$

(Multiply definition of conditional probability with $P(B)$.)

- **Multiple application** of the product rule yields:

$$\begin{aligned} P(A_1, \dots, A_m) &= P(A_m \mid A_1, \dots, A_{m-1}) \cdot P(A_1, \dots, A_{m-1}) \\ &= P(A_m \mid A_1, \dots, A_{m-1}) \\ &\quad \cdot P(A_{m-1} \mid A_1, \dots, A_{m-2}) \cdot P(A_1, \dots, A_{m-2}) \\ &= \vdots \\ &= \prod_{k=1}^m P(A_k \mid A_1, \dots, A_{k-1}) \end{aligned}$$

- The scheme works also if there is already a condition in the original expression:

$$P(A_1, \dots, A_m \mid C) = \prod_{i=1}^m P(A_i \mid A_1, \dots, A_{i-1}, C)$$

Conditional Independence

- Reminder: **stochastic independence** (unconditional)

$$P(A \wedge B) = P(A) \cdot P(B)$$

(Joint probability is the product of the individual probabilities.)

- Comparison to the **product rule**

$$P(A \wedge B) = P(A | B) \cdot P(B)$$

shows that this is equivalent to

$$P(A | B) = P(A)$$

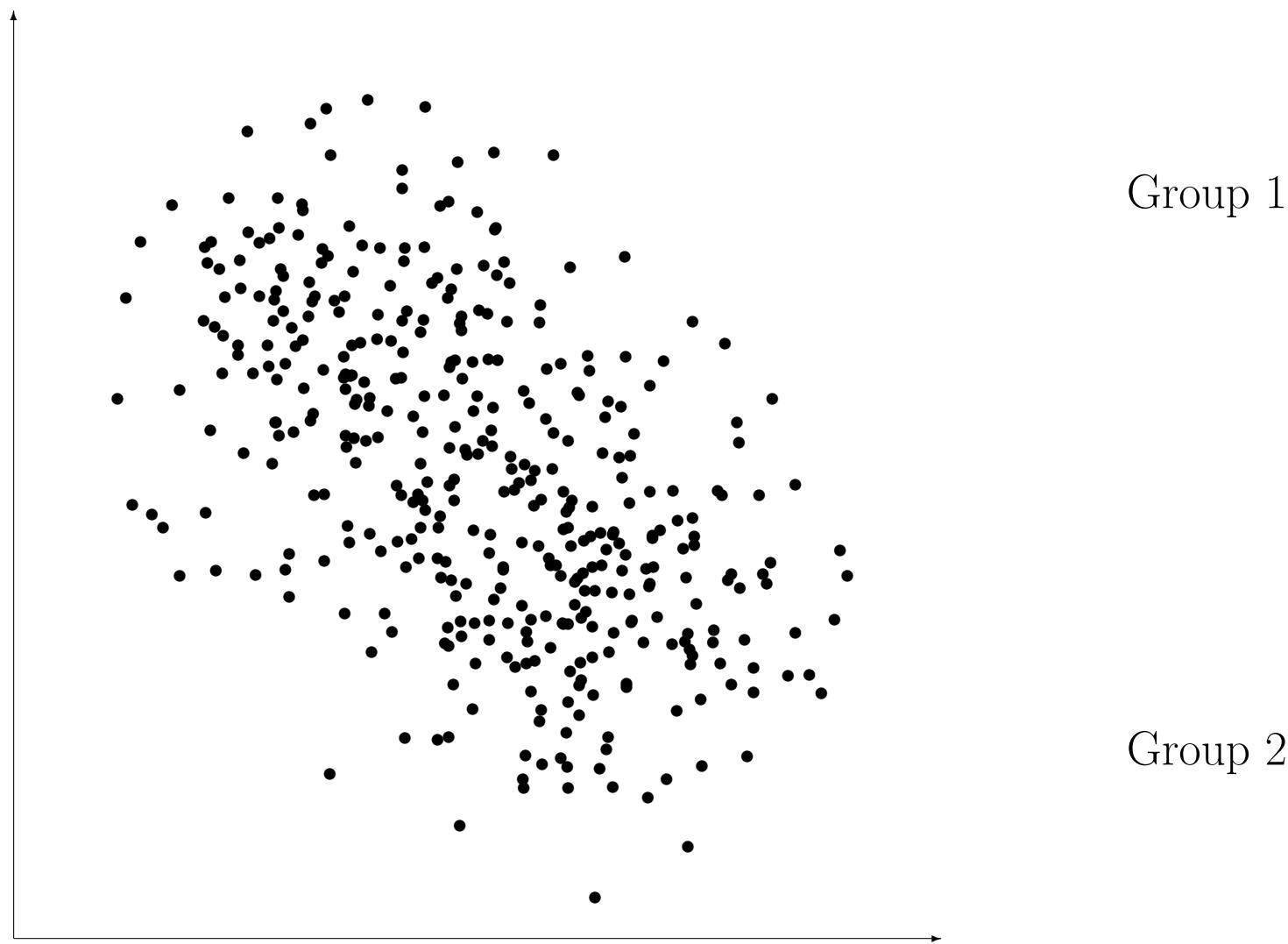
- The same formulae hold conditionally, i.e.

$$P(A \wedge B | C) = P(A | C) \cdot P(B | C) \quad \text{and}$$

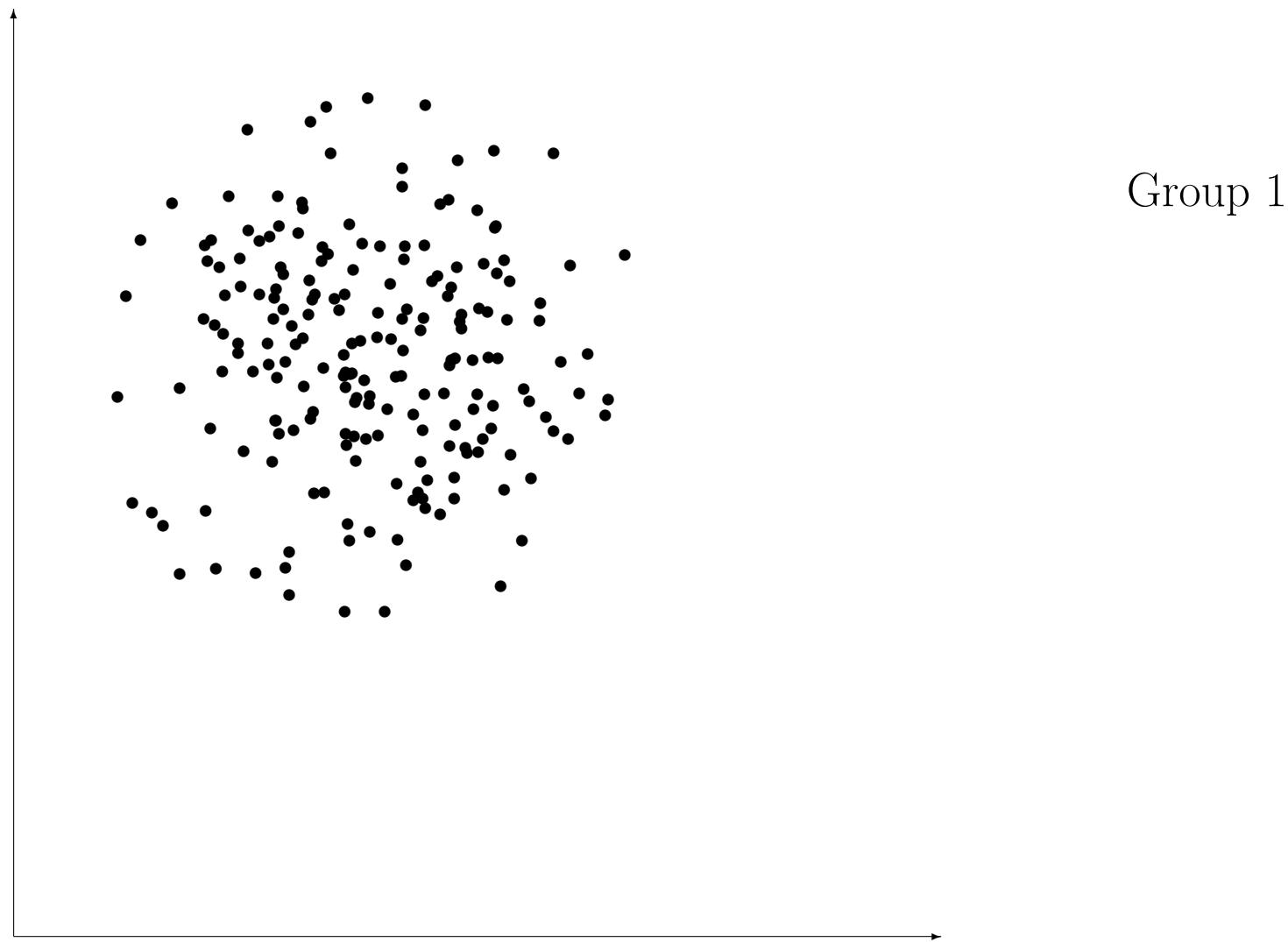
$$P(A | B, C) = P(A | C).$$

- **Conditional independence allows us to cancel some conditions.**

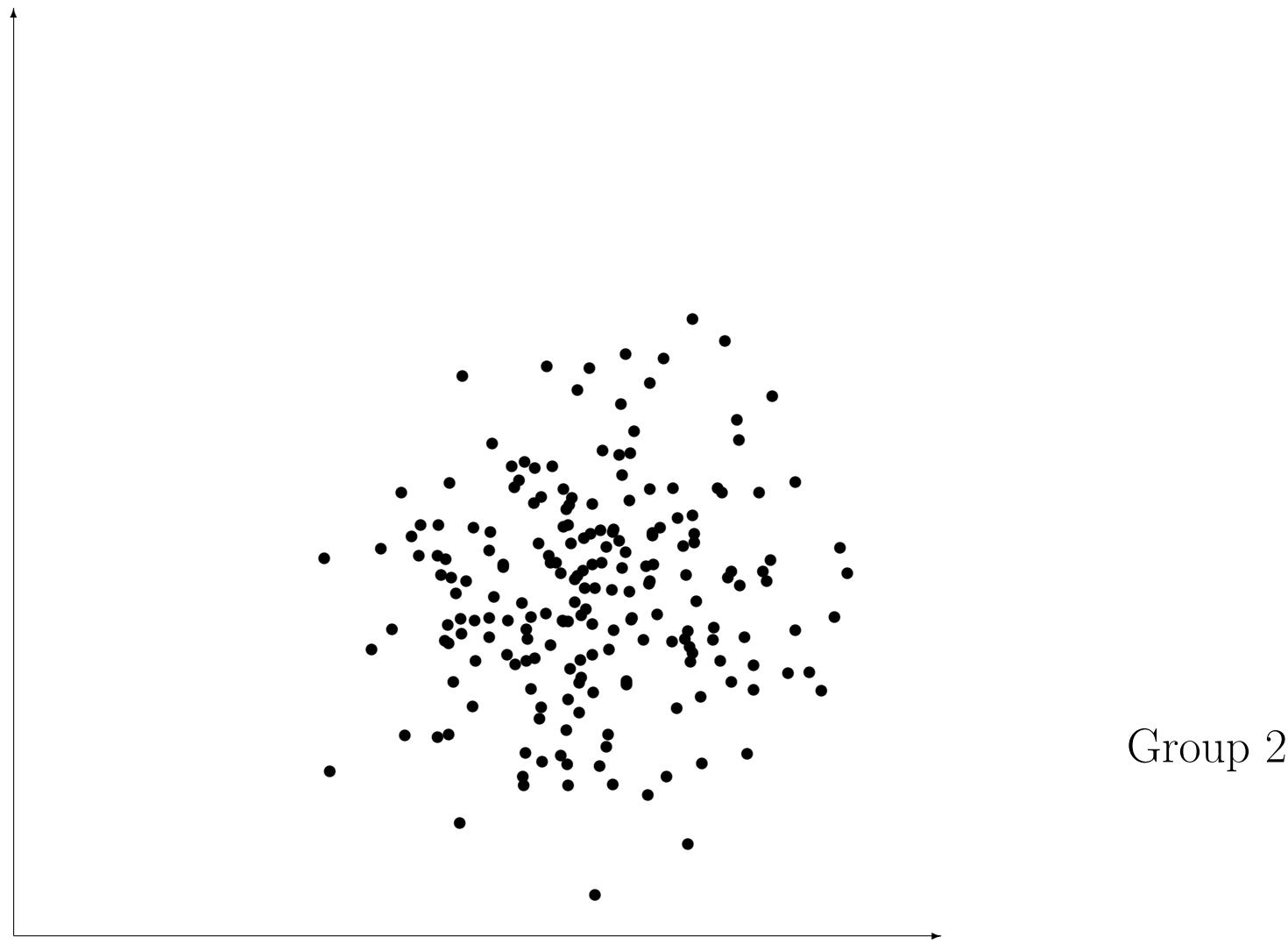
Conditional Independence: An Example



Conditional Independence: An Example



Conditional Independence: An Example



Naive Bayes Classifiers

- Consequence: Manageable amount of data to store.
Store distributions $P(C = c_i)$ and $\forall 1 \leq k \leq m : P(A_k = a_k | C = c_i)$.
- It is not necessary to compute p_0 explicitly, because it can be computed implicitly by normalizing the computed values to sum 1.

Estimation of Probabilities:

- **Nominal/Symbolic Attributes**

$$\hat{P}(A_k = a_k | C = c_i) = \frac{\#(A_k = a_k, C = c_i) + \gamma}{\#(C = c_i) + n_{A_k} \gamma}$$

γ is called **Laplace correction**.

$\gamma = 0$: Maximum likelihood estimation.

Common choices: $\gamma = 1$ or $\gamma = \frac{1}{2}$.

Naive Bayes Classifiers

Estimation of Probabilities:

- **Metric/Numeric Attributes:** Assume a normal distribution.

$$P(A_k = a_k \mid C = c_i) = \frac{1}{\sqrt{2\pi}\sigma_k(c_i)} \exp\left(-\frac{(a_k - \mu_k(c_i))^2}{2\sigma_k^2(c_i)}\right)$$

- Estimate of mean value

$$\hat{\mu}_k(c_i) = \frac{1}{\#(C = c_i)} \sum_{j=1}^{\#(C=c_i)} a_k(j)$$

- Estimate of variance

$$\hat{\sigma}_k^2(c_i) = \frac{1}{\xi} \sum_{j=1}^{\#(C=c_i)} (a_k(j) - \hat{\mu}_k(c_i))^2$$

$\xi = \#(C = c_i)$: Maximum likelihood estimation

$\xi = \#(C = c_i) - 1$: Unbiased estimation

Naive Bayes Classifiers: Simple Example 1

No	Sex	Age	Blood pr.	Drug
1	male	20	normal	A
2	female	73	normal	B
3	female	37	high	A
4	male	33	low	B
5	female	48	high	A
6	male	29	normal	A
7	female	52	normal	B
8	male	42	low	B
9	male	61	normal	B
10	female	30	normal	A
11	female	26	low	B
12	male	54	high	A

$P(\text{Drug})$	A	B
	0.5	0.5
$P(\text{Sex} \mid \text{Drug})$	A	B
male	0.5	0.5
female	0.5	0.5
$P(\text{Age} \mid \text{Drug})$	A	B
μ	36.3	47.8
σ^2	161.9	311.0
$P(\text{Blood Pr.} \mid \text{Drug})$	A	B
low	0	0.5
normal	0.5	0.5
high	0.5	0

A simple database and estimated (conditional) probability distributions.

Naive Bayes Classifiers: Simple Example 1

$$P(\text{Drug A} \mid \text{male, 61, normal})$$

$$\begin{aligned} &= c_1 \cdot P(\text{Drug A}) \cdot P(\text{male} \mid \text{Drug A}) \cdot P(61 \mid \text{Drug A}) \cdot P(\text{normal} \mid \text{Drug A}) \\ &\approx c_1 \cdot 0.5 \cdot 0.5 \cdot 0.004787 \cdot 0.5 = c_1 \cdot 5.984 \cdot 10^{-4} = 0.219 \end{aligned}$$

$$P(\text{Drug A} \mid \text{male, 61, normal})$$

$$\begin{aligned} &= c_1 \cdot P(\text{Drug B}) \cdot P(\text{male} \mid \text{Drug B}) \cdot P(61 \mid \text{Drug B}) \cdot P(\text{normal} \mid \text{Drug B}) \\ &\approx c_1 \cdot 0.5 \cdot 0.5 \cdot 0.017120 \cdot 0.5 = c_1 \cdot 2.140 \cdot 10^{-3} = 0.781 \end{aligned}$$

$$P(\text{Drug A} \mid \text{female, 30, normal})$$

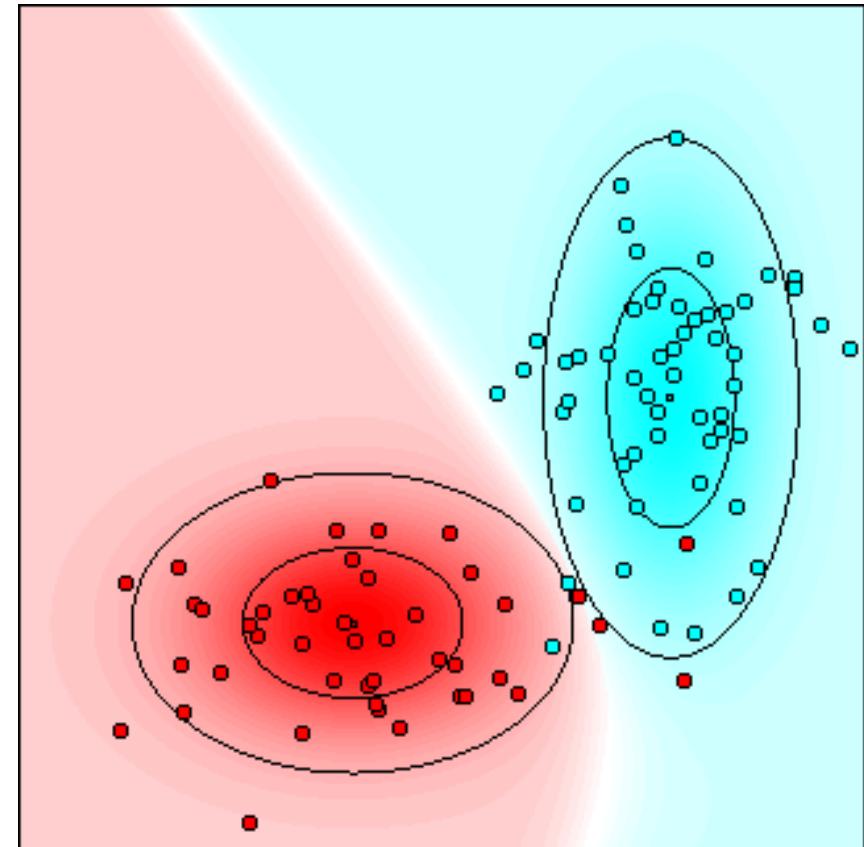
$$\begin{aligned} &= c_2 \cdot P(\text{Drug A}) \cdot P(\text{female} \mid \text{Drug A}) \cdot P(30 \mid \text{Drug A}) \cdot P(\text{normal} \mid \text{Drug A}) \\ &\approx c_2 \cdot 0.5 \cdot 0.5 \cdot 0.027703 \cdot 0.5 = c_2 \cdot 3.471 \cdot 10^{-3} = 0.671 \end{aligned}$$

$$P(\text{Drug A} \mid \text{female, 30, normal})$$

$$\begin{aligned} &= c_2 \cdot P(\text{Drug B}) \cdot P(\text{female} \mid \text{Drug B}) \cdot P(30 \mid \text{Drug B}) \cdot P(\text{normal} \mid \text{Drug B}) \\ &\approx c_2 \cdot 0.5 \cdot 0.5 \cdot 0.013567 \cdot 0.5 = c_2 \cdot 1.696 \cdot 10^{-3} = 0.329 \end{aligned}$$

Naive Bayes Classifiers: Simple Example 2

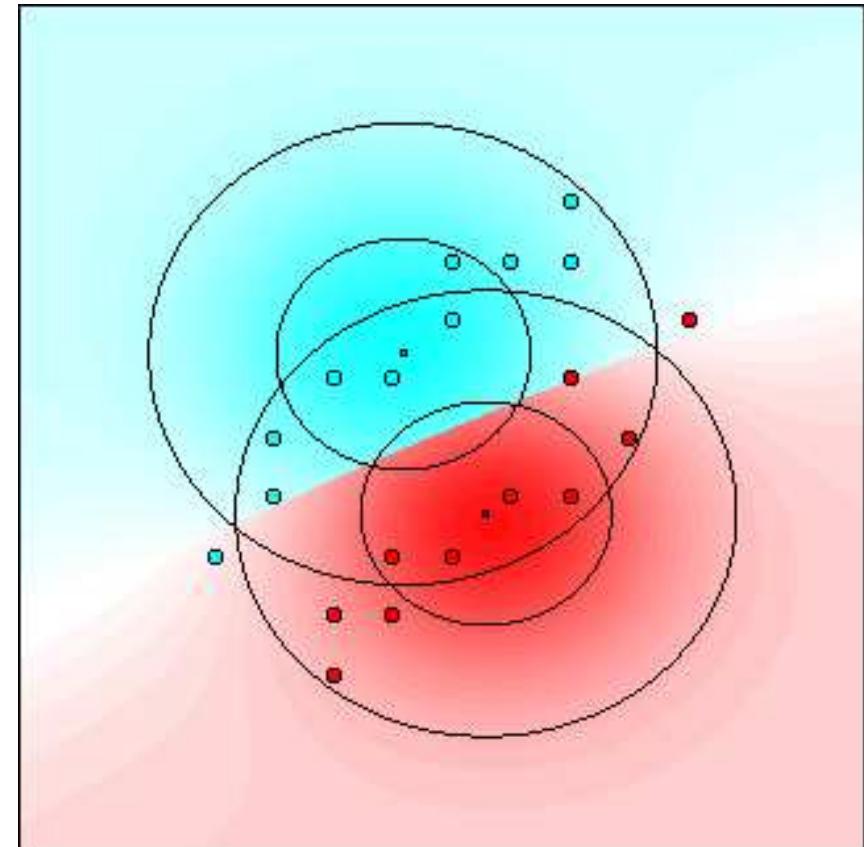
- 100 data points, 2 classes
- Small squares: mean values
- Inner ellipses:
one standard deviation
- Outer ellipses:
two standard deviations
- Classes overlap:
classification is not perfect



Naive Bayes Classifier

Naive Bayes Classifiers: Simple Example 3

- 20 data points, 2 classes
- Small squares: mean values
- Inner ellipses:
one standard deviation
- Outer ellipses:
two standard deviations
- Attributes are not conditionally
independent given the class



Naive Bayes Classifier

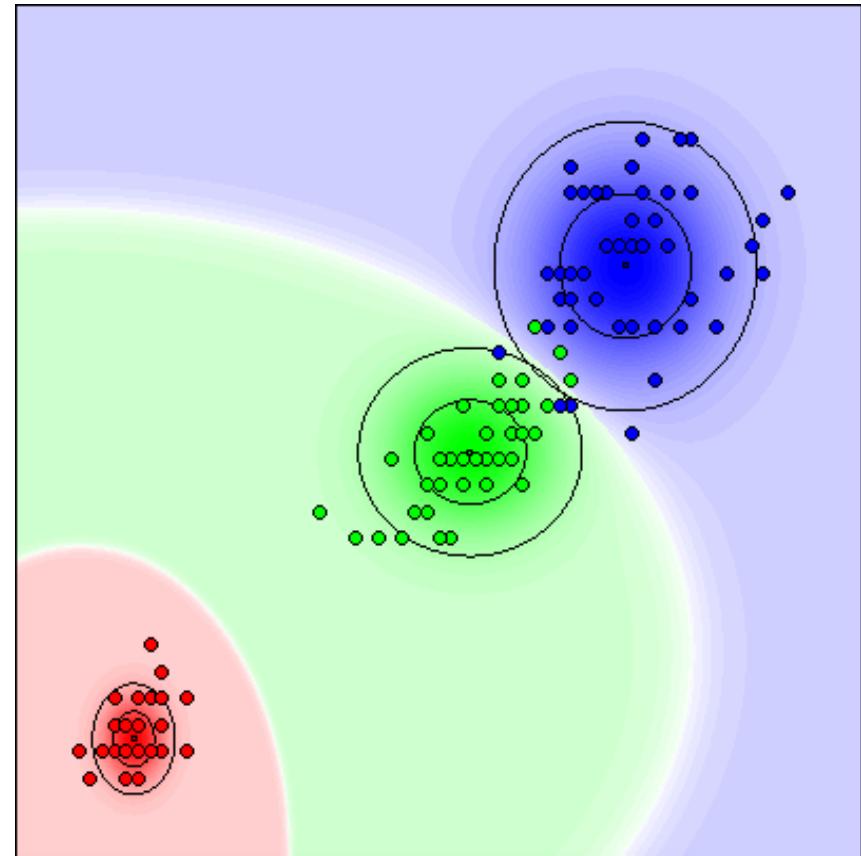
Reminder: The Iris Data

pictures not available in online version

- Collected by Edgar Anderson on the Gaspé Peninsula (Canada).
- First analyzed by Ronald Aylmer Fisher (famous statistician).
- 150 cases in total, 50 cases per Iris flower type.
- Measurements of sepal length and width and petal length and width (in cm).
- Most famous data set in pattern recognition and data analysis.

Naive Bayes Classifiers: Iris Data

- 150 data points, 3 classes
 - Iris setosa (red)
 - Iris versicolor (green)
 - Iris virginica (blue)
- Shown: 2 out of 4 attributes
 - sepal length
 - sepal width
 - petal length (horizontal)
 - petal width (vertical)
- 6 misclassifications on the training data (with all 4 attributes)



Naive Bayes Classifier

Full Bayes Classifiers

- Restricted to metric/numeric attributes (only the class is nominal/symbolic).
- **Simplifying Assumption:**
Each class can be described by a multivariate normal distribution.

$$\begin{aligned} & f(A_1 = a_1, \dots, A_m = a_m \mid C = c_i) \\ &= \frac{1}{\sqrt{(2\pi)^m |\Sigma_i|}} \exp\left(-\frac{1}{2}(\vec{a} - \vec{\mu}_i)^\top \Sigma_i^{-1} (\vec{a} - \vec{\mu}_i)\right) \end{aligned}$$

$\vec{\mu}_i$: mean value vector for class c_i

Σ_i : covariance matrix for class c_i

- Intuitively: Each class has a bell-shaped probability density.
- Naive Bayes classifiers: Covariance matrices are diagonal matrices.
(Details about this relation are given below.)

Full Bayes Classifiers

Estimation of Probabilities:

- Estimate of mean value vector

$$\hat{\vec{\mu}}_i = \frac{1}{\#(C = c_i)} \sum_{j=1}^{\#(C=c_i)} \vec{a}(j)$$

- Estimate of covariance matrix

$$\hat{\Sigma}_i = \frac{1}{\xi} \sum_{j=1}^{\#(C=c_i)} (\vec{a}(j) - \hat{\vec{\mu}}_i) (\vec{a}(j) - \hat{\vec{\mu}}_i)^\top$$

$\xi = \#(C = c_i)$: Maximum likelihood estimation

$\xi = \#(C = c_i) - 1$: Unbiased estimation

\vec{x}^\top denotes the transpose of the vector \vec{x} .

$\vec{x}\vec{x}^\top$ is the so-called **outer product** or **matrix product** of \vec{x} with itself.

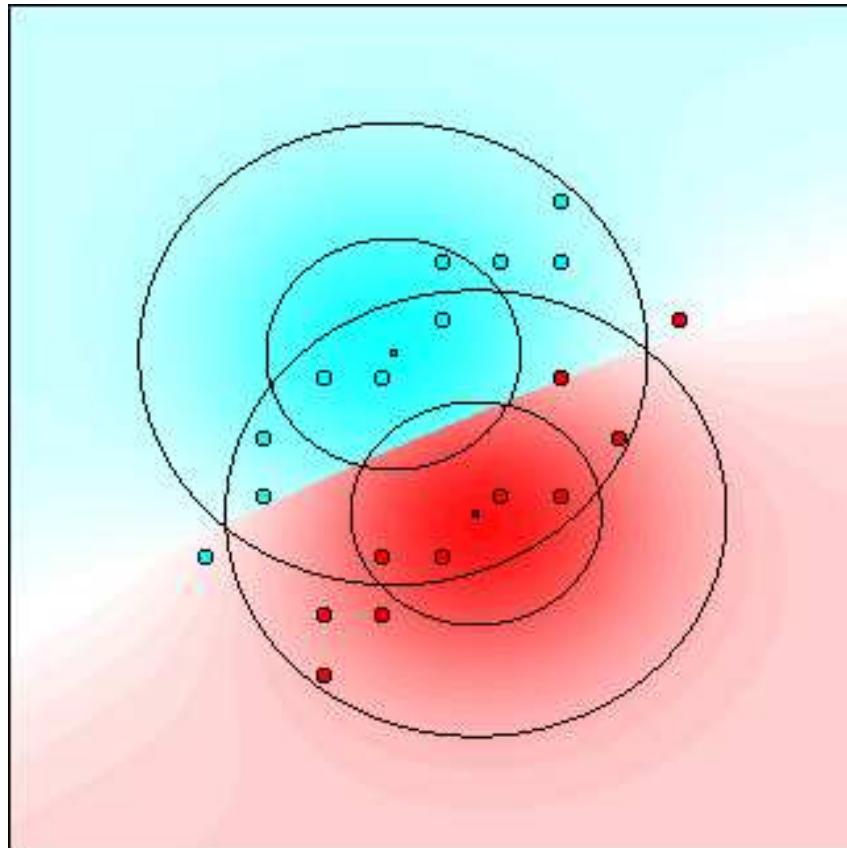
Comparison of Naive and Full Bayes Classifiers

Naive Bayes classifiers for metric/numeric data are equivalent to full Bayes classifiers with diagonal covariance matrices:

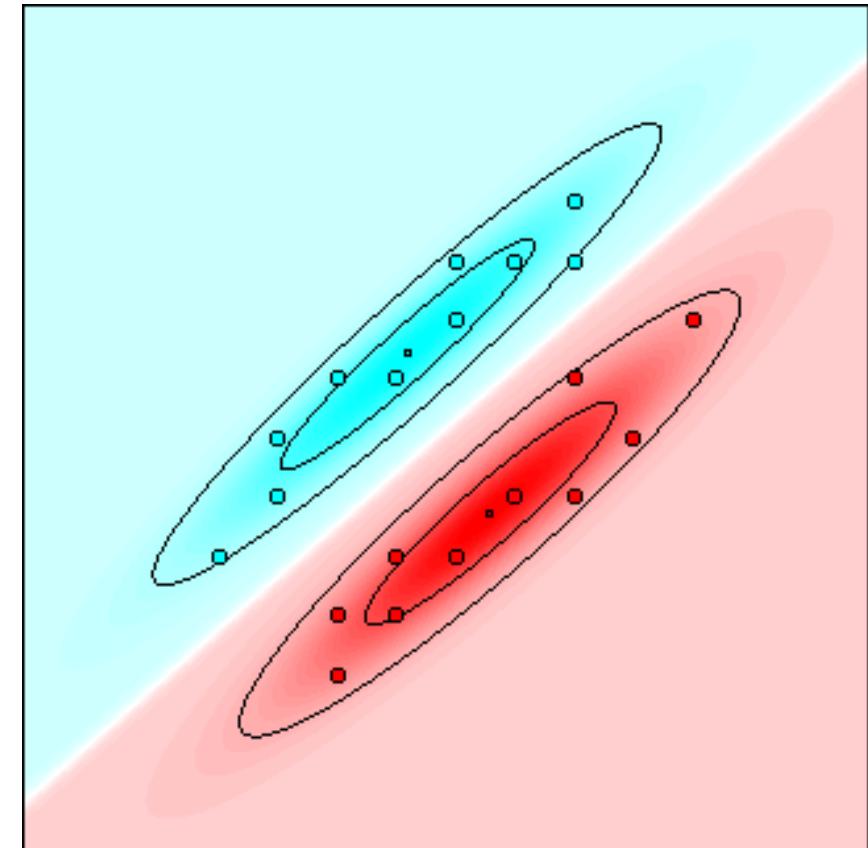
$$\begin{aligned} & f(A_1 = a_1, \dots, A_m = a_m \mid C = c_i) \\ &= \frac{1}{\sqrt{(2\pi)^m |\Sigma_i|}} \cdot \exp\left(-\frac{1}{2}(\vec{a} - \vec{\mu}_i)^\top \Sigma_i^{-1} (\vec{a} - \vec{\mu}_i)\right) \\ &= \frac{1}{\sqrt{(2\pi)^m \prod_{k=1}^m \sigma_{i,k}^2}} \cdot \exp\left(-\frac{1}{2}(\vec{a} - \vec{\mu}_i)^\top \text{diag}(\sigma_{i,1}^{-2}, \dots, \sigma_{i,m}^{-2}) (\vec{a} - \vec{\mu}_i)\right) \\ &= \frac{1}{\prod_{k=1}^m \sqrt{2\pi \sigma_{i,k}^2}} \cdot \exp\left(-\frac{1}{2} \sum_{k=1}^m \frac{(a_k - \mu_{i,k})^2}{\sigma_{i,k}^2}\right) \\ &= \prod_{k=1}^m \frac{1}{\sqrt{2\pi \sigma_{i,k}^2}} \cdot \exp\left(-\frac{(a_k - \mu_{i,k})^2}{2\sigma_{i,k}^2}\right) \stackrel{\cong}{=} \prod_{k=1}^m f(A_k = a_k \mid C = c_i), \end{aligned}$$

where $f(A_k = a_k \mid C = c_i)$ are the density functions of a naive Bayes classifier.

Comparison of Naive and Full Bayes Classifiers



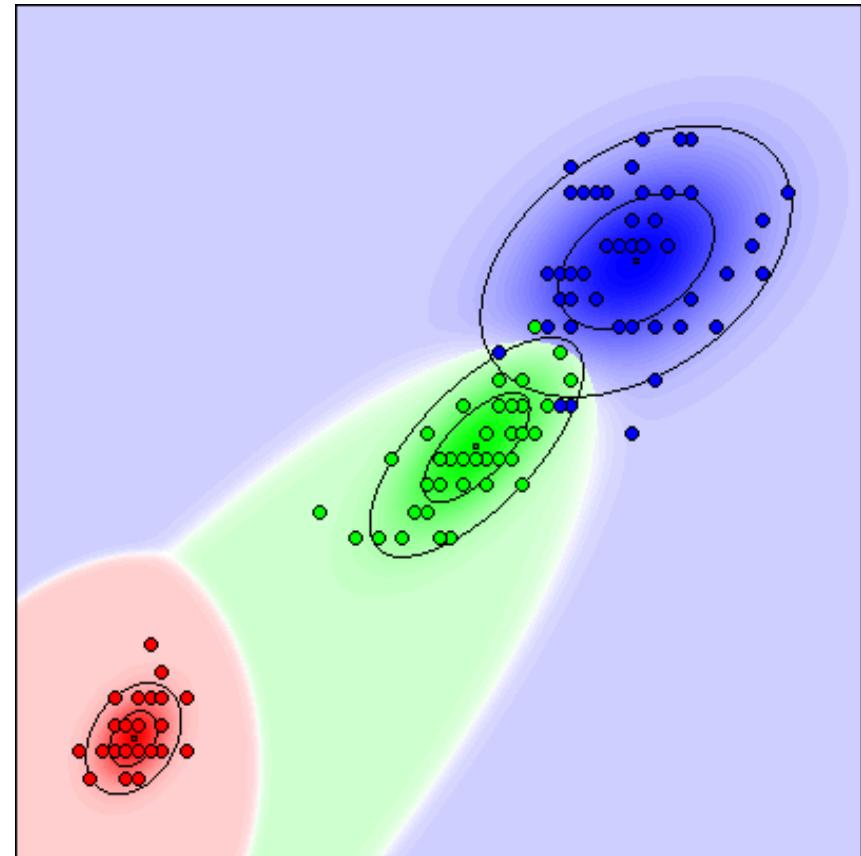
Naive Bayes Classifier



Full Bayes Classifier

Full Bayes Classifiers: Iris Data

- 150 data points, 3 classes
 - Iris setosa (red)
 - Iris versicolor (green)
 - Iris virginica (blue)
- Shown: 2 out of 4 attributes
 - sepal length
 - sepal width
 - petal length (horizontal)
 - petal width (vertical)
- 2 misclassifications on the training data (with all 4 attributes)



Full Bayes Classifier

Tree-Augmented Naive Bayes Classifiers

- A naive Bayes classifier can be seen as a special **Bayesian network**.
- Intuitively, Bayesian networks are a graphical language for expressing conditional independence statements: A directed acyclic graph encodes, by a vertex separation criterion, which conditional independence statements hold in the joint probability distribution on the space spanned by the vertex attributes.

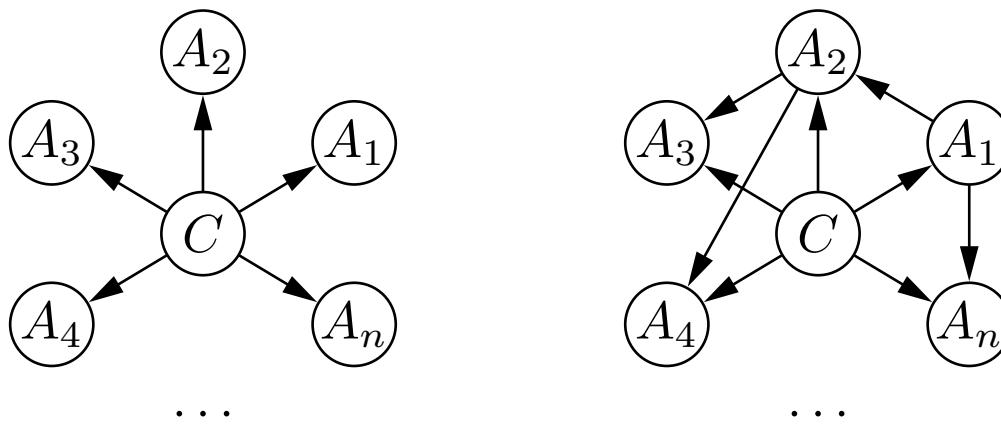
Definition (d -separation):

Let $\vec{G} = (\vec{V}, \vec{E})$ be a directed acyclic graph and X , Y , and Z three disjoint subsets of vertices. Z **d-separates** X and Y in \vec{G} , written $\langle X \mid Z \mid Y \rangle_{\vec{G}}$, iff there is no path from a vertex in X to a vertex in Y along which the following two conditions hold:

1. every vertex with converging edges (from its predecessor and its successor on the path) either is in Z or has a descendant in Z ,
2. every other vertex is not in Z .

A path satisfying the conditions above is said to be **active**, otherwise it is said to be **blocked** (by Z); so separation means that all paths are blocked.

Tree-Augmented Naive Bayes Classifiers



- If in a directed acyclic graph all paths from a vertex set X to a vertex set Y are blocked by a vertex set Z (according to d -separation), this expresses that the conditional independence $X \perp\!\!\!\perp Y | Z$ holds in the probability distribution that is described by a Bayesian network having this graph structure.
- A star-like network, with the class attribute in the middle, represents a naive Bayes classifier: All paths are blocked by the class attribute C .
- The strong conditional independence assumptions can be mitigated by allowing for additional edges between attributes. Restricting these edges to a (directed) tree allows for efficient learning (**tree-augmented naive Bayes classifiers**).

Summary Bayes Classifiers

- **Probabilistic Classification:** Assign the most probable class.
- **Bayes' Rule:** “Invert” the conditional class probabilities.
- **Naive Bayes Classifiers**
 - Simplifying Assumption:
Attributes are conditionally independent given the class.
 - Can handle nominal/symbolic as well as metric/numeric attributes.
- **Full Bayes Classifiers**
 - Simplifying Assumption:
Each class can be described by a multivariate normal distribution.
 - Can handle only metric/numeric attributes.
- **Tree-Augmented Naive Bayes Classifiers**
 - Mitigate the strong conditional independence assumptions.

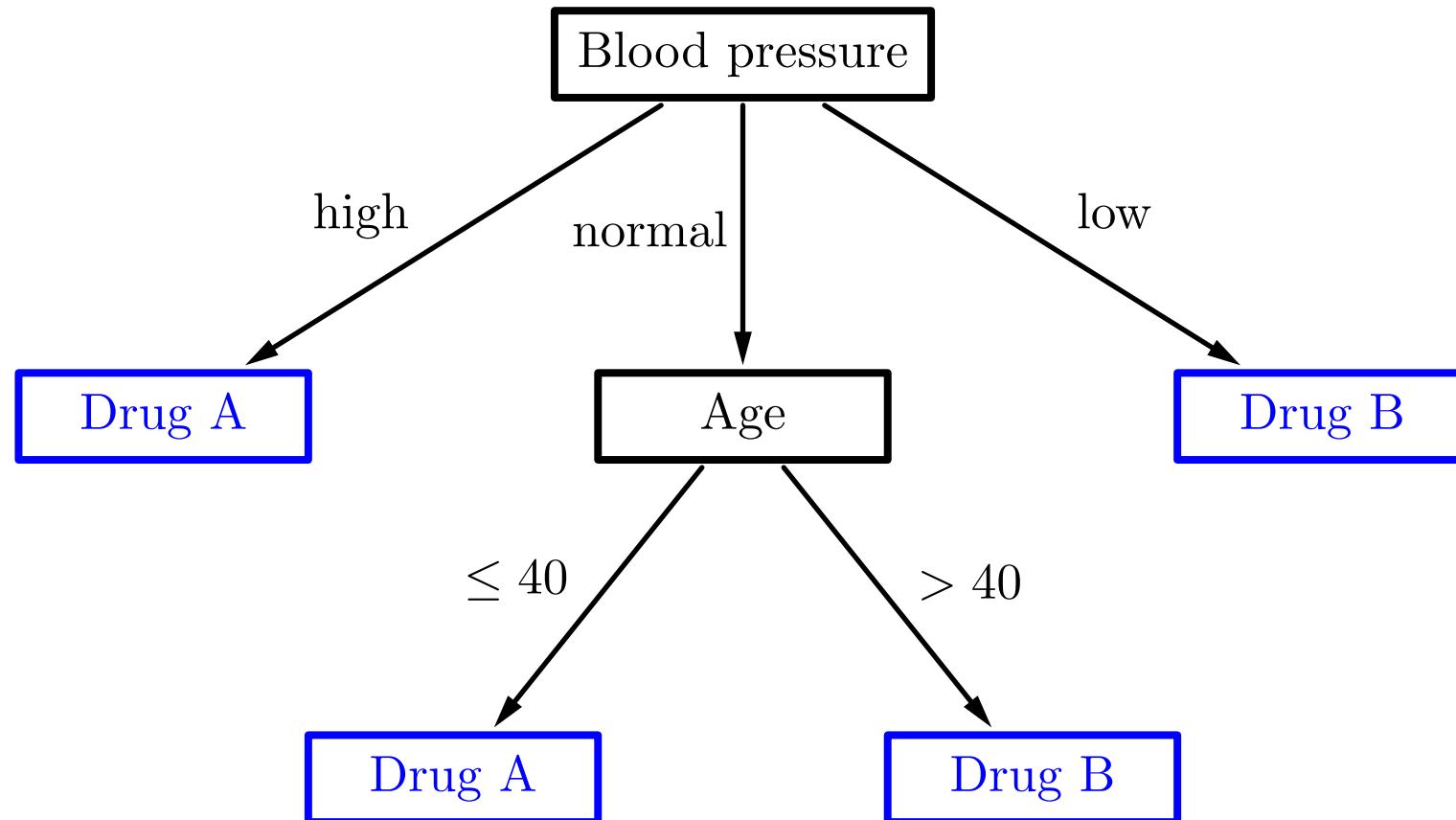
Decision and Regression Trees

Decision and Regression Trees

- **Classification with a Decision Tree**
- **Top-down Induction of Decision Trees**
 - A simple example
 - The general algorithm
 - Attribute selection measures
 - Treatment of numeric attributes and missing values
- **Pruning Decision Trees**
 - General approaches
 - A simple example
- **Regression Trees**
- **Summary**

A Very Simple Decision Tree

Assignment of a drug to a patient:



Classification with a Decision Tree

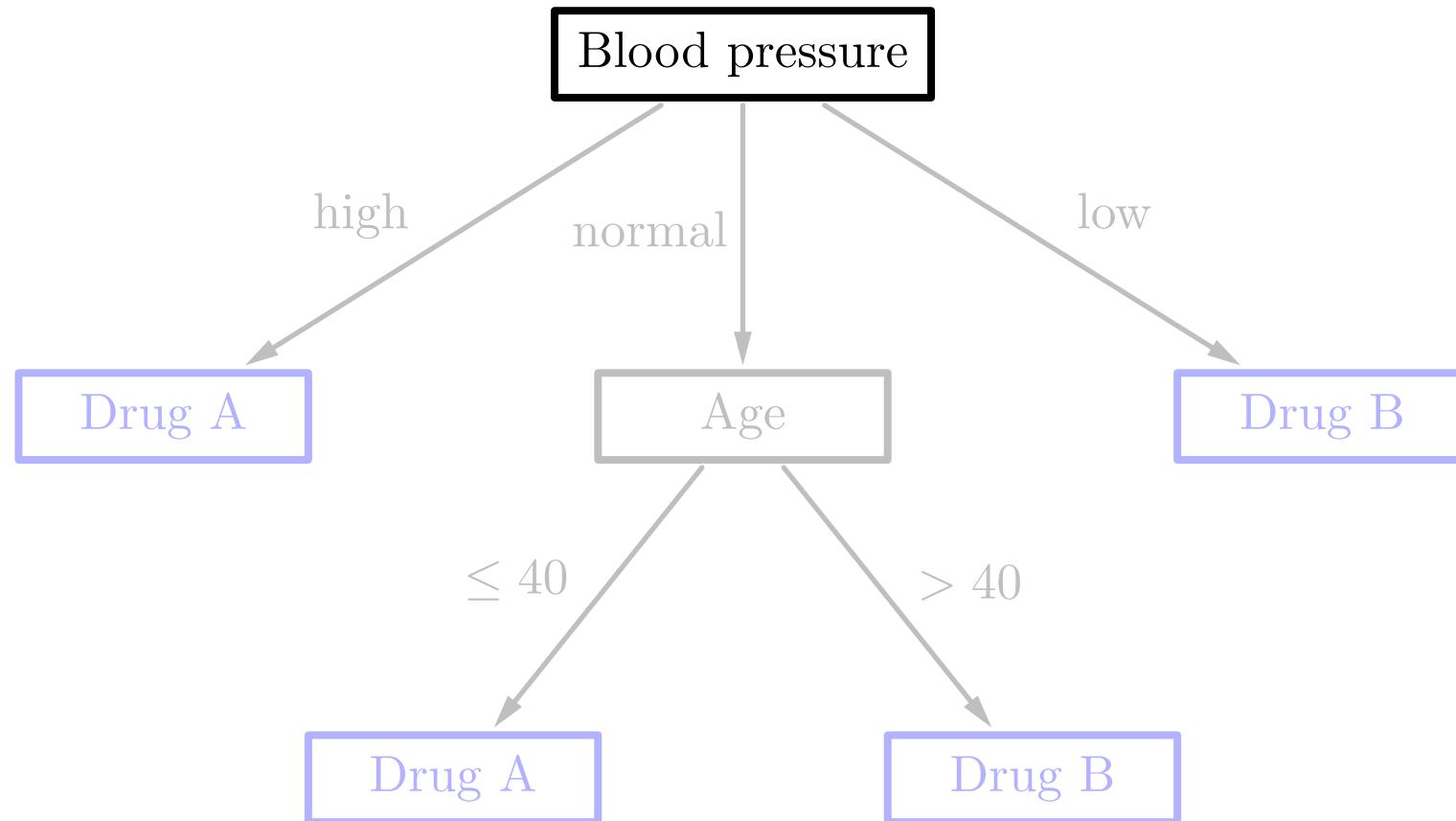
Recursive Descent:

- Start at the root node.
- If the current node is an **leaf node**:
 - Return the class assigned to the node.
- If the current node is an **inner node**:
 - Test the attribute associated with the node.
 - Follow the branch labeled with the outcome of the test.
 - Apply the algorithm recursively.

Intuitively: Follow the path corresponding to the case to be classified.

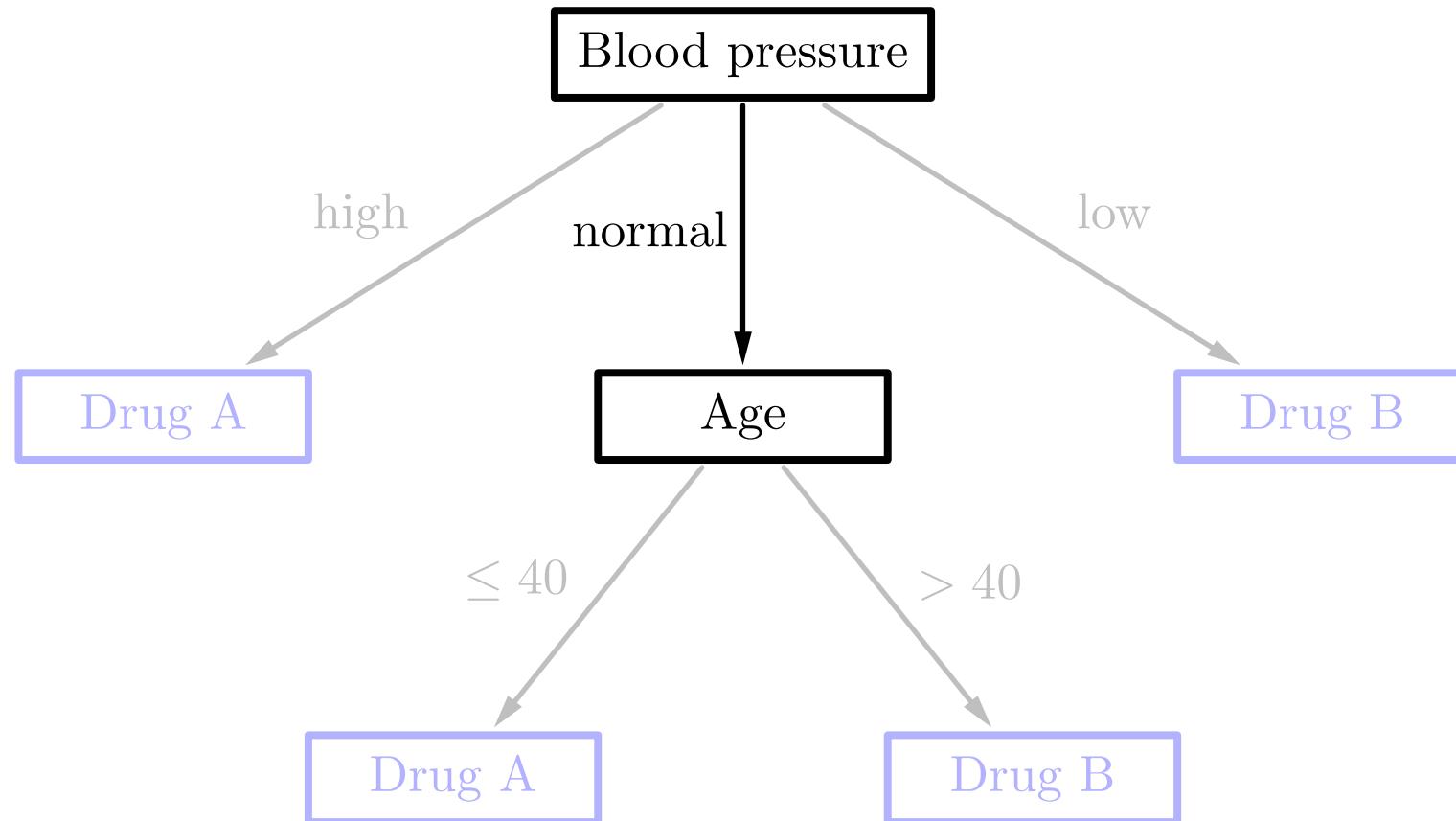
Classification in the Example

Assignment of a drug to a patient:



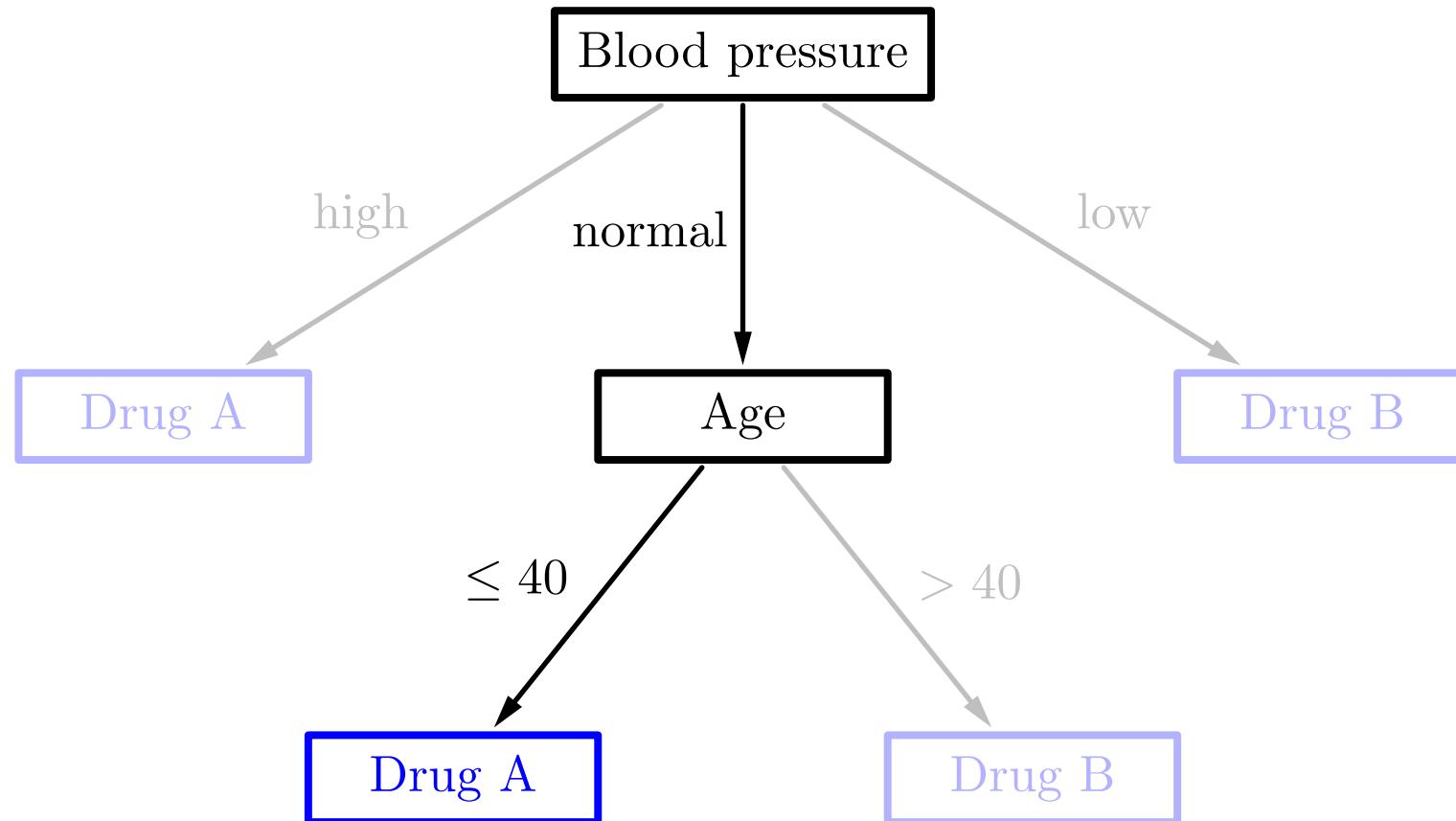
Classification in the Example

Assignment of a drug to a patient:



Classification in the Example

Assignment of a drug to a patient:



Induction of Decision Trees

- **Top-down approach**
 - Build the decision tree from top to bottom
(from the root to the leaves).
- **Greedy Selection of a Test Attribute**
 - Compute an evaluation measure for all attributes.
 - Select the attribute with the best evaluation.
- **Divide and Conquer / Recursive Descent**
 - Divide the example cases according to the values of the test attribute.
 - Apply the procedure recursively to the subsets.
 - Terminate the recursion if
 - all cases belong to the same class
 - no more test attributes are available

Induction of a Decision Tree: Example

Patient database

- 12 example cases
- 3 descriptive attributes
- 1 class attribute

Assignment of drug

(without patient attributes)

always drug A or always drug B:

50% correct (in 6 of 12 cases)

No	Gender	Age	Blood pr.	Drug
1	male	20	normal	A
2	female	73	normal	B
3	female	37	high	A
4	male	33	low	B
5	female	48	high	A
6	male	29	normal	A
7	female	52	normal	B
8	male	42	low	B
9	male	61	normal	B
10	female	30	normal	A
11	female	26	low	B
12	male	54	high	A

Induction of a Decision Tree: Example

Gender of the patient

- Division w.r.t. male/female.

Assignment of drug

male: 50% correct (in 3 of 6 cases)

female: 50% correct (in 3 of 6 cases)

total: **50% correct** (in 6 of 12 cases)

No	Gender	Drug
1	male	A
6	male	A
12	male	A
4	male	B
8	male	B
9	male	B
3	female	A
5	female	A
10	female	A
2	female	B
7	female	B
11	female	B

Induction of a Decision Tree: Example

Age of the patient

- Sort according to age.
- Find best age split.
here: ca. 40 years

Assignment of drug

≤ 40 : A 67% correct (in 4 of 6 cases)

> 40 : B 67% correct (in 4 of 6 cases)

total: **67% correct** (in 8 of 12 cases)

No	Age	Drug
1	20	A
11	26	B
6	29	A
10	30	A
4	33	B
3	37	A
8	42	B
5	48	A
7	52	B
12	54	A
9	61	B
2	73	B

Induction of a Decision Tree: Example

Blood pressure of the patient

- Division w.r.t. high/normal/low.

Assignment of drug

high: A 100% correct (in 3 of 3 cases)

normal: 50% correct (in 3 of 6 cases)

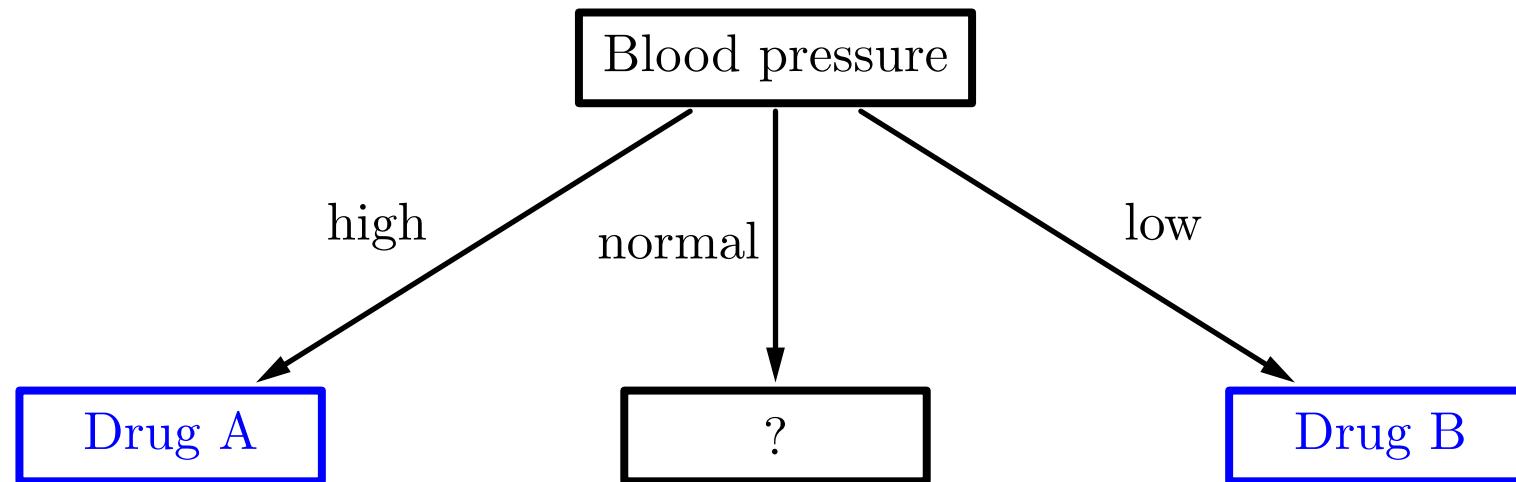
low: B 100% correct (in 3 of 3 cases)

total: **75% correct** (in 9 of 12 cases)

No	Blood pr.	Drug
3	high	A
5	high	A
12	high	A
1	normal	A
6	normal	A
10	normal	A
2	normal	B
7	normal	B
9	normal	B
4	low	B
8	low	B
11	low	B

Induction of a Decision Tree: Example

Current Decision Tree:



Induction of a Decision Tree: Example

Blood pressure and gender

- Only patients with normal blood pressure.
- Division w.r.t. male/female.

Assignment of drug

male: A 67% correct (2 of 3)

female: B 67% correct (2 of 3)

total: **67% correct** (4 of 6)

No	Blood pr.	Gender	Drug
3	high		A
5	high		A
12	high		A
1	normal	male	A
6	normal	male	A
9	normal	male	B
2	normal	female	B
7	normal	female	B
10	normal	female	A
4	low		B
8	low		B
11	low		B

Induction of a Decision Tree: Example

Blood pressure and age

- Only patients with normal blood pressure.
- Sort according to age.
- Find best age split.
here: ca. 40 years

Assignment of drug

≤ 40 : A 100% correct (3 of 3)

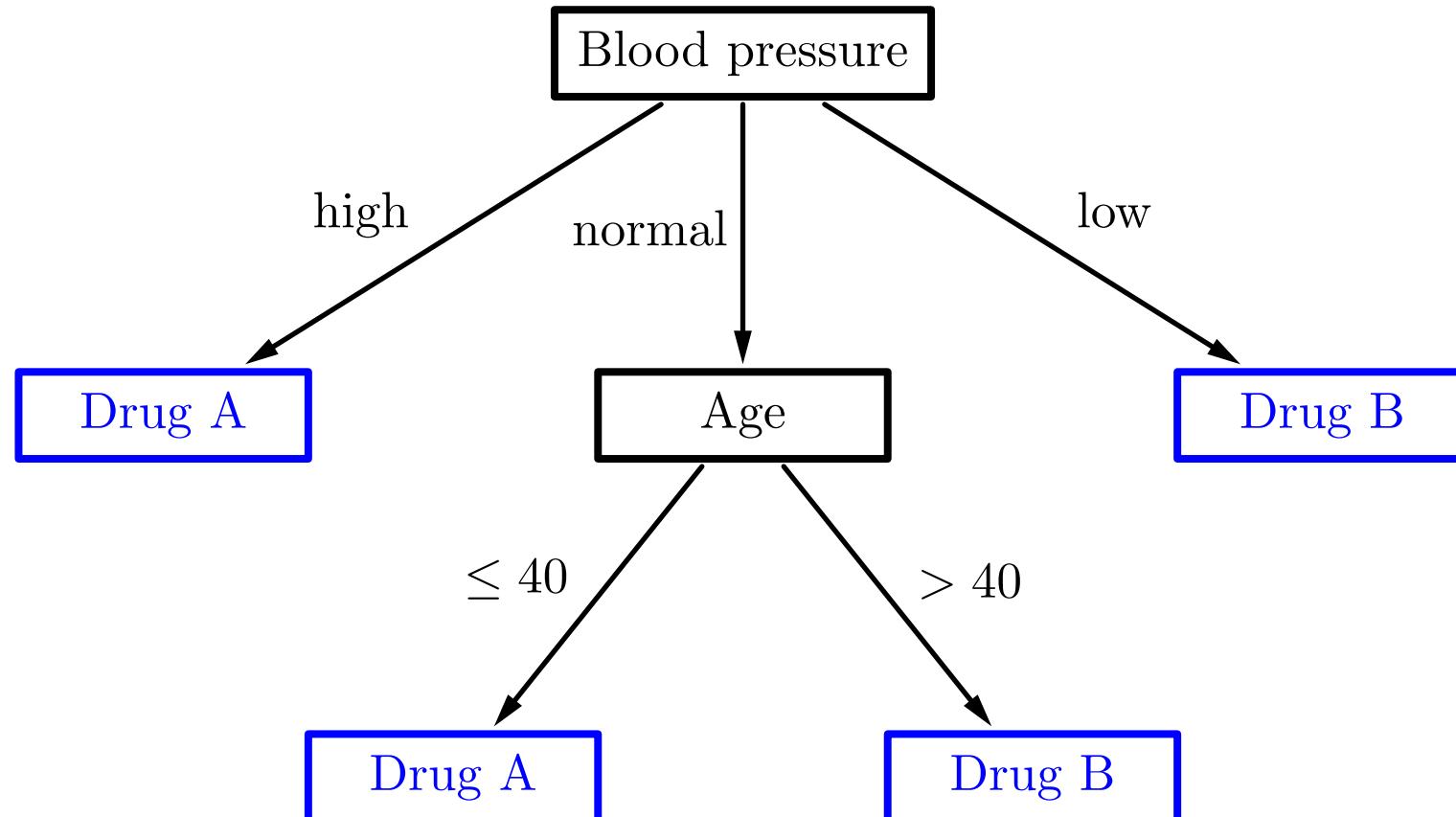
> 40 : B 100% correct (3 of 3)

total: **100% correct** (6 of 6)

No	Blood pr.	Age	Drug
3	high		A
5	high		A
12	high		A
1	normal	20	A
6	normal	29	A
10	normal	30	A
7	normal	52	B
9	normal	61	B
2	normal	73	B
11	low		B
4	low		B
8	low		B

Result of Decision Tree Induction

Assignment of a drug to a patient:



Decision Tree Induction: Notation

S	a set of case or object descriptions
C	the class attribute
$A^{(1)}, \dots, A^{(m)}$	other attributes (index dropped in the following)
$\text{dom}(C)$	$= \{c_1, \dots, c_{n_C}\}$, n_C : number of classes
$\text{dom}(A)$	$= \{a_1, \dots, a_{n_A}\}$, n_A : number of attribute values
$N_{..}$	total number of case or object descriptions i.e. $N_{..} = S $
$N_{i.}$	absolute frequency of the class c_i
$N_{.j}$	absolute frequency of the attribute value a_j
N_{ij}	absolute frequency of the combination of the class c_i and the attribute value a_j . It is $N_{i.} = \sum_{j=1}^{n_A} N_{ij}$ and $N_{.j} = \sum_{i=1}^{n_C} N_{ij}$.
$p_{i.}$	relative frequency of the class c_i , $p_{i.} = \frac{N_{i.}}{N_{..}}$
$p_{.j}$	relative frequency of the attribute value a_j , $p_{.j} = \frac{N_{.j}}{N_{..}}$
p_{ij}	relative frequency of the combination of class c_i and attribute value a_j , $p_{ij} = \frac{N_{ij}}{N_{..}}$
$p_{i j}$	relative frequency of the class c_i in cases having attribute value a_j , $p_{i j} = \frac{N_{ij}}{N_{.j}} = \frac{p_{ij}}{p_{.j}}$

Decision Tree Induction: General Algorithm

```
function grow_tree ( $S$  : set of cases) : node;  
begin  
     $best\_v := \text{WORTHLESS}$ ;  
    for all untested attributes  $A$  do  
        compute frequencies  $N_{ij}$ ,  $N_{i\cdot}$ ,  $N_{\cdot j}$  for  $1 \leq i \leq n_C$  and  $1 \leq j \leq n_A$ ;  
        compute value  $v$  of an evaluation measure using  $N_{ij}$ ,  $N_{i\cdot}$ ,  $N_{\cdot j}$ ;  
        if  $v > best\_v$  then  $best\_v := v$ ;  $best\_A := A$ ; end;  
    end  
    if  $best\_v = \text{WORTHLESS}$   
        then create leaf node  $x$ ;  
        assign majority class of  $S$  to  $x$ ;  
    else create test node  $x$ ;  
        assign test on attribute  $best\_A$  to  $x$ ;  
        for all  $a \in \text{dom}(best\_A)$  do  $x.\text{child}[a] := \text{grow\_tree}(S|_{best\_A=a})$ ; end;  
    end;  
    return  $x$ ;  
end;
```

Evaluation Measures

- Evaluation measure used in the above example:
rate of correctly classified example cases.
 - Advantage: simple to compute, easy to understand.
 - Disadvantage: works well only for two classes.
- If there are more than two classes, the rate of misclassified example cases **neglects a lot of the available information.**
 - Only the majority class—that is, the class occurring most often in (a subset of) the example cases—is really considered.
 - The distribution of the other classes has no influence. However, a good choice here can be important for deeper levels of the decision tree.
- **Therefore:** Study also other evaluation measures. Here:
 - **Information gain** and its various normalizations.
 - χ^2 **measure** (well-known in statistics).

An Information-theoretic Evaluation Measure

Information Gain (Kullback and Leibler 1951, Quinlan 1986)

Based on Shannon Entropy $H = - \sum_{i=1}^n p_i \log_2 p_i$ (Shannon 1948)

$$\begin{aligned} I_{\text{gain}}(C, A) &= H(C) - H(C|A) \\ &= \overbrace{- \sum_{i=1}^{n_C} p_{i\cdot} \log_2 p_{i\cdot}}^{} - \overbrace{\sum_{j=1}^{n_A} p_{\cdot j} \left(- \sum_{i=1}^{n_C} p_{i|j} \log_2 p_{i|j} \right)} \end{aligned}$$

$$H(C)$$

Entropy of the class distribution (C : class attribute)

$$H(C|A)$$

Expected entropy of the class distribution
if the value of the attribute A becomes known

$$H(C) - H(C|A)$$

Expected entropy reduction or *information gain*

Interpretation of Shannon Entropy

- Let $S = \{s_1, \dots, s_n\}$ be a finite set of alternatives having positive probabilities $P(s_i)$, $i = 1, \dots, n$, satisfying $\sum_{i=1}^n P(s_i) = 1$.
- **Shannon Entropy:**

$$H(S) = - \sum_{i=1}^n P(s_i) \log_2 P(s_i)$$

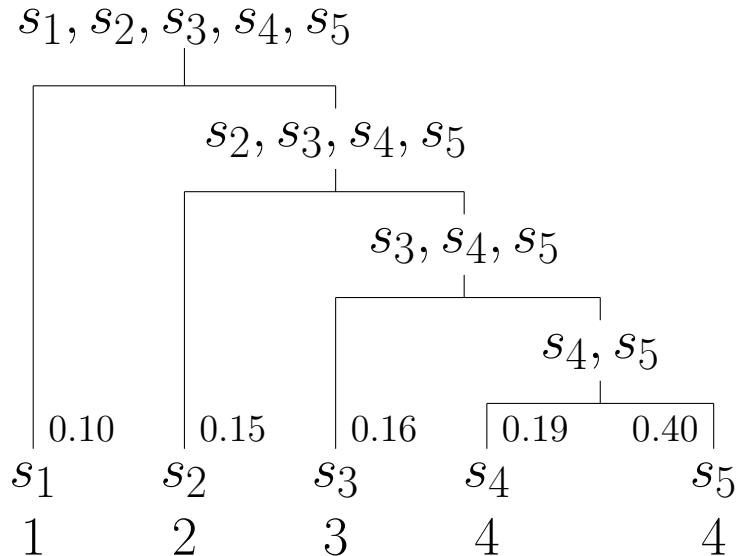
- Intuitively: **Expected number of yes/no questions that have to be asked in order to determine the obtaining alternative.**
 - Suppose there is an oracle, which knows the obtaining alternative, but responds only if the question can be answered with “yes” or “no”.
 - A better question scheme than asking for one alternative after the other can easily be found: Divide the set into two subsets of about equal size.
 - Ask for containment in an arbitrarily chosen subset.
 - Apply this scheme recursively \rightarrow number of questions bounded by $\lceil \log_2 n \rceil$.

Question/Coding Schemes

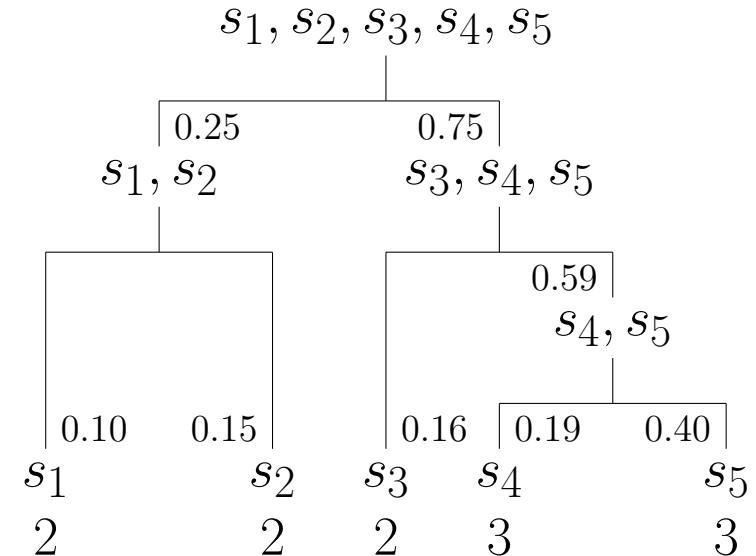
$$P(s_1) = 0.10, \quad P(s_2) = 0.15, \quad P(s_3) = 0.16, \quad P(s_4) = 0.19, \quad P(s_5) = 0.40$$

Shannon entropy: $-\sum_i P(s_i) \log_2 P(s_i) = 2.15$ bit/symbol

Linear Traversal



Equal Size Subsets



Code length: 3.24 bit/symbol

Code efficiency: 0.664

Code length: 2.59 bit/symbol

Code efficiency: 0.830

Question/Coding Schemes

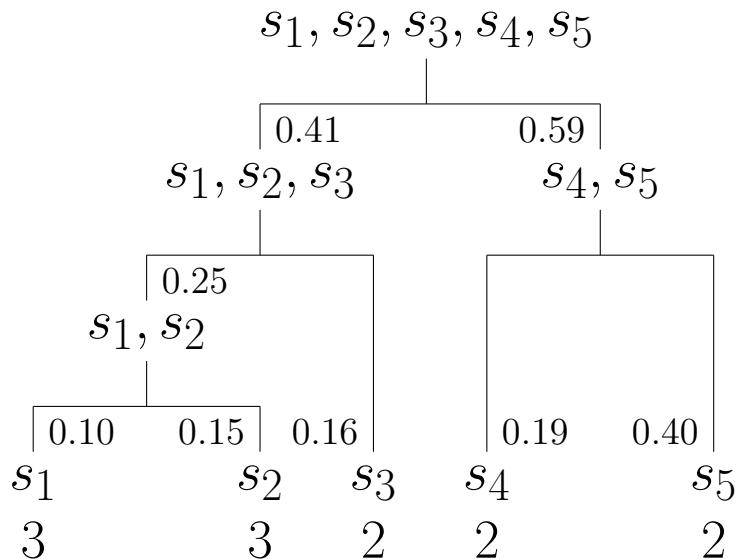
- Splitting into subsets of about equal size can lead to a bad arrangement of the alternatives into subsets → high expected number of questions.
- Good question schemes take the probability of the alternatives into account.
- **Shannon-Fano Coding** (1948)
 - Build the question/coding scheme top-down.
 - Sort the alternatives w.r.t. their probabilities.
 - Split the set so that the subsets have about equal *probability* (splits must respect the probability order of the alternatives).
- **Huffman Coding** (1952)
 - Build the question/coding scheme bottom-up.
 - Start with one element sets.
 - Always combine those two sets that have the smallest probabilities.

Question/Coding Schemes

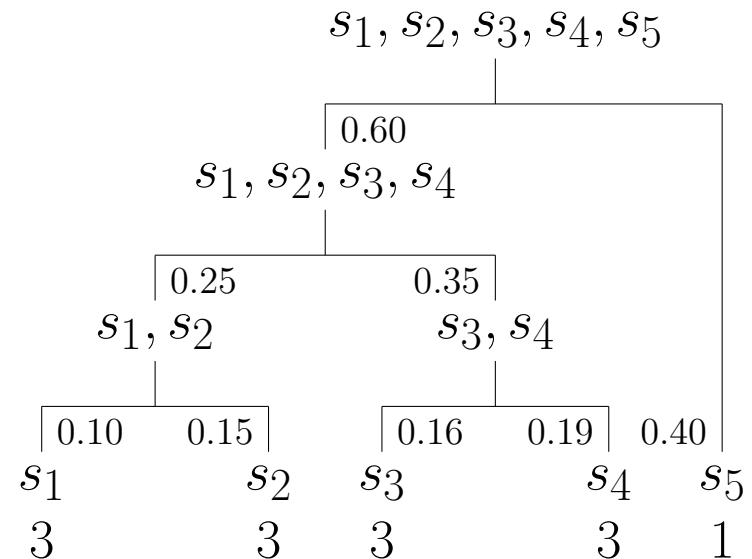
$$P(s_1) = 0.10, \quad P(s_2) = 0.15, \quad P(s_3) = 0.16, \quad P(s_4) = 0.19, \quad P(s_5) = 0.40$$

Shannon entropy: $-\sum_i P(s_i) \log_2 P(s_i) = 2.15$ bit/symbol

Shannon–Fano Coding (1948)



Huffman Coding (1952)



Code length: 2.25 bit/symbol

Code efficiency: 0.955

Code length: 2.20 bit/symbol

Code efficiency: 0.977

Question/Coding Schemes

- It can be shown that Huffman coding is optimal if we have to determine the obtaining alternative in a single instance.
(No question/coding scheme has a smaller expected number of questions.)
- Only if the obtaining alternative has to be determined in a sequence of (independent) situations, this scheme can be improved upon.
- Idea: Process the sequence not instance by instance, but combine two, three or more consecutive instances and ask directly for the obtaining combination of alternatives.
- Although this enlarges the question/coding scheme, the expected number of questions per identification is reduced (because each interrogation identifies the obtaining alternative for several situations).
- However, the expected number of questions per identification cannot be made arbitrarily small. Shannon showed that there is a lower bound, namely the Shannon entropy.

Interpretation of Shannon Entropy

$$P(s_1) = \frac{1}{2}, \quad P(s_2) = \frac{1}{4}, \quad P(s_3) = \frac{1}{8}, \quad P(s_4) = \frac{1}{16}, \quad P(s_5) = \frac{1}{16}$$

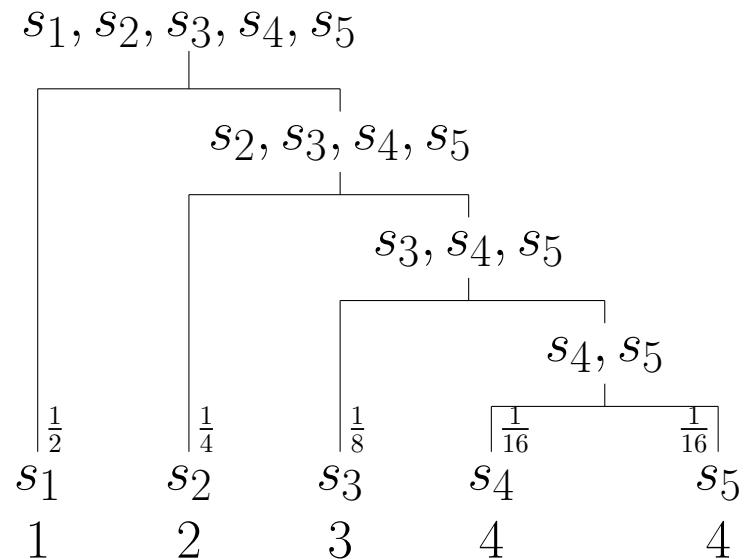
Shannon entropy: $-\sum_i P(s_i) \log_2 P(s_i) = 1.875$ bit/symbol

If the probability distribution allows for a perfect Huffman code (code efficiency 1), the Shannon entropy can easily be interpreted as follows:

$$\begin{aligned} & -\sum_i P(s_i) \log_2 P(s_i) \\ &= \sum_i \underbrace{P(s_i)}_{\substack{\text{occurrence} \\ \text{probability}}} \cdot \underbrace{\log_2 \frac{1}{P(s_i)}}_{\substack{\text{path length} \\ \text{in tree}}}. \end{aligned}$$

In other words, it is the expected number of needed yes/no questions.

Perfect Question Scheme



Code length: 1.875 bit/symbol
Code efficiency: 1

Other Information-theoretic Evaluation Measures

Normalized Information Gain

- Information gain is biased towards many-valued attributes.
- Normalization removes / reduces this bias.

Information Gain Ratio (Quinlan 1986 / 1993)

$$I_{\text{gr}}(C, A) = \frac{I_{\text{gain}}(C, A)}{H_A} = \frac{I_{\text{gain}}(C, A)}{-\sum_{j=1}^{n_A} p_{.j} \log_2 p_{.j}}$$

Symmetric Information Gain Ratio (López de Mántaras 1991)

$$I_{\text{sgr}}^{(1)}(C, A) = \frac{I_{\text{gain}}(C, A)}{H_{AC}} \quad \text{or} \quad I_{\text{sgr}}^{(2)}(C, A) = \frac{I_{\text{gain}}(C, A)}{H_A + H_C}$$

Bias of Information Gain

- **Information gain is biased towards many-valued attributes**, i.e., of two attributes having about the same information content it tends to select the one having more values.
- The reasons are quantization effects caused by the finite number of example cases (due to which only a finite number of different probabilities can result in estimations) in connection with the following theorem:
- **Theorem:** Let A , B , and C be three attributes with finite domains and let their joint probability distribution be strictly positive, i.e., $\forall a \in \text{dom}(A) : \forall b \in \text{dom}(B) : \forall c \in \text{dom}(C) : P(A = a, B = b, C = c) > 0$. Then

$$I_{\text{gain}}(C, AB) \geq I_{\text{gain}}(C, B),$$

with equality obtaining only if the attributes C and A are conditionally independent given B , i.e., if $P(C = c | A = a, B = b) = P(C = c | B = b)$.

(A detailed proof of this theorem can be found, for example, in [Borgelt and Kruse 2002], p. 311ff.)

A Statistical Evaluation Measure

χ^2 Measure

- Compares the actual joint distribution with a **hypothetical independent distribution**.
- Uses absolute comparison.
- Can be interpreted as a difference measure.

$$\chi^2(C, A) = \sum_{i=1}^{n_C} \sum_{j=1}^{n_A} N_{..} \frac{(p_{i..} p_{.j} - p_{ij})^2}{p_{i..} p_{.j}}$$

- Side remark: Information gain can also be interpreted as a difference measure.

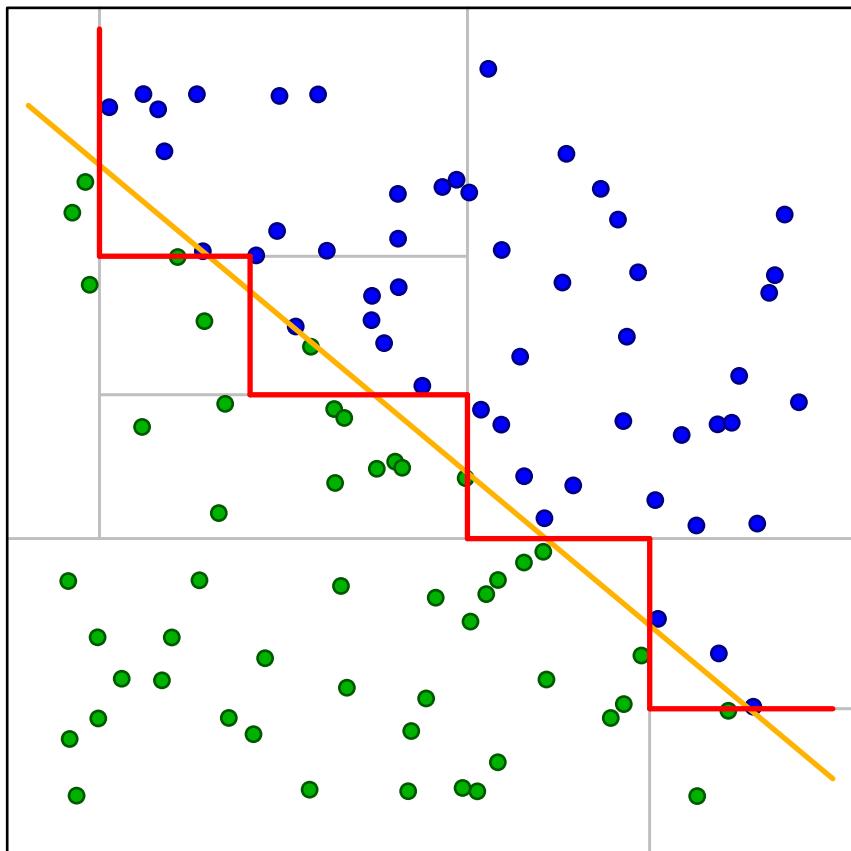
$$I_{\text{gain}}(C, A) = \sum_{i=1}^{n_C} \sum_{j=1}^{n_A} p_{ij} \log_2 \frac{p_{ij}}{p_{i..} p_{.j}}$$

Treatment of Numeric Attributes

General Approach: Discretization

- **Preprocessing I**
 - Form equally sized or equally populated intervals.
- **During the tree construction**
 - Sort the example cases according to the attribute's values.
 - Construct a binary symbolic attribute for every possible split (values: “ \leq threshold” and “ $>$ threshold”).
 - Compute the evaluation measure for these binary attributes.
 - Possible improvements: Add a penalty depending on the number of splits.
- **Preprocessing II / Multisplits during tree construction**
 - Build a decision tree using only the numeric attribute.
 - Flatten the tree to obtain a multi-interval discretization.

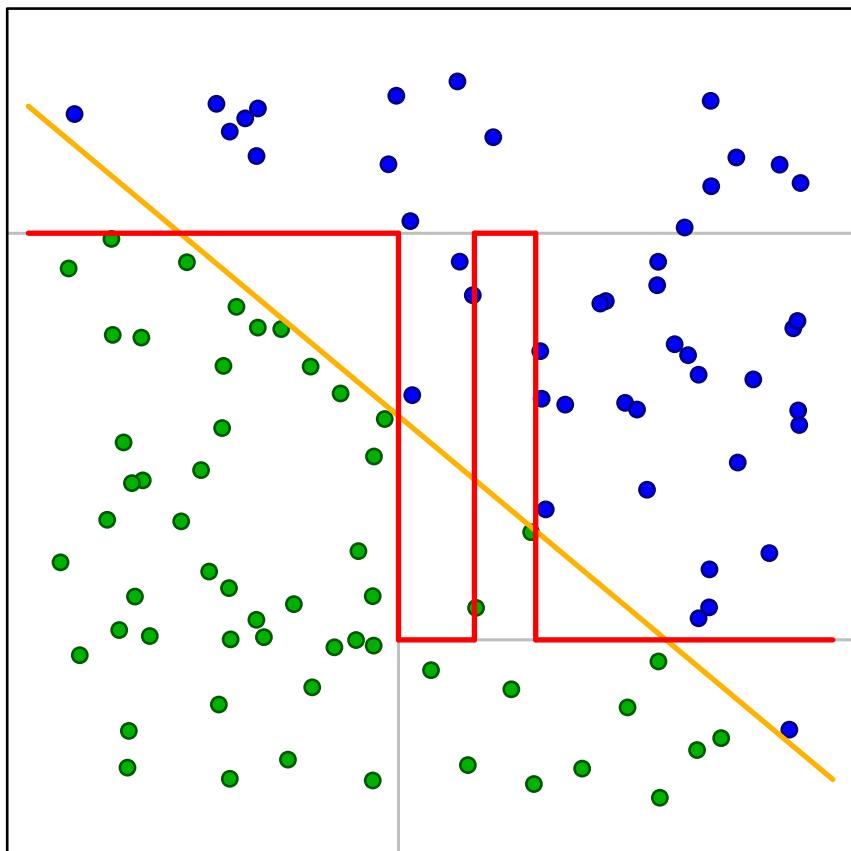
Treatment of Numeric Attributes



So-called “oblique” decision trees are able to find the yellow line.

- Problem: If the class boundary is oblique in the space spanned by two or more numeric attributes, decision trees construct a step function as the decision boundary.
- Green: data points of class A
Blue: data points of class B
Yellow: actual class boundary
Red: decision boundary built by a decision tree
Gray: subdivision of the space used by a decision tree (threshold values)
- Note: the complex decision boundary even produces an error!

Treatment of Numeric Attributes



So-called “oblique” decision trees are able to find the yellow line.

- For the data set on the preceding slide a decision tree builds a proper step function as the decision boundary. Although sub-optimal, this may still be acceptable.
- Unfortunately, other data point configurations can lead to strange anomalies, which do not approximate the actual decision boundary well.
- Green: data points of class A
Blue: data points of class B
Yellow: actual class boundary
Red: decision boundary built by a decision tree
Gray: subdivision of the space used by a decision tree

Induction

- Weight the evaluation measure with the fraction of cases with known values.
 - Idea: The attribute provides information only if it is known.
- Try to find a surrogate test attribute with similar properties (CART, Breiman *et al.* 1984)
- Assign the case to all branches, weighted in each branch with the relative frequency of the corresponding attribute value (C4.5, Quinlan 1993).

Classification

- Use the surrogate test attribute found during induction.
- Follow all branches of the test attribute, weighted with their relative number of cases, aggregate the class distributions of all leaves reached, and assign the majority class of the aggregated class distribution.

Pruning Decision Trees

Pruning serves the purpose

- to simplify the tree (improve interpretability),
- to avoid overfitting (improve generalization).

Basic ideas:

- Replace “bad” branches (subtrees) by leaves.
- Replace a subtree by its largest branch if it is better.

Common approaches:

- Reduced error pruning
- Pessimistic pruning
- Confidence level pruning
- Minimum description length pruning

Reduced Error Pruning

- Classify a set of new example cases with the decision tree.
(These cases must not have been used for the induction!)
- Determine the number of errors for all leaves.
- The number of errors of a subtree is the sum of the errors of all of its leaves.
- Determine the number of errors for leaves that replace subtrees.
- If such a leaf leads to the same or fewer errors than the subtree,
replace the subtree by the leaf.
- If a subtree has been replaced,
recompute the number of errors of the subtrees it is part of.

Advantage: Very good pruning, effective avoidance of overfitting.

Disadvantage: Additional example cases needed.

Pessimistic Pruning

- Classify a set of example cases with the decision tree.
(These cases may or may not have been used for the induction.)
- Determine the number of errors for all leaves and increase this number by a fixed, user-specified amount r .
- The number of errors of a subtree is the sum of the errors of all of its leaves.
- Determine the number of errors for leaves that replace subtrees (also increased by r).
- If such a leaf leads to the same or fewer errors than the subtree, replace the subtree by the leaf and recompute subtree errors.

Advantage: No additional example cases needed.

Disadvantage: Number of cases in a leaf has no influence.

Confidence Level Pruning

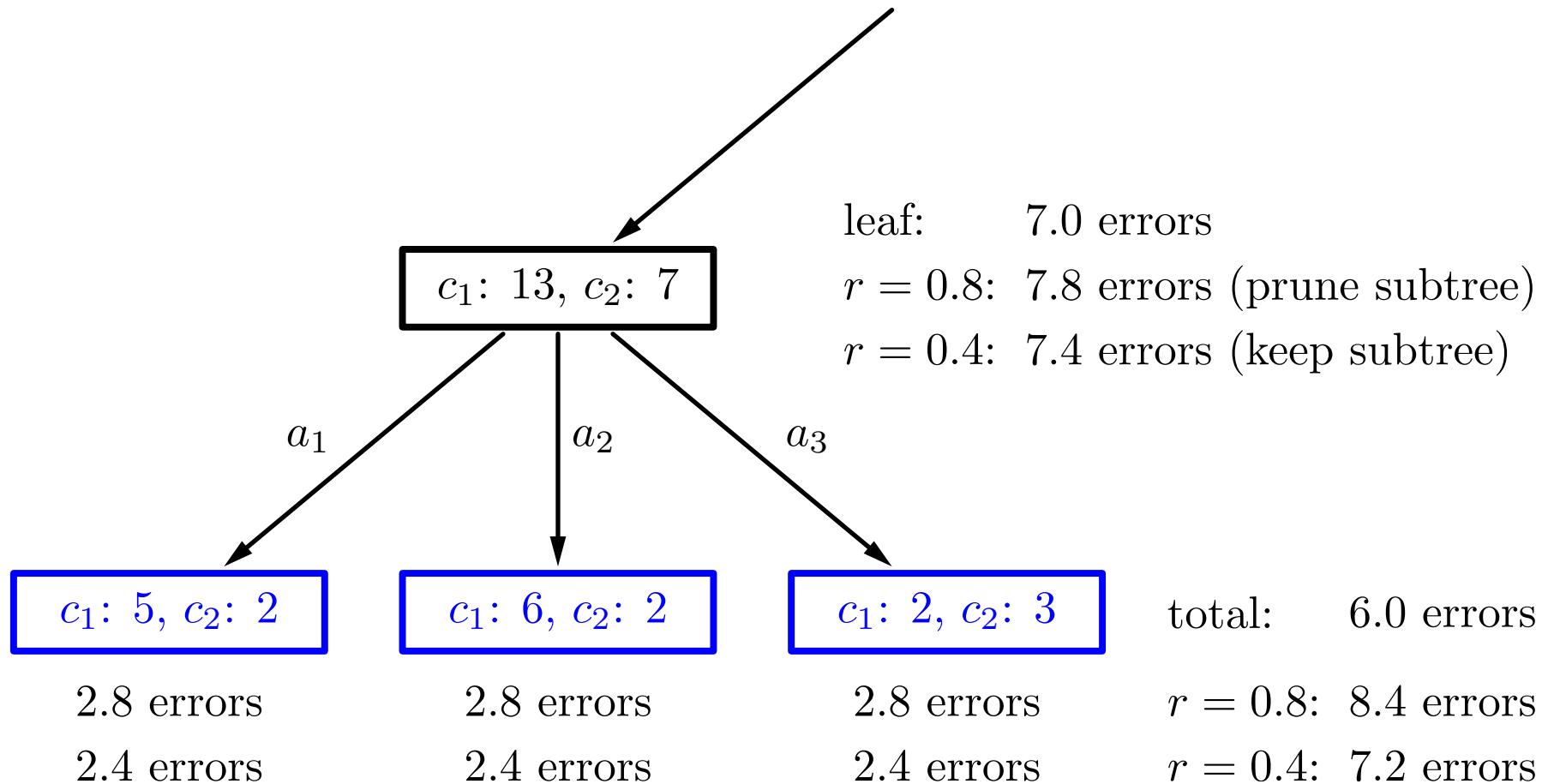
- Like pessimistic pruning, but the number of errors is computed as follows:
 - See classification in a leaf as a Bernoulli experiment (error / no error).
 - Estimate an interval for the error probability based on a user-specified confidence level α .
(use approximation of the binomial distribution by a normal distribution)
 - Increase error number to the upper level of the confidence interval times the number of cases assigned to the leaf.
 - Formal problem: Classification is not a random experiment.

Advantage: No additional example cases needed, good pruning.

Disadvantage: Statistically dubious foundation.

Pruning a Decision Tree: A Simple Example

Pessimistic Pruning with $r = 0.8$ and $r = 0.4$:



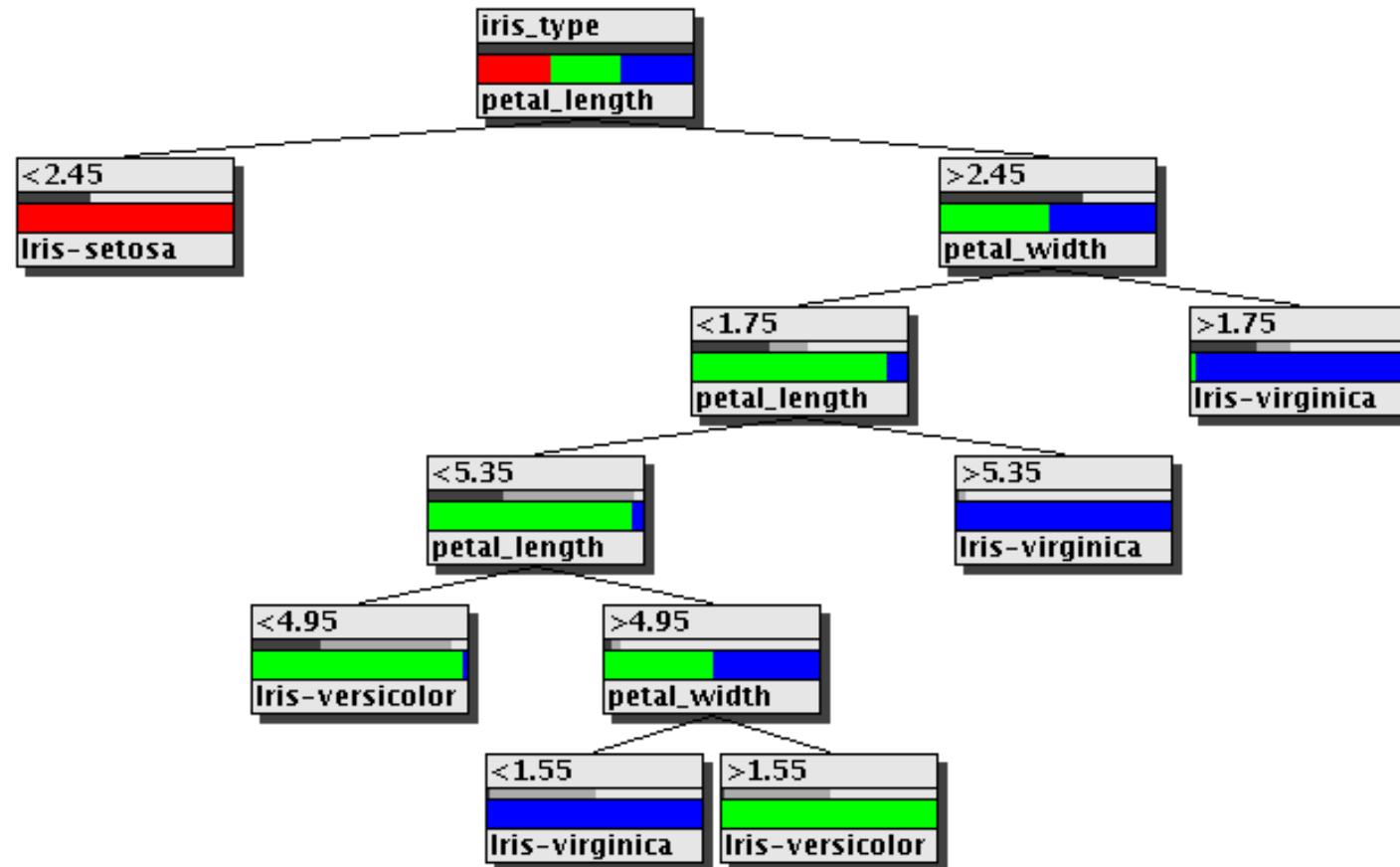
Reminder: The Iris Data

pictures not available in online version

- Collected by Edgar Anderson on the Gaspé Peninsula (Canada).
- First analyzed by Ronald Aylmer Fisher (famous statistician).
- 150 cases in total, 50 cases per Iris flower type.
- Measurements of sepal length and width and petal length and width (in cm).
- Most famous data set in pattern recognition and data analysis.

Decision Trees: An Example

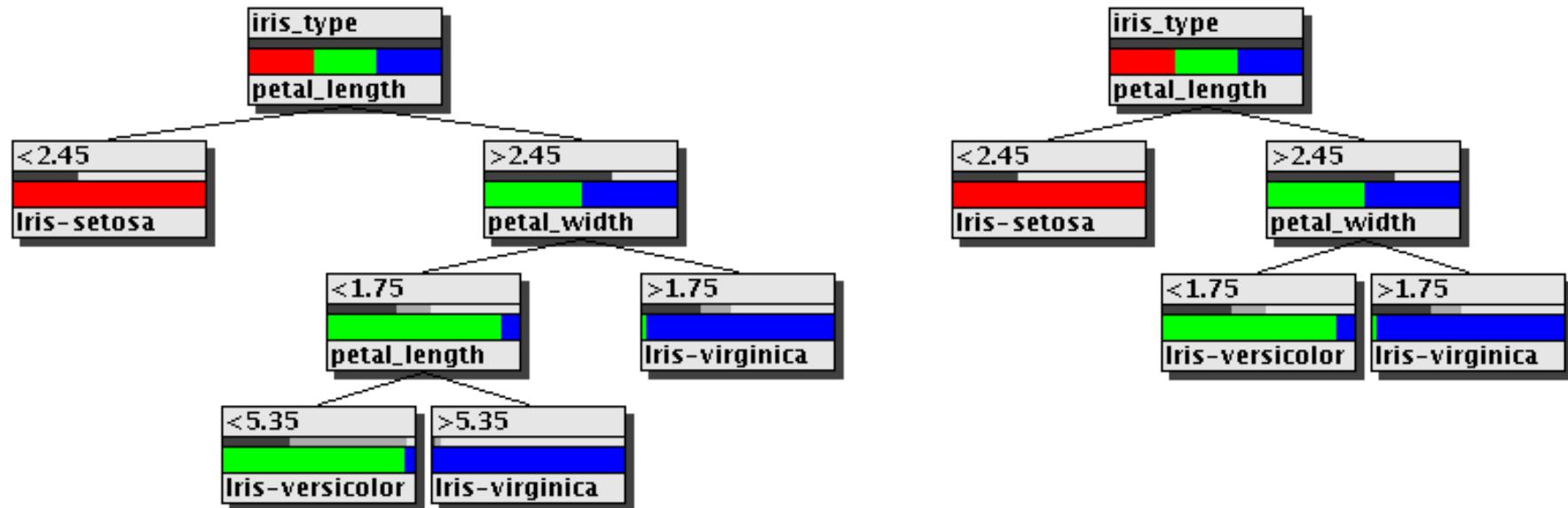
A decision tree for the Iris data
(induced with information gain ratio, unpruned)



Decision Trees: An Example

A decision tree for the Iris data

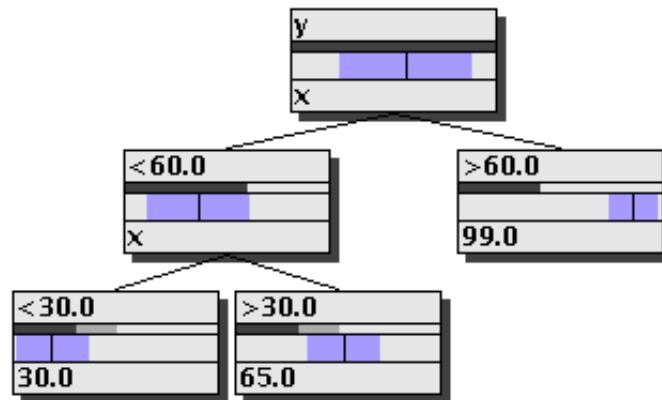
(pruned with confidence level pruning, $\alpha = 0.8$, and pessimistic pruning, $r = 2$)



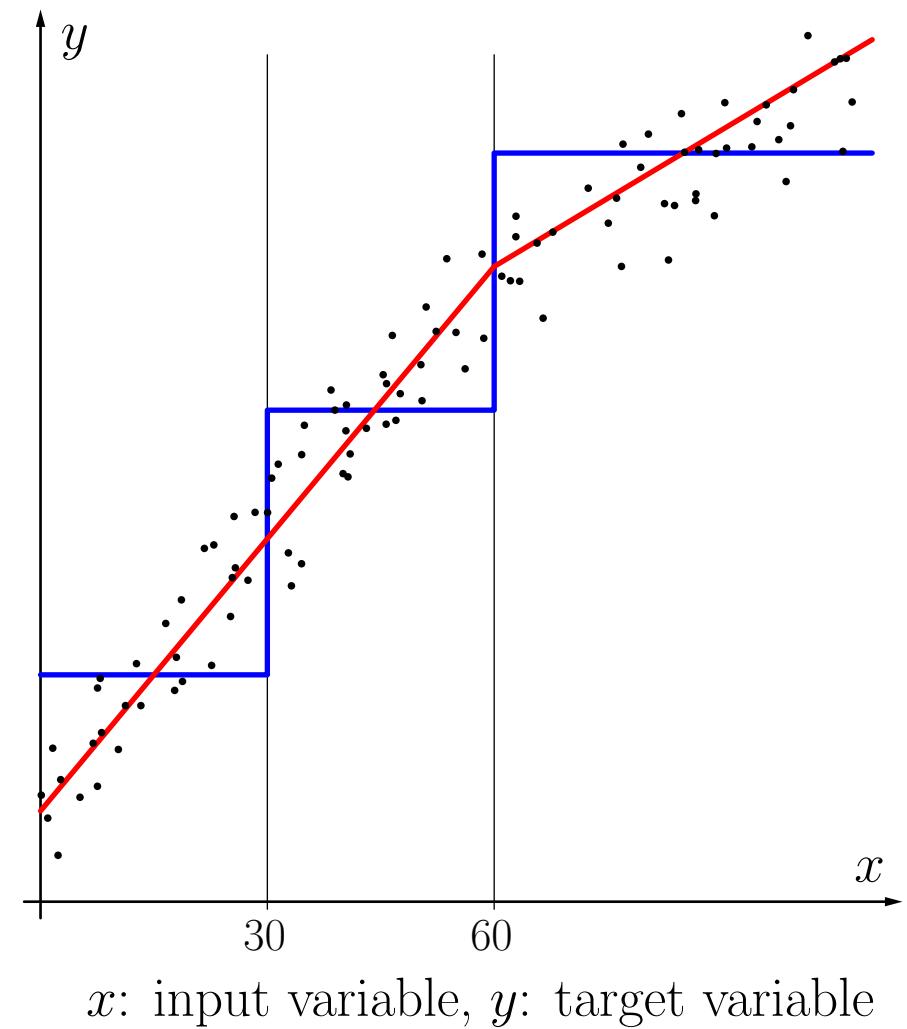
- Left: 7 instead of 11 nodes, 4 instead of 2 misclassifications.
- Right: 5 instead of 11 nodes, 6 instead of 2 misclassifications.
- The right tree is “minimal” for the three classes.

Regression Trees

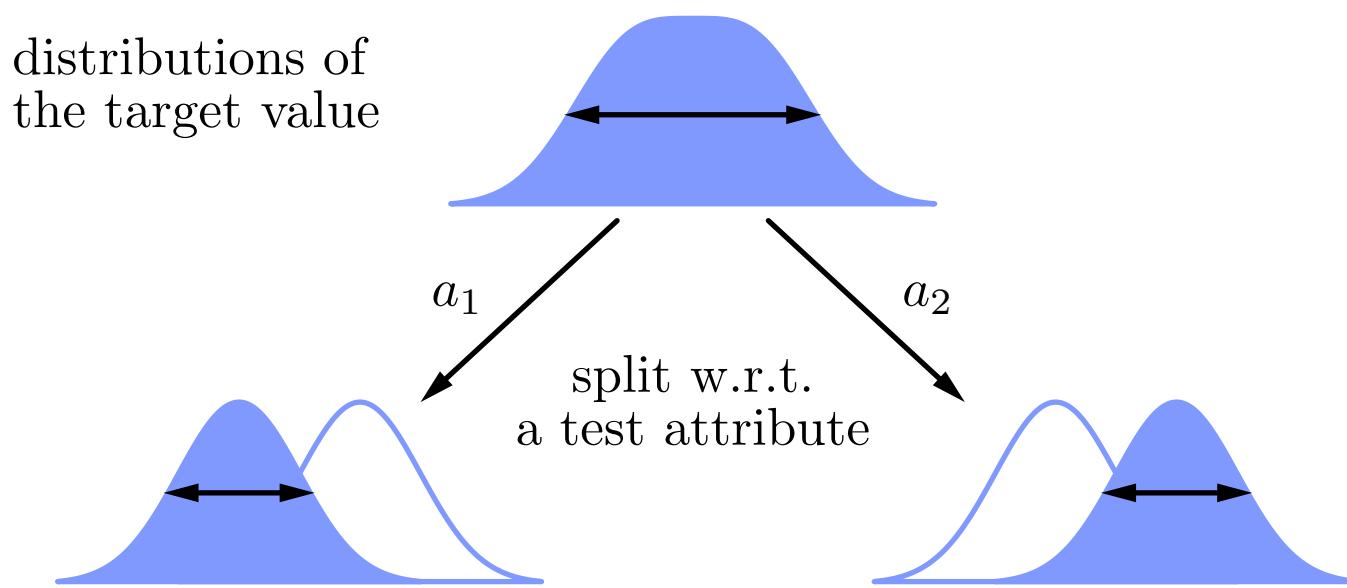
- Target variable is not a class, but a numeric quantity.
- Simple regression trees:
predict constant values in leaves.
(blue lines)



- More complex regression trees:
predict linear functions in leaves.
(red line)



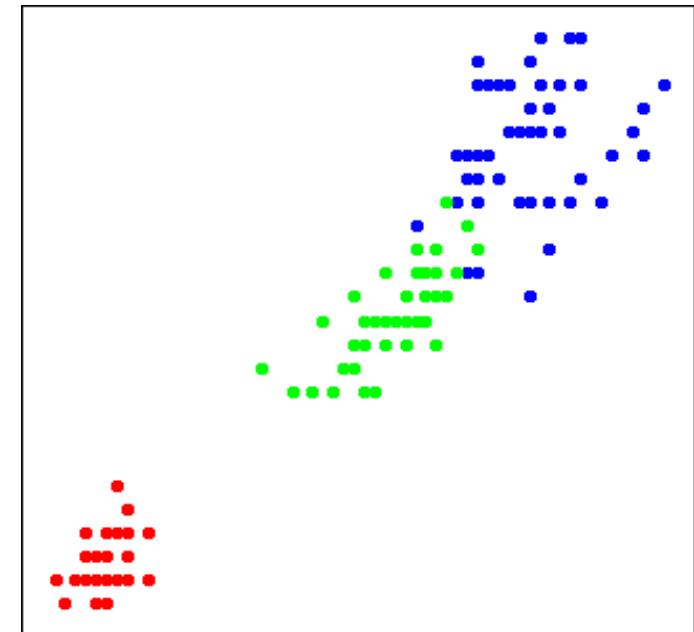
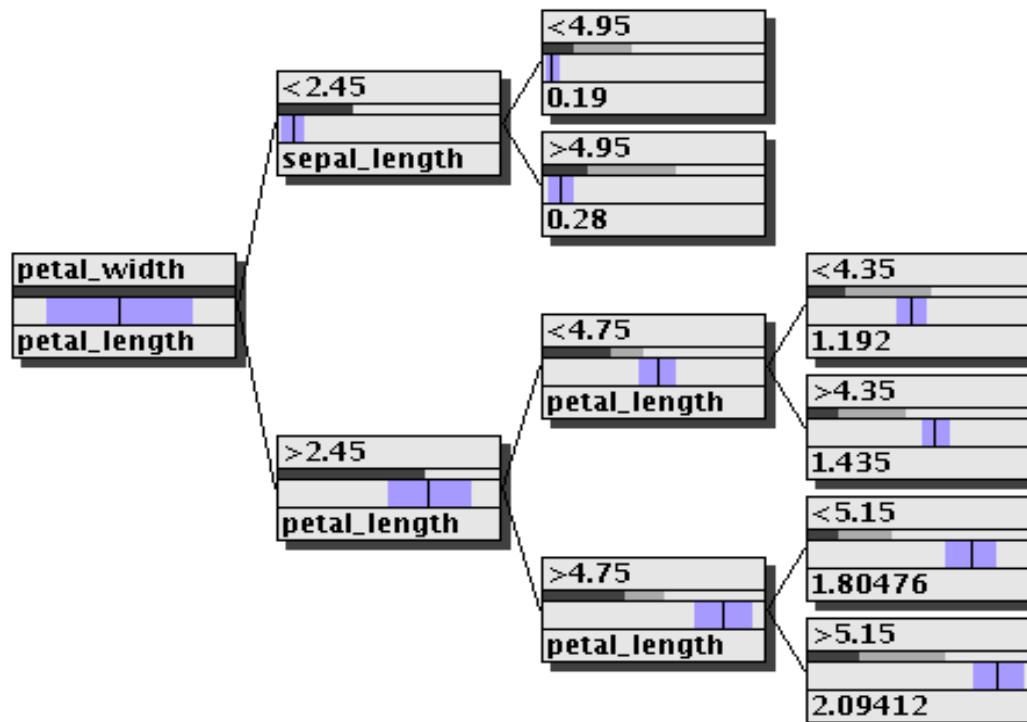
Regression Trees: Attribute Selection



- The variance / standard deviation is compared to the variance / standard deviation in the branches.
- The attribute that yields the highest reduction is selected.

Regression Trees: An Example

A regression tree for the Iris data (petal width)
(induced with reduction of sum of squared errors)



Summary Decision and Regression Trees

- **Decision Trees are Classifiers with Tree Structure**
 - Inner node: Test of a descriptive attribute
 - Leaf node: Assignment of a class
- **Induction of Decision Trees from Data**
(Top-Down Induction of Decision Trees, TDIDT)
 - *Divide and conquer* approach / *recursive descent*
 - *Greedy* selection of the test attributes
 - Attributes are selected based on an *evaluation measure*,
e.g. information gain, χ^2 measure
 - Recommended: *Pruning* of the decision tree
- **Numeric Target: Regression Trees**

Classification Evaluation: Cross Validation

- General method to evaluate / predict the performance of classifiers.
- Serves the purpose to estimate the error rate on new example cases.
- Procedure of cross validation:
 - Split the given data set into n so-called *folds* of equal size (n -fold cross validation).
 - Combine $n - 1$ folds into a training data set, build a classifier, and test it on the n -th fold.
 - Do this for all n possible selections of $n - 1$ folds and average the error rates.
- Special case: Leave-1-out cross validation.
(use as many folds as there are example cases)
- Final classifier is learned from the full data set.

Multi-layer Perceptrons

Multi-layer Perceptrons

- **Biological Background**
- **Threshold Logic Units**
 - Definition, Geometric Interpretation, Linear Separability
 - Training Threshold Logic Units, Limitations
 - Networks of Threshold Logic Units
- **Multilayer Perceptrons**
 - Definition of Multilayer Perceptrons
 - Why Non-linear Activation Functions?
 - Function Approximation
 - Training with Gradient Descent
 - Training Examples and Variants
- **Summary**

Biological Background

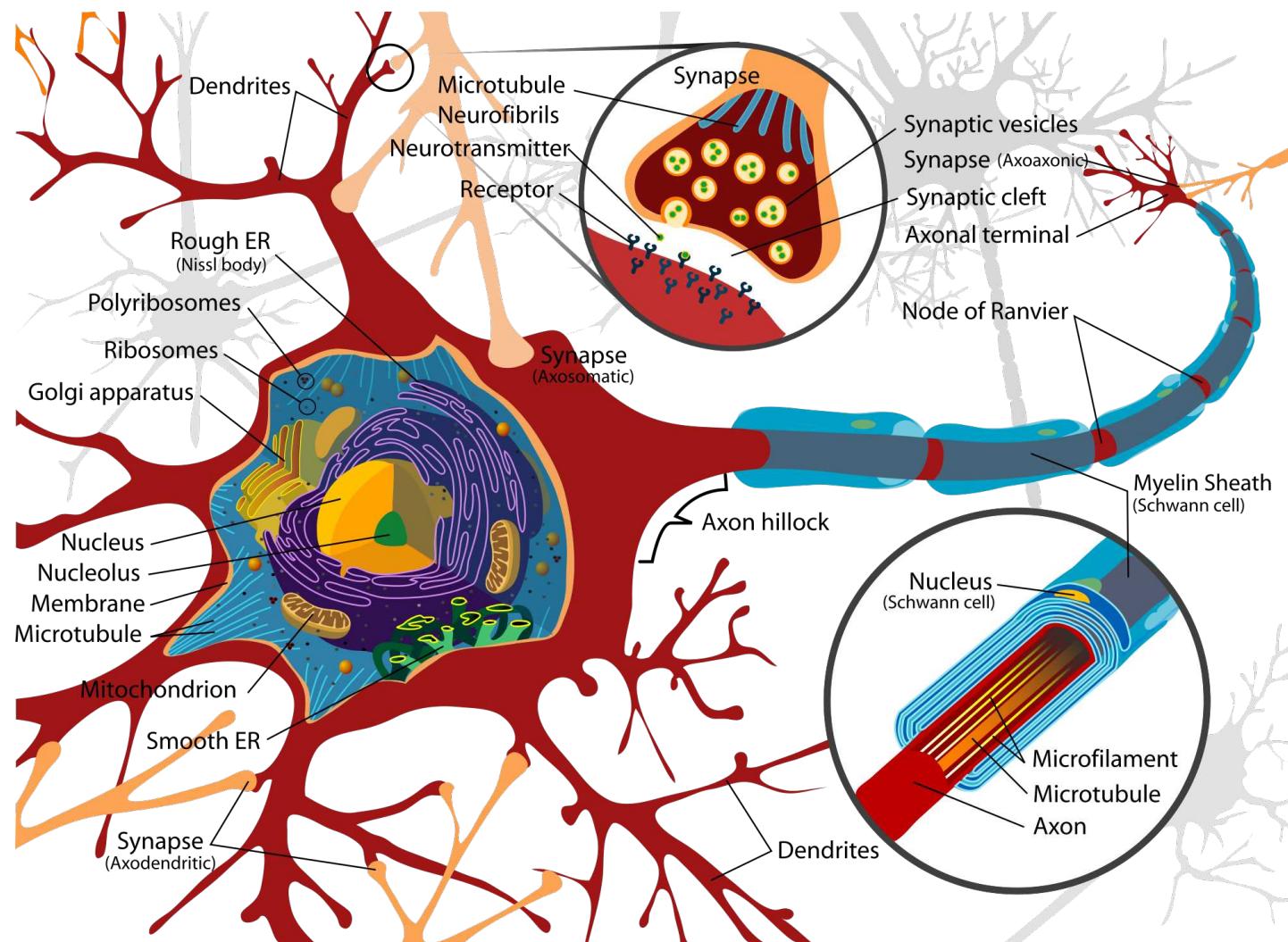
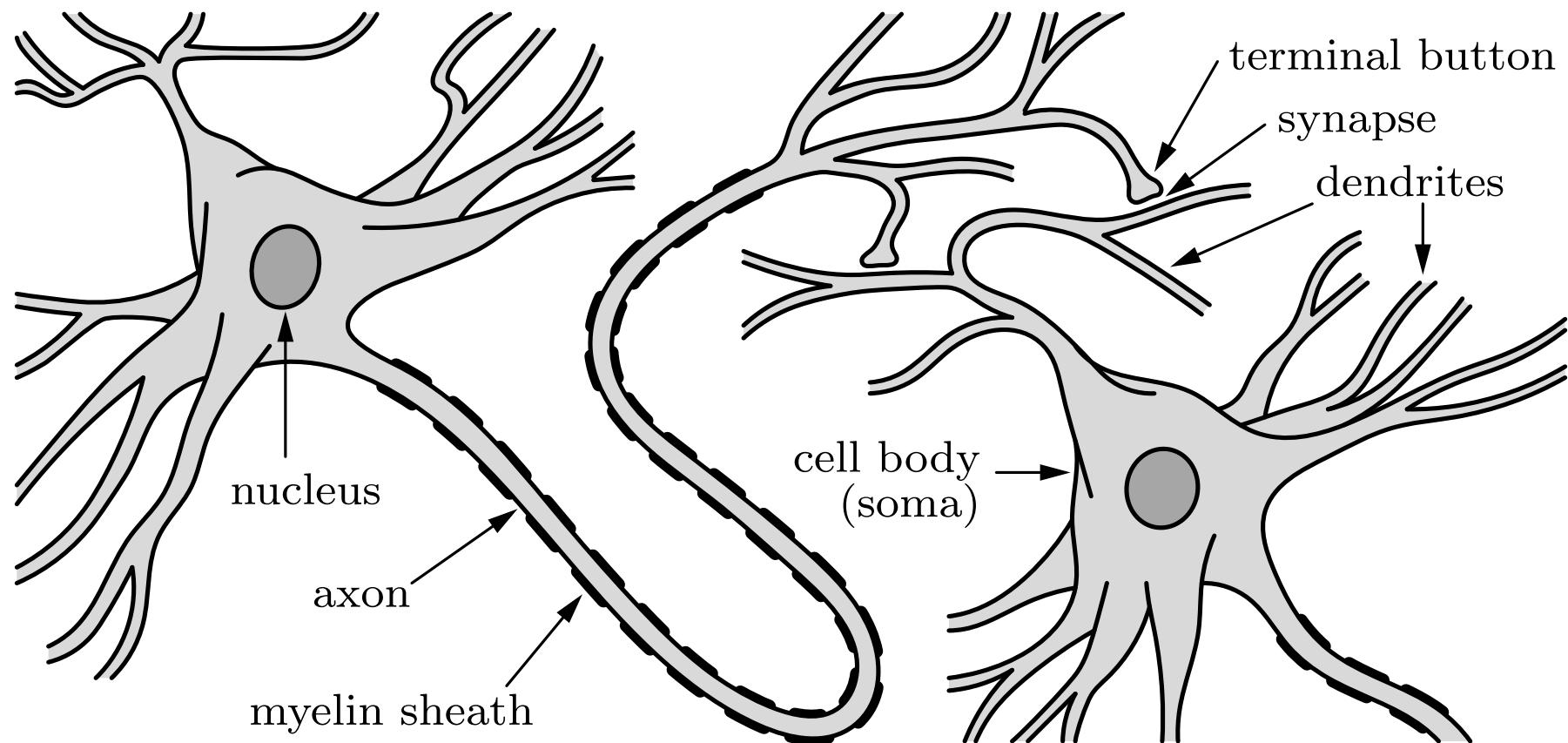


Diagram of a typical myelinated vertebrate motoneuron (source: Wikipedia, Ruiz-Villarreal 2007), showing the main parts involved in its signaling activity like the *dendrites*, the *axon*, and the *synapses*.

Biological Background

Structure of a prototypical biological neuron (simplified)



Biological Background

(Very) simplified description of neural information processing

- Axon terminal releases chemicals, called **neurotransmitters**.
- These act on the membrane of the receptor dendrite to change its polarization.
(The inside is usually 70mV more negative than the outside.)
- Decrease in potential difference: **excitatory** synapse
Increase in potential difference: **inhibitory** synapse
- If there is enough net excitatory input, the axon is depolarized.
- The resulting **action potential** travels along the axon.
(Speed depends on the degree to which the axon is covered with myelin.)
- When the action potential reaches the terminal buttons,
it triggers the release of neurotransmitters.

(Personal) Computers versus the Human Brain

	Personal Computer	Human Brain
processing units	1 CPU, 2–10 cores 10^{10} transistors 1–2 graphics cards/GPUs, 10^3 cores/shaders 10^{10} transistors	10^{11} neurons
storage capacity	10^{10} bytes main memory (RAM) 10^{12} bytes external memory	10^{11} neurons 10^{14} synapses
processing speed	10^{-9} seconds 10^9 operations per second	$> 10^{-3}$ seconds < 1000 per second
bandwidth	10^{12} bits/second	10^{14} bits/second
neural updates	10^6 per second	10^{14} per second

(Personal) Computers versus the Human Brain

- The processing/switching time of a neuron is relatively large ($> 10^{-3}$ seconds), but updates are computed in parallel.
- A serial simulation on a computer takes several hundred clock cycles per update.

Advantages of Neural Networks:

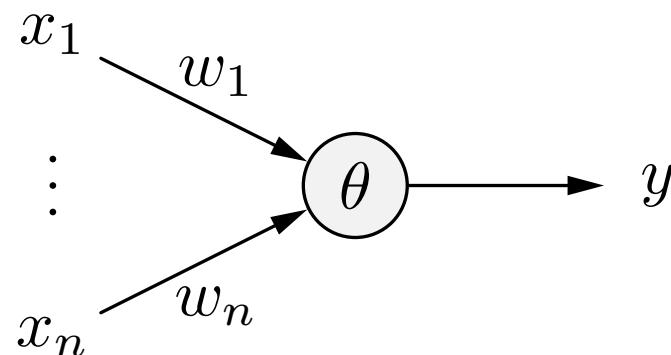
- High processing speed due to massive parallelism.
- Fault Tolerance:
Remain functional even if (larger) parts of a network get damaged.
- “Graceful Degradation”:
gradual degradation of performance if an increasing number of neurons fail.
- Well suited for inductive learning
(learning from examples, generalization from instances).

It appears to be reasonable to try to mimic or to recreate these advantages by constructing **artificial neural networks**.

Threshold Logic Units

A **Threshold Logic Unit (TLU)** is a processing unit for numbers with n inputs x_1, \dots, x_n and one output y . The unit has a **threshold** θ and each input x_i is associated with a **weight** w_i . A threshold logic unit computes the function

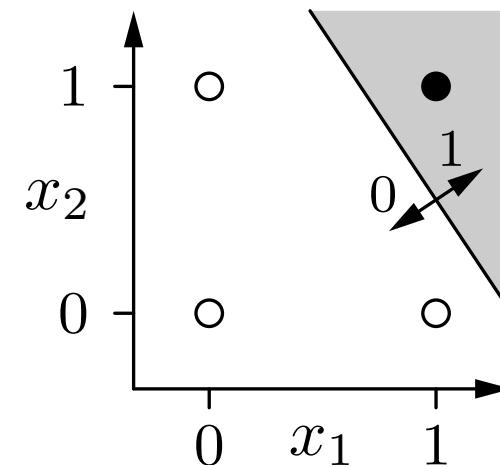
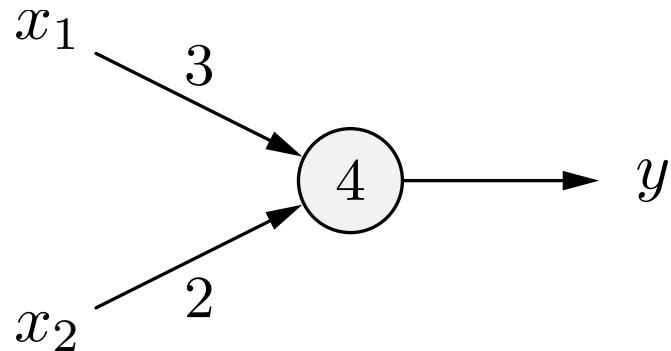
$$y = \begin{cases} 1, & \text{if } \sum_{i=1}^n w_i x_i \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$



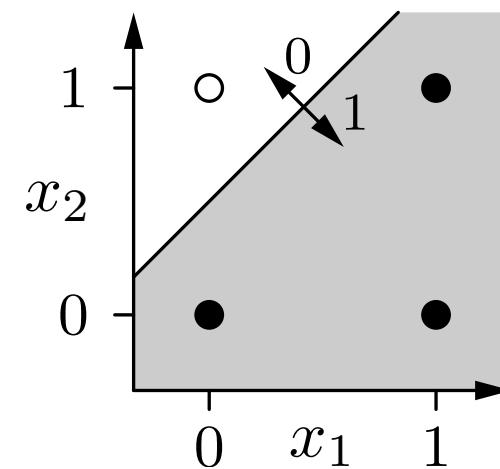
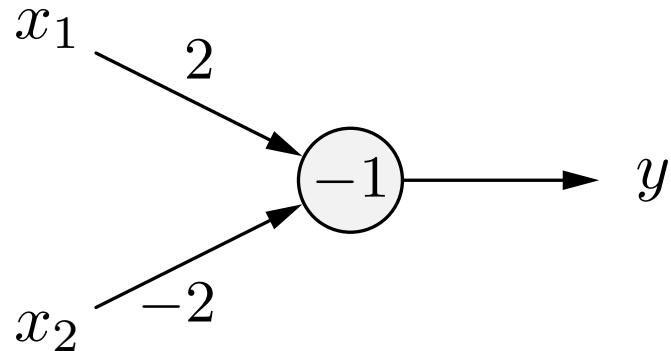
TLUs mimic the thresholding behavior of biological neurons in a (very) simple fashion.

Threshold Logic Units: Geometric Interpretation

Threshold logic unit for $x_1 \wedge x_2$.



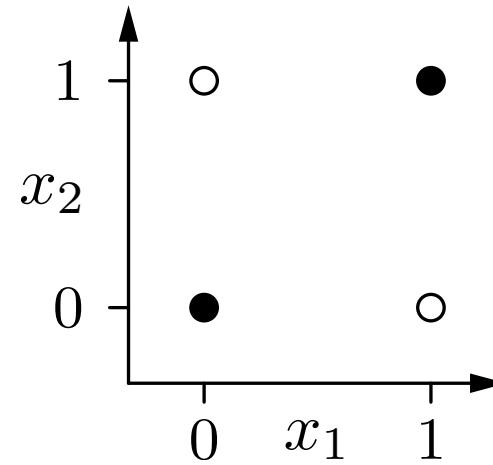
Threshold logic unit for $x_2 \rightarrow x_1$.



Threshold Logic Units: Limitations

The biimplication problem $x_1 \leftrightarrow x_2$: There is no separating line.

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1



Formal proof by *reductio ad absurdum*:

$$\text{since } (0, 0) \mapsto 1: \quad 0 \quad \geq \theta, \quad (1)$$

$$\text{since } (1, 0) \mapsto 0: \quad w_1 \quad < \theta, \quad (2)$$

$$\text{since } (0, 1) \mapsto 0: \quad w_2 \quad < \theta, \quad (3)$$

$$\text{since } (1, 1) \mapsto 1: \quad w_1 + w_2 \geq \theta. \quad (4)$$

(2) and (3): $w_1 + w_2 < 2\theta$. With (4): $2\theta > \theta$, or $\theta > 0$. Contradiction to (1).

Linear Separability

Definition: Two sets of points in a Euclidean space are called **linearly separable**, iff there exists at least one point, line, plane or hyperplane (depending on the dimension of the Euclidean space), such that all points of the one set lie on one side and all points of the other set lie on the other side of this point, line, plane or hyperplane (or on it). That is, the point sets can be separated by a **linear decision function**. Formally: Two sets $X, Y \subset \mathbb{R}^m$ are linearly separable iff $\vec{w} \in \mathbb{R}^m$ and $\theta \in \mathbb{R}$ exist such that

$$\forall \vec{x} \in X : \quad \vec{w}^\top \vec{x} < \theta \quad \text{and} \quad \forall \vec{y} \in Y : \quad \vec{w}^\top \vec{y} \geq \theta.$$

- **Boolean functions** define two points sets, namely the set of points that are mapped to the function value 0 and the set of points that are mapped to 1.
⇒ The term “linearly separable” can be transferred to Boolean functions.
- As we have seen, **conjunction** and **implication** are **linearly separable** (as are **disjunction**, NAND, NOR etc.).
- The **biimplication** is **not linearly separable** (and neither is the **exclusive or** (XOR)).

Linear Separability

Definition: A set of points in a Euclidean space is called **convex** if it is non-empty and connected (that is, if it is a *region*) and for every pair of points in it every point on the straight line segment connecting the points of the pair is also in the set.

Definition: The **convex hull** of a set of points X in a Euclidean space is the smallest convex set of points that contains X . Alternatively, the **convex hull** of a set of points X is the intersection of all convex sets that contain X .

Theorem: Two sets of points in a Euclidean space are **linearly separable** if and only if their convex hulls are disjoint (that is, have no point in common).

- For the biimplication problem, the convex hulls are the diagonal line segments.
- They share their intersection point and are thus not disjoint.
- Therefore the biimplication is not linearly separable.

Threshold Logic Units: Limitations

Total number and number of linearly separable Boolean functions
(On-Line Encyclopedia of Integer Sequences, oeis.org, A001146 and A000609):

inputs	Boolean functions	linearly separable functions
1	4	4
2	16	14
3	256	104
4	65,536	1,882
5	4,294,967,296	94,572
6	18,446,744,073,709,551,616	15,028,134
n	$2^{(2^n)}$	no general formula known

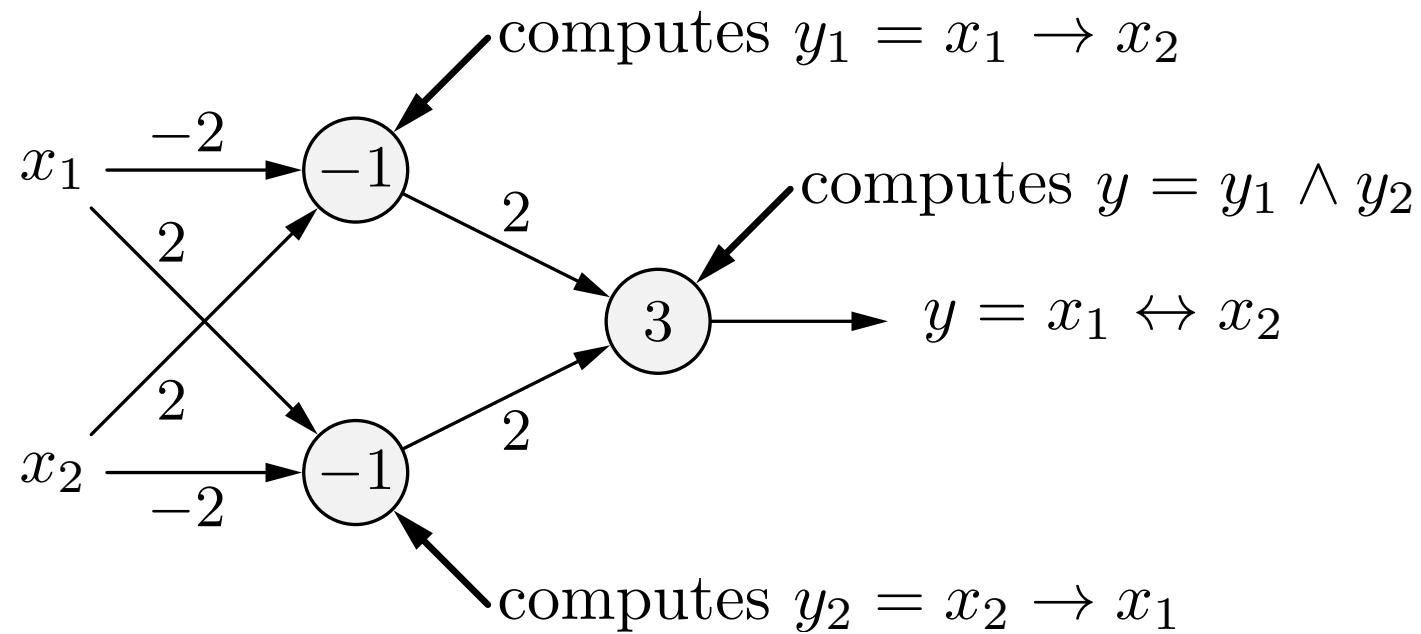
- For many inputs a threshold logic unit can compute almost no functions.
- Networks of threshold logic units are needed to overcome the limitations.

Networks of Threshold Logic Units

Solving the biimplication problem with a network.

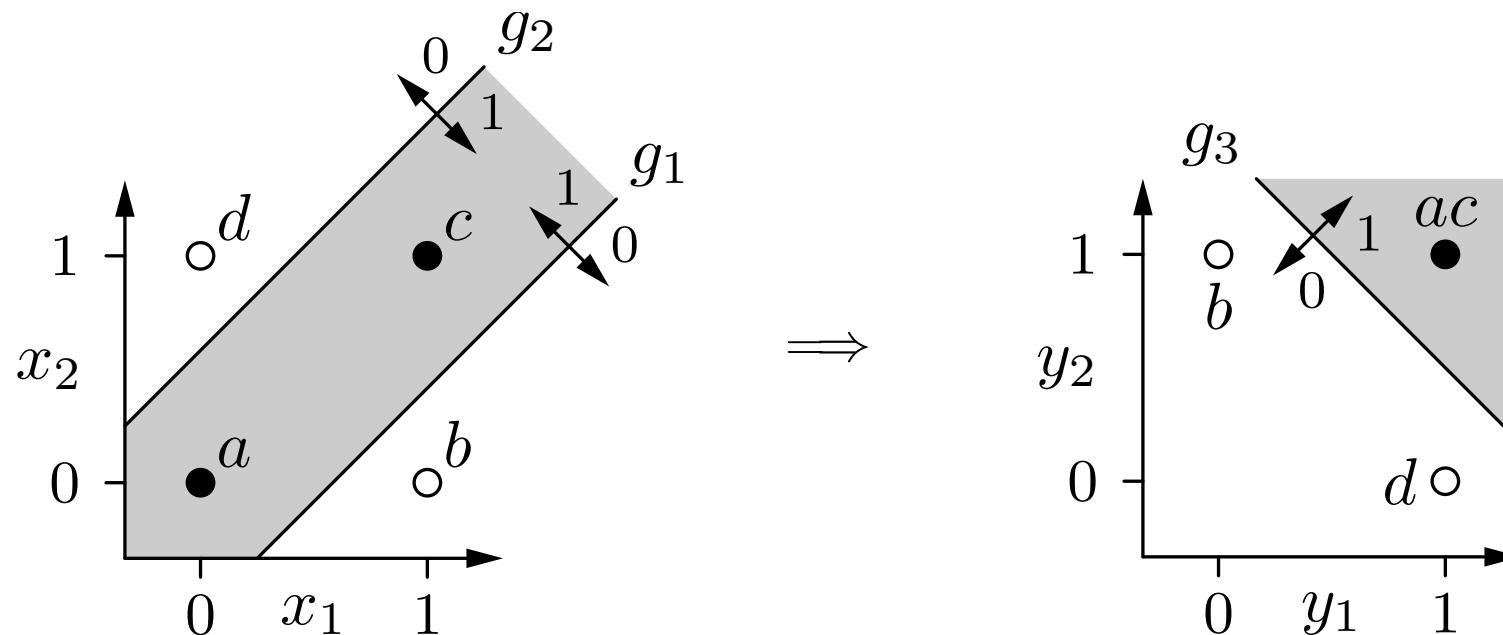
Idea: logical decomposition

$$x_1 \leftrightarrow x_2 \equiv (x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_1)$$



Networks of Threshold Logic Units

Solving the biimplication problem: Geometric interpretation



- The first layer computes new Boolean coordinates for the points.
- After the coordinate transformation the problem is linearly separable.

Representing Arbitrary Boolean Functions

Algorithm: Let $y = f(x_1, \dots, x_n)$ be a Boolean function of n variables.

- (i) Represent the given function $f(x_1, \dots, x_n)$ in disjunctive normal form. That is, determine $D_f = C_1 \vee \dots \vee C_m$, where all C_j are conjunctions of n literals, that is, $C_j = l_{j1} \wedge \dots \wedge l_{jn}$ with $l_{ji} = x_i$ (positive literal) or $l_{ji} = \neg x_i$ (negative literal).
- (ii) Create a neuron for each conjunction C_j of the disjunctive normal form (having n inputs — one input for each variable), where

$$w_{ji} = \begin{cases} +2, & \text{if } l_{ji} = x_i, \\ -2, & \text{if } l_{ji} = \neg x_i, \end{cases} \quad \text{and} \quad \theta_j = n - 1 + \frac{1}{2} \sum_{i=1}^n w_{ji}.$$

- (iii) Create an output neuron (having m inputs — one input for each neuron that was created in step (ii)), where

$$w_{(n+1)k} = 2, \quad k = 1, \dots, m, \quad \text{and} \quad \theta_{n+1} = 1.$$

Remark: weights are set to ± 2 instead of ± 1 in order to ensure integer thresholds.

Representing Arbitrary Boolean Functions

Example:

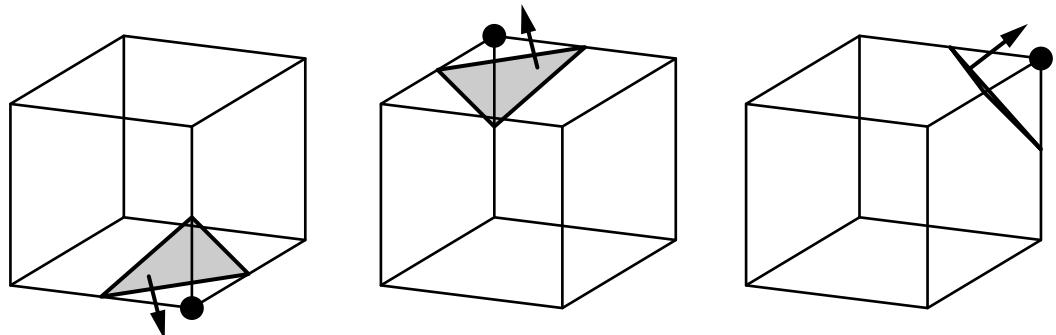
ternary Boolean function:

x_1	x_2	x_3	y	C_j
0	0	0	0	
1	0	0	1	$x_1 \wedge \overline{x_2} \wedge \overline{x_3}$
0	1	0	0	
1	1	0	0	
0	0	1	0	
1	0	1	0	
0	1	1	1	$\overline{x_1} \wedge x_2 \wedge x_3$
1	1	1	1	$x_1 \wedge x_2 \wedge x_3$

$$D_f = C_1 \vee C_2 \vee C_3$$

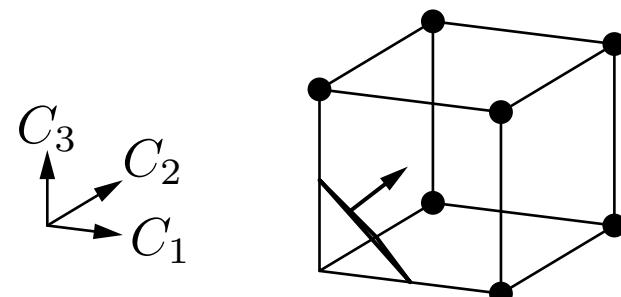
One conjunction for each row
where the output y is 1 with
 literals according to input values.

First layer (conjunctions):



$$\begin{aligned} C_1 &= & C_2 &= & C_3 &= \\ x_1 \wedge \overline{x_2} \wedge \overline{x_3} & & \overline{x_1} \wedge x_2 \wedge x_3 & & x_1 \wedge x_2 \wedge x_3 \end{aligned}$$

Second layer (disjunction):



$$D_f = C_1 \vee C_2 \vee C_3$$

Representing Arbitrary Boolean Functions

Example:

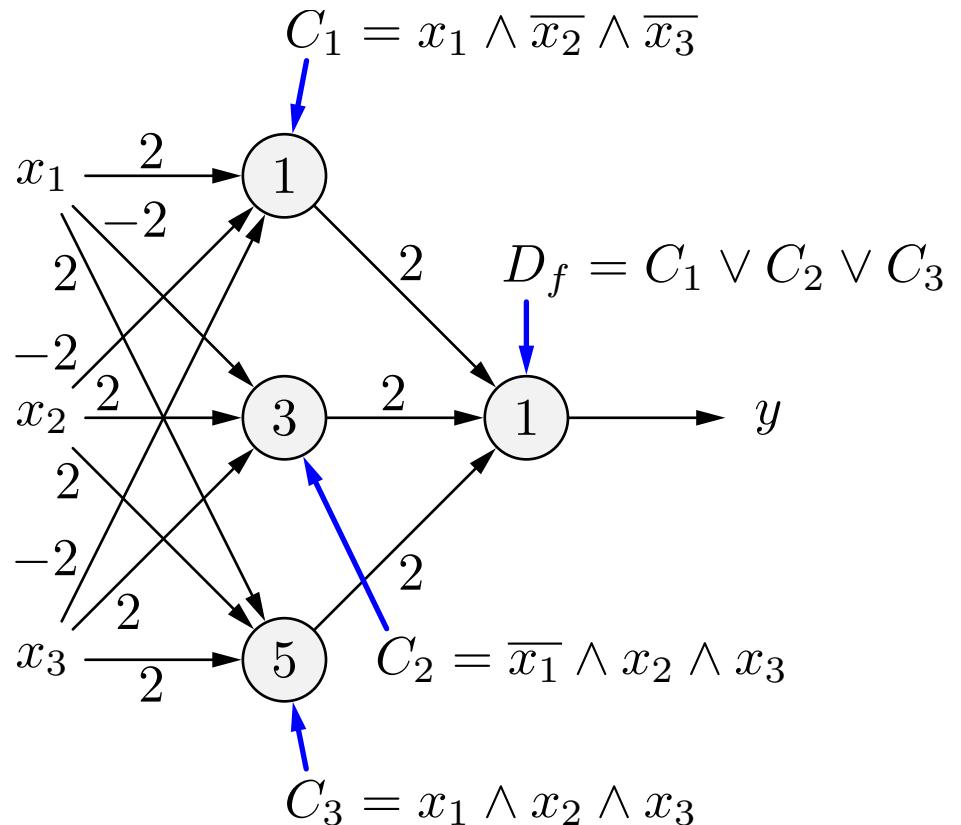
ternary Boolean function:

x_1	x_2	x_3	y	C_j
0	0	0	0	
1	0	0	1	$x_1 \wedge \overline{x_2} \wedge \overline{x_3}$
0	1	0	0	
1	1	0	0	
0	0	1	0	
1	0	1	0	
0	1	1	1	$\overline{x_1} \wedge x_2 \wedge x_3$
1	1	1	1	$x_1 \wedge x_2 \wedge x_3$

$$D_f = C_1 \vee C_2 \vee C_3$$

One conjunction for each row
where the output y is 1 with
 literals according to input value.

Resulting network of threshold logic units:



Training Threshold Logic Units

- Geometric interpretation provides a way to construct threshold logic units with 2 and 3 inputs, but:
 - Not an automatic method (human visualization needed).
 - Not feasible for more than 3 inputs.
- **General idea of automatic training:**
 - Start with random values for weights and threshold.
 - Determine the error of the output for a set of training patterns.
 - Error is a function of the weights and the threshold: $e = e(w_1, \dots, w_n, \theta)$.
 - Adapt weights and threshold so that the error becomes smaller.
 - Iterate adaptation until the error vanishes.

Training Threshold Logic Units: Delta Rule

Formal Training Rule: Let $\vec{x} = (x_1, \dots, x_n)^\top$ be an input vector of a threshold logic unit, o the desired output for this input vector and y the actual output of the threshold logic unit. If $y \neq o$, then the threshold θ and the weight vector $\vec{w} = (w_1, \dots, w_n)^\top$ are adapted as follows in order to reduce the error:

$$\begin{aligned}\theta^{(\text{new})} &= \theta^{(\text{old})} + \Delta\theta \quad \text{with } \Delta\theta = -\eta(o - y), \\ \forall i \in \{1, \dots, n\} : w_i^{(\text{new})} &= w_i^{(\text{old})} + \Delta w_i \quad \text{with } \Delta w_i = \eta(o - y)x_i,\end{aligned}$$

where η is a parameter that is called **learning rate**. It determines the severity of the weight changes. This procedure is called **Delta Rule** or **Widrow–Hoff Procedure** [Widrow and Hoff 1960].

- **Online Training:** Adapt parameters after each training pattern.
- **Batch Training:** Adapt parameters only at the end of each **epoch**, that is, after a traversal of all training patterns.

Training Threshold Logic Units: Convergence

Convergence Theorem: Let $L = \{(\vec{x}_1, o_1), \dots, (\vec{x}_m, o_m)\}$ be a set of training patterns, each consisting of an input vector $\vec{x}_i \in \mathbb{R}^n$ and a desired output $o_i \in \{0, 1\}$. Furthermore, let $L_0 = \{(\vec{x}, o) \in L \mid o = 0\}$ and $L_1 = \{(\vec{x}, o) \in L \mid o = 1\}$. If L_0 and L_1 are linearly separable, that is, if $\vec{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ exist such that

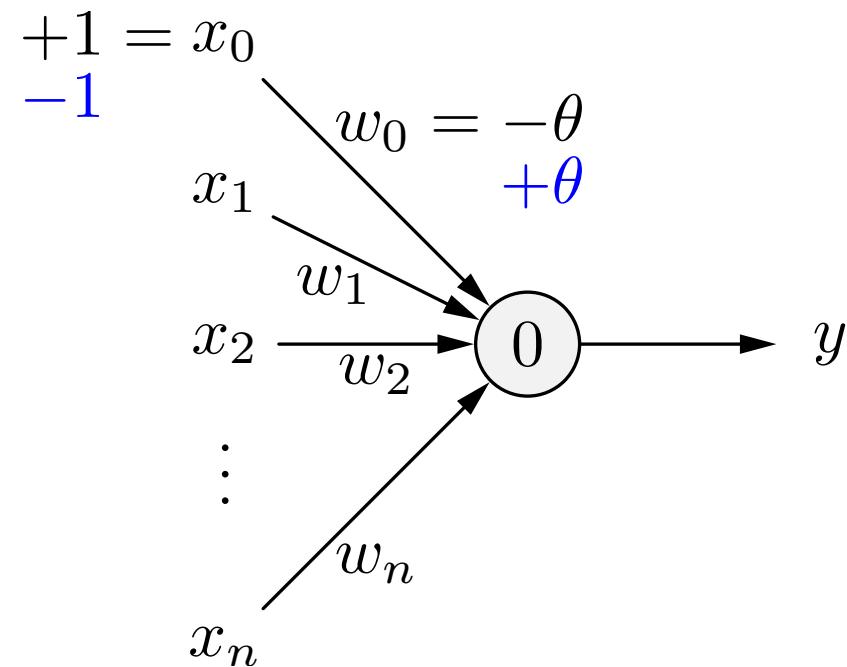
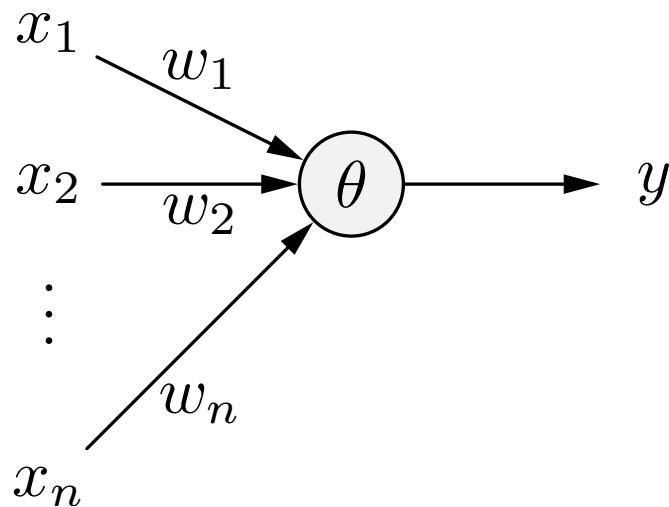
$$\begin{aligned}\forall (\vec{x}, 0) \in L_0 : \quad & \vec{w}^\top \vec{x} < \theta \quad \text{and} \\ \forall (\vec{x}, 1) \in L_1 : \quad & \vec{w}^\top \vec{x} \geq \theta,\end{aligned}$$

then online as well as batch training terminate.

- The algorithms terminate only when the error vanishes.
- Therefore the resulting threshold and weights must solve the problem.
- For not linearly separable problems the algorithms do not terminate (oscillation, repeated computation of same non-solving \vec{w} and θ).

Training Threshold Logic Units: Delta Rule

Turning the threshold value into a weight:



$$\sum_{i=1}^n w_i x_i \geq \theta$$

$$\sum_{i=1}^n w_i x_i - \theta \geq 0$$

Training Threshold Logic Units: Delta Rule

Formal Training Rule (with threshold turned into a weight):

Let $\vec{x} = (\textcolor{blue}{x_0 = 1}, x_1, \dots, x_n)^\top$ be an (extended) input vector of a threshold logic unit, o the desired output for this input vector and y the actual output of the threshold logic unit. If $y \neq o$, then the (extended) weight vector $\vec{w} = (\textcolor{blue}{w_0 = -\theta}, w_1, \dots, w_n)^\top$ is adapted as follows in order to reduce the error:

$$\forall i \in \{0, \dots, n\} : \quad w_i^{(\text{new})} = w_i^{(\text{old})} + \Delta w_i \quad \text{with} \quad \Delta w_i = \eta(o - y)x_i,$$

where η is a parameter that is called **learning rate**. It determines the severity of the weight changes. This procedure is called **Delta Rule** or **Widrow–Hoff Procedure** [Widrow and Hoff 1960].

- Note that with extended input and weight vectors, there is only one update rule (no distinction of threshold and weights).
- Note also that the (extended) input vector may be $\vec{x} = (\textcolor{blue}{x_0 = -1}, x_1, \dots, x_n)^\top$ and the corresponding (extended) weight vector $\vec{w} = (\textcolor{blue}{w_0 = +\theta}, w_1, \dots, w_n)^\top$.

Training Networks of Threshold Logic Units

- **Single threshold logic units** have strong limitations:
They **can only compute linearly separable functions**.
- **Networks of threshold logic units**
can compute arbitrary Boolean functions.
- **Training single threshold logic units** with the delta rule **is easy and fast** and guaranteed to find a solution if one exists.
- **Networks of threshold logic units cannot be trained**, because
 - there are no desired values for the neurons of the first layer(s),
 - the problem can usually be solved with several different functions computed by the neurons of the first layer(s) (non-unique solution).
- When this situation became clear,
neural networks were first seen as a “research dead end”.

General Neural Networks

Basic graph theoretic notions

A (directed) **graph** is a pair $G = (V, E)$ consisting of a (finite) set V of **vertices** or **nodes** and a (finite) set $E \subseteq V \times V$ of **edges**.

We call an edge $e = (u, v) \in E$ **directed** from vertex u to vertex v .

Let $G = (V, E)$ be a (directed) graph and $u \in V$ a vertex.

Then the vertices of the set

$$\text{pred}(u) = \{v \in V \mid (v, u) \in E\}$$

are called the **predecessors** of the vertex u

and the vertices of the set

$$\text{succ}(u) = \{v \in V \mid (u, v) \in E\}$$

are called the **successors** of the vertex u .

General Neural Networks

General definition of a neural network

An (artificial) **neural network** is a (directed) graph $G = (U, C)$, whose vertices $u \in U$ are called **neurons** or **units** and whose edges $c \in C$ are called **connections**.

The set U of vertices is partitioned into

- the set U_{in} of **input neurons**,
- the set U_{out} of **output neurons**, and
- the set U_{hidden} of **hidden neurons**.

It is

$$U = U_{\text{in}} \cup U_{\text{out}} \cup U_{\text{hidden}},$$

$$U_{\text{in}} \neq \emptyset, \quad U_{\text{out}} \neq \emptyset, \quad U_{\text{hidden}} \cap (U_{\text{in}} \cup U_{\text{out}}) = \emptyset.$$

General Neural Networks

Each connection $(v, u) \in C$ possesses a **weight** w_{uv} and each neuron $u \in U$ possesses three (real-valued) state variables:

- the **network input** net_u ,
- the **activation** act_u , and
- the **output** out_u .

Each input neuron $u \in U_{\text{in}}$ also possesses a fourth (real-valued) state variable,

- the **external input** ext_u .

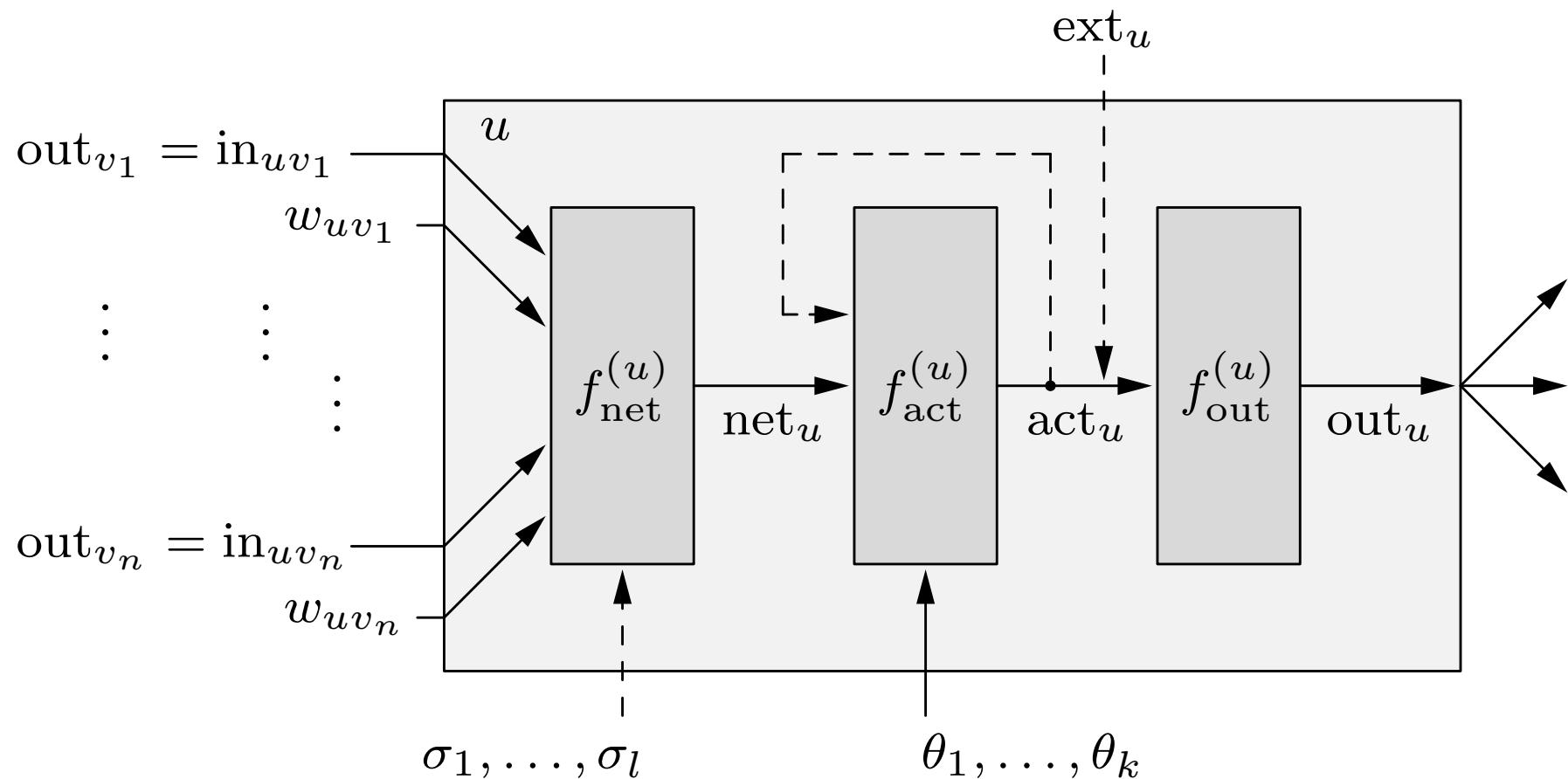
Furthermore, each neuron $u \in U$ possesses three functions:

- the **network input function** $f_{\text{net}}^{(u)} : \mathbb{R}^{2|\text{pred}(u)| + \kappa_1(u)} \rightarrow \mathbb{R}$,
- the **activation function** $f_{\text{act}}^{(u)} : \mathbb{R}^{\kappa_2(u)} \rightarrow \mathbb{R}$, and
- the **output function** $f_{\text{out}}^{(u)} : \mathbb{R} \rightarrow \mathbb{R}$,

which are used to compute the values of the state variables.

Structure of a Generalized Neuron

A generalized neuron is a simple numeric processor



General Neural Networks

Types of (artificial) neural networks:

- If the graph of a neural network is **acyclic**, it is called a **feed-forward network**.
- If the graph of a neural network contains **cycles** (backward connections), it is called a **recurrent network**.

Representation of the connection weights as a matrix:

$$\begin{array}{cccc} u_1 & u_2 & \dots & u_r \\ \left(\begin{array}{cccc} w_{u_1u_1} & w_{u_1u_2} & \dots & w_{u_1u_r} \\ w_{u_2u_1} & w_{u_2u_2} & & w_{u_2u_r} \\ \vdots & & & \vdots \\ w_{u_ru_1} & w_{u_ru_2} & \dots & w_{u_ru_r} \end{array} \right) & \begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_r \end{array} \end{array}$$

Multi-layer Perceptrons

An **r-layer perceptron** is a neural network with a graph $G = (U, C)$ that satisfies the following conditions:

(i) $U_{\text{in}} \cap U_{\text{out}} = \emptyset$,

(ii) $U_{\text{hidden}} = U_{\text{hidden}}^{(1)} \cup \dots \cup U_{\text{hidden}}^{(r-2)}$,

$$\forall 1 \leq i < j \leq r-2 : \quad U_{\text{hidden}}^{(i)} \cap U_{\text{hidden}}^{(j)} = \emptyset,$$

(iii) $C \subseteq \left(U_{\text{in}} \times U_{\text{hidden}}^{(1)} \right) \cup \left(\bigcup_{i=1}^{r-3} U_{\text{hidden}}^{(i)} \times U_{\text{hidden}}^{(i+1)} \right) \cup \left(U_{\text{hidden}}^{(r-2)} \times U_{\text{out}} \right)$

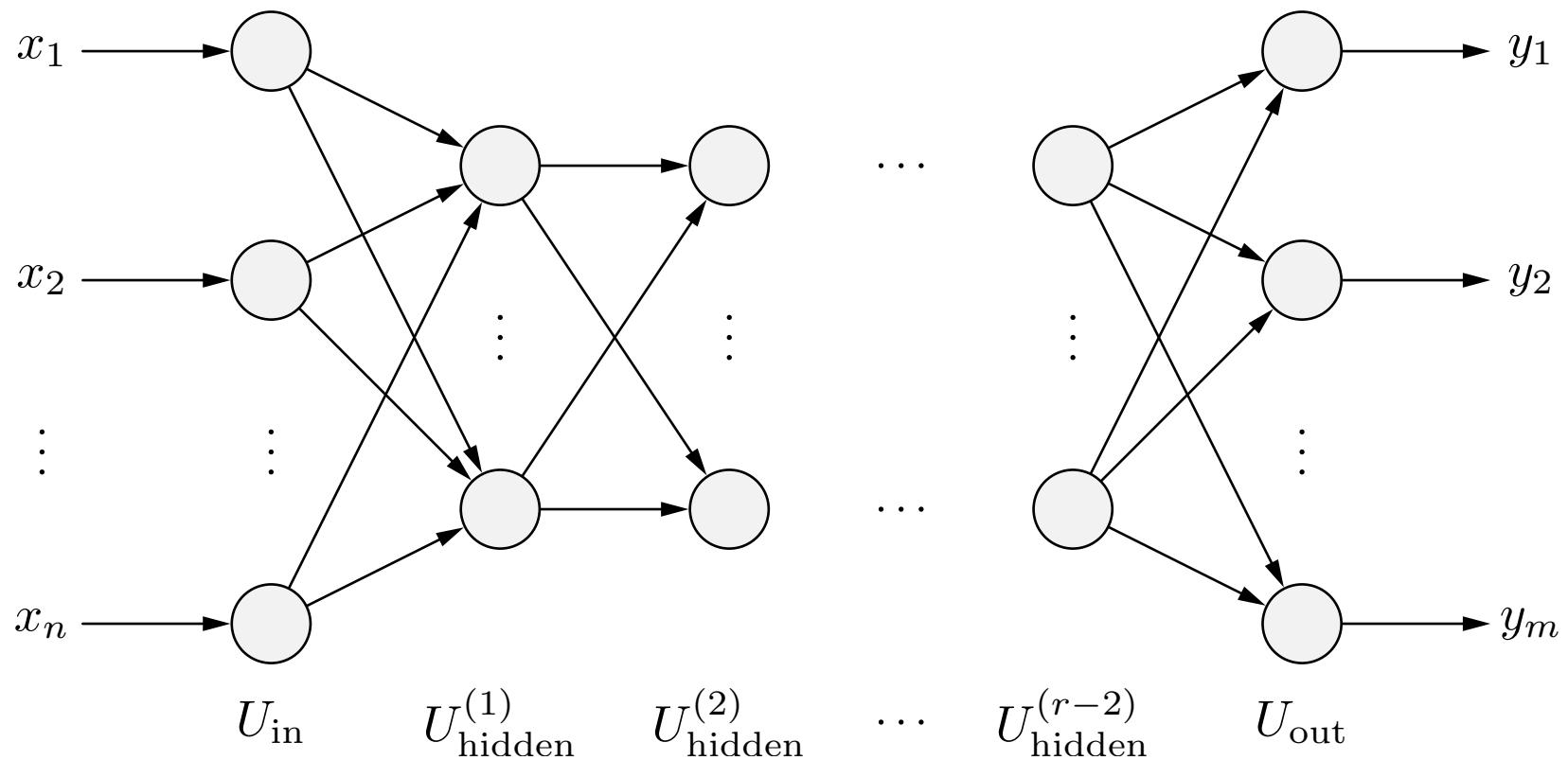
or, if there are no hidden neurons ($r = 2, U_{\text{hidden}} = \emptyset$),

$$C \subseteq U_{\text{in}} \times U_{\text{out}}.$$

- Feed-forward network with strictly layered structure.

Multi-layer Perceptrons

General structure of a multi-layer perceptron



Multi-layer Perceptrons

- The network input function of each hidden neuron and of each output neuron is the **weighted sum** of its inputs, that is,

$$\forall u \in U_{\text{hidden}} \cup U_{\text{out}} : f_{\text{net}}^{(u)}(\vec{w}_u, \vec{\text{in}}_u) = \vec{w}_u^\top \vec{\text{in}}_u = \sum_{v \in \text{pred}(u)} w_{uv} \text{out}_v.$$

- The activation function of each hidden neuron is a so-called **sigmoid function**, that is, a monotonically increasing function

$$f : \mathbb{R} \rightarrow [0, 1] \quad \text{with} \quad \lim_{x \rightarrow -\infty} f(x) = 0 \quad \text{and} \quad \lim_{x \rightarrow \infty} f(x) = 1.$$

- The activation function of each output neuron is either also a sigmoid function or a **linear function**, that is,

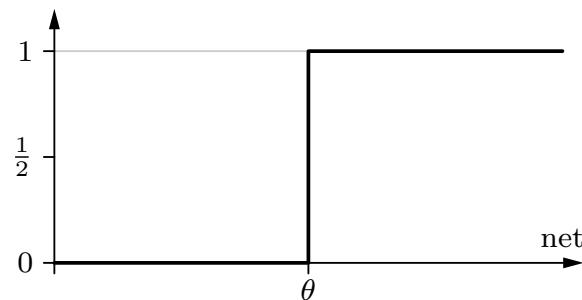
$$f_{\text{act}}(\text{net}, \theta) = \alpha \text{net} - \theta.$$

Only the step function is a neurobiologically plausible activation function.

Sigmoid Activation Functions

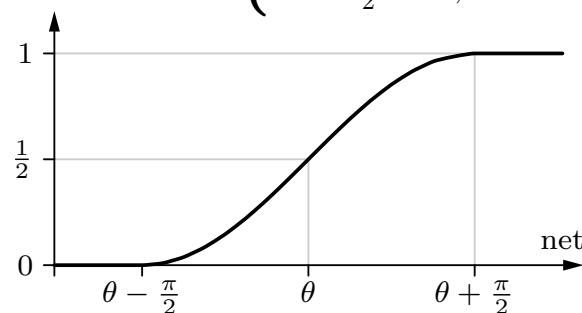
step function:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$



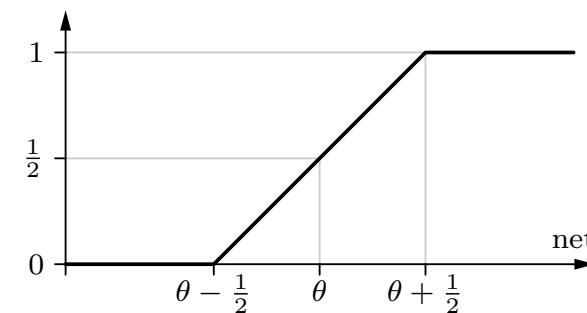
sine until saturation:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} > \theta + \frac{\pi}{2}, \\ 0, & \text{if net} < \theta - \frac{\pi}{2}, \\ \frac{\sin(\text{net} - \theta) + 1}{2}, & \text{otherwise.} \end{cases}$$



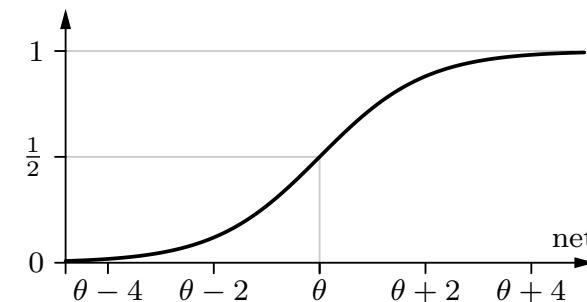
semi-linear function:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} > \theta + \frac{1}{2}, \\ 0, & \text{if net} < \theta - \frac{1}{2}, \\ (\text{net} - \theta) + \frac{1}{2}, & \text{otherwise.} \end{cases}$$



logistic function:

$$f_{\text{act}}(\text{net}, \theta) = \frac{1}{1 + e^{-(\text{net} - \theta)}}$$

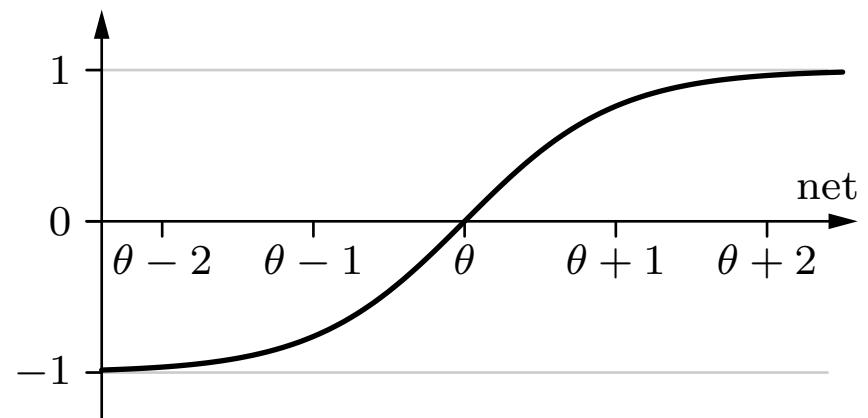


Sigmoid Activation Functions

- All sigmoid functions on the previous slide are **unipolar**, that is, they range from 0 to 1.
- Sometimes **bipolar** sigmoid functions are used (ranging from -1 to $+1$), like the hyperbolic tangent (*tangens hyperbolicus*).

hyperbolic tangent:

$$\begin{aligned}f_{\text{act}}(\text{net}, \theta) &= \tanh(\text{net} - \theta) \\&= \frac{e^{(\text{net} - \theta)} - e^{-(\text{net} - \theta)}}{e^{(\text{net} - \theta)} + e^{-(\text{net} - \theta)}} \\&= \frac{1 - e^{-2(\text{net} - \theta)}}{1 + e^{-2(\text{net} - \theta)}} \\&= \frac{2}{1 + e^{-2(\text{net} - \theta)}} - 1\end{aligned}$$



Multi-layer Perceptrons: Weight Matrices

Let $U_1 = \{v_1, \dots, v_m\}$ and $U_2 = \{u_1, \dots, u_n\}$ be the neurons of two consecutive layers of a multi-layer perceptron.

Their connection weights are represented by an $n \times m$ matrix

$$\mathbf{W} = \begin{pmatrix} w_{u_1 v_1} & w_{u_1 v_2} & \cdots & w_{u_1 v_m} \\ w_{u_2 v_1} & w_{u_2 v_2} & \cdots & w_{u_2 v_m} \\ \vdots & \vdots & & \vdots \\ w_{u_n v_1} & w_{u_n v_2} & \cdots & w_{u_n v_m} \end{pmatrix},$$

where $w_{u_i v_j} = 0$ if there is no connection from neuron v_j to neuron u_i .

Advantage: The computation of the network input can be written as

$$\vec{\text{net}}_{U_2} = \mathbf{W} \cdot \vec{\text{in}}_{U_2} = \mathbf{W} \cdot \vec{\text{out}}_{U_1}$$

where $\vec{\text{net}}_{U_2} = (\text{net}_{u_1}, \dots, \text{net}_{u_n})^\top$ and $\vec{\text{in}}_{U_2} = \vec{\text{out}}_{U_1} = (\text{out}_{v_1}, \dots, \text{out}_{v_m})^\top$.

Why Non-linear Activation Functions?

With weight matrices we have for two consecutive layers U_1 and U_2

$$\vec{\text{net}}_{U_2} = \mathbf{W} \cdot \vec{\text{in}}_{U_2} = \mathbf{W} \cdot \vec{\text{out}}_{U_1}.$$

If the activation functions are linear, that is,

$$f_{\text{act}}(\text{net}, \theta) = \alpha \text{ net} - \theta.$$

the activations of the neurons in the layer U_2 can be computed as

$$\vec{\text{act}}_{U_2} = \mathbf{D}_{\text{act}} \cdot \vec{\text{net}}_{U_2} - \vec{\theta},$$

where

- $\vec{\text{act}}_{U_2} = (\text{act}_{u_1}, \dots, \text{act}_{u_n})^\top$ is the activation vector,
- \mathbf{D}_{act} is an $n \times n$ diagonal matrix of the factors α_{u_i} , $i = 1, \dots, n$, and
- $\vec{\theta} = (\theta_{u_1}, \dots, \theta_{u_n})^\top$ is a bias vector.

Why Non-linear Activation Functions?

If the output function is also linear, it is analogously

$$\vec{\text{out}}_{U_2} = \mathbf{D}_{\text{out}} \cdot \vec{\text{act}}_{U_2} - \vec{\xi},$$

where

- $\vec{\text{out}}_{U_2} = (\text{out}_{u_1}, \dots, \text{out}_{u_n})^\top$ is the output vector,
- \mathbf{D}_{out} is again an $n \times n$ diagonal matrix of factors, and
- $\vec{\xi} = (\xi_{u_1}, \dots, \xi_{u_n})^\top$ a bias vector.

Combining these computations we get

$$\vec{\text{out}}_{U_2} = \mathbf{D}_{\text{out}} \cdot \left(\mathbf{D}_{\text{act}} \cdot \left(\mathbf{W} \cdot \vec{\text{out}}_{U_1} \right) - \vec{\theta} \right) - \vec{\xi}$$

and thus

$$\vec{\text{out}}_{U_2} = \mathbf{A}_{12} \cdot \vec{\text{out}}_{U_1} + \vec{b}_{12}$$

with an $n \times m$ matrix \mathbf{A}_{12} and an n -dimensional vector \vec{b}_{12} .

Why Non-linear Activation Functions?

Therefore we have

$$\vec{\text{out}}_{U_2} = \mathbf{A}_{12} \cdot \vec{\text{out}}_{U_1} + \vec{b}_{12}$$

and

$$\vec{\text{out}}_{U_3} = \mathbf{A}_{23} \cdot \vec{\text{out}}_{U_2} + \vec{b}_{23}$$

for the computations of two consecutive layers U_2 and U_3 .

These two computations can be combined into

$$\vec{\text{out}}_{U_3} = \mathbf{A}_{13} \cdot \vec{\text{out}}_{U_1} + \vec{b}_{13},$$

where $\mathbf{A}_{13} = \mathbf{A}_{23} \cdot \mathbf{A}_{12}$ and $\vec{b}_{13} = \mathbf{A}_{23} \cdot \vec{b}_{12} + \vec{b}_{23}$.

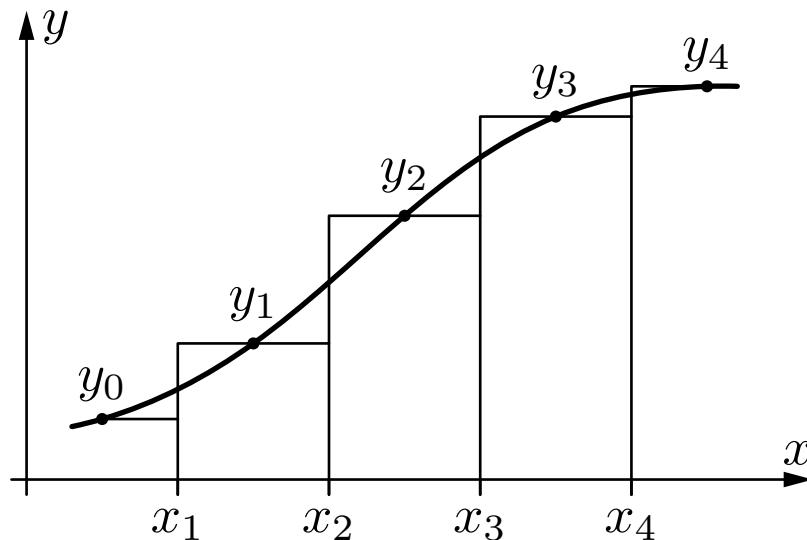
Result: With linear activation and output functions any multi-layer perceptron can be reduced to a two-layer perceptron.

Multi-layer Perceptrons: Function Approximation

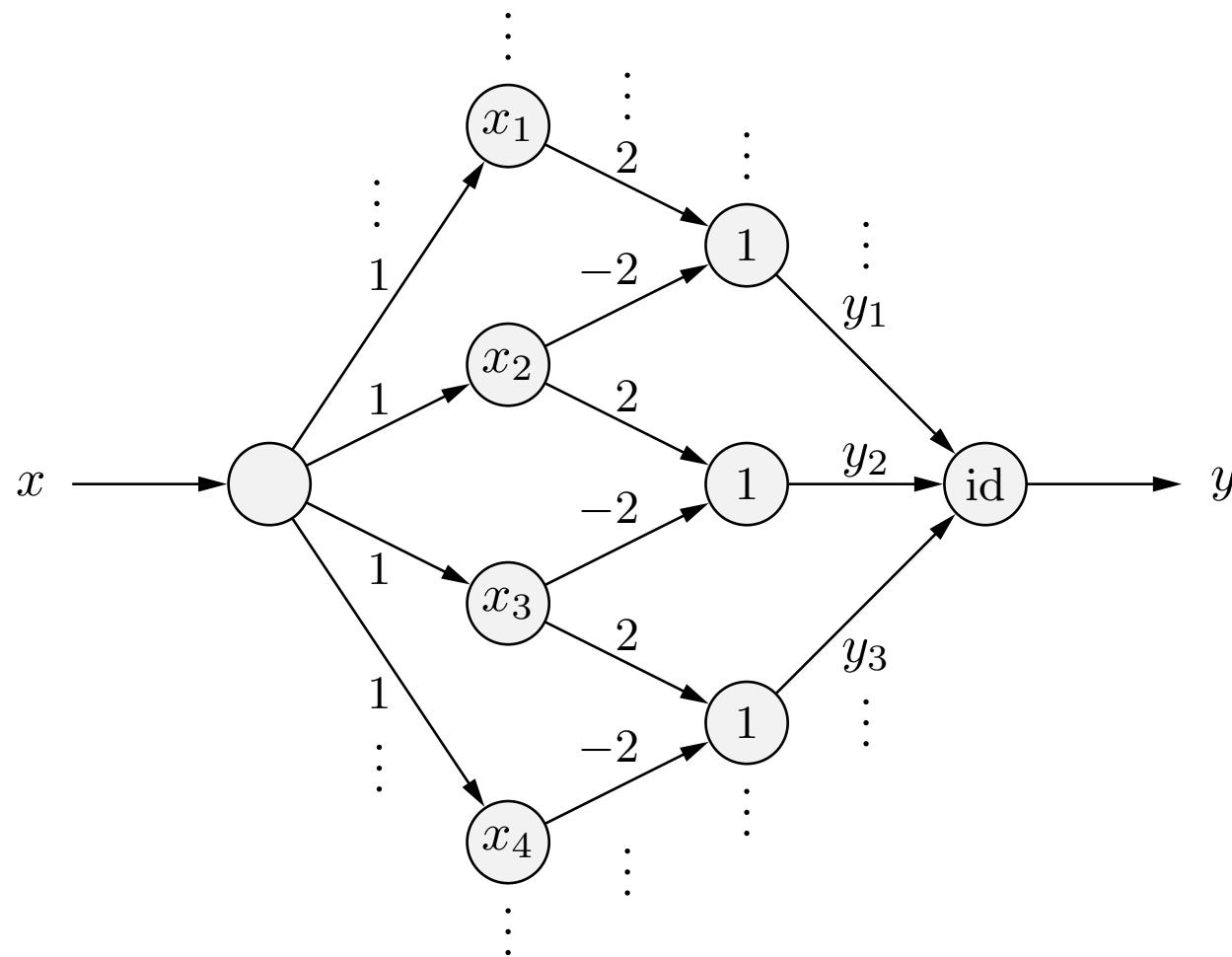
- Up to now: representing and learning Boolean functions $f : \{0, 1\}^n \rightarrow \{0, 1\}$.
- Now: representing and learning real-valued functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

General idea of function approximation:

- Approximate a given function by a step function.
- Construct a neural network that computes the step function.



Multi-layer Perceptrons: Function Approximation

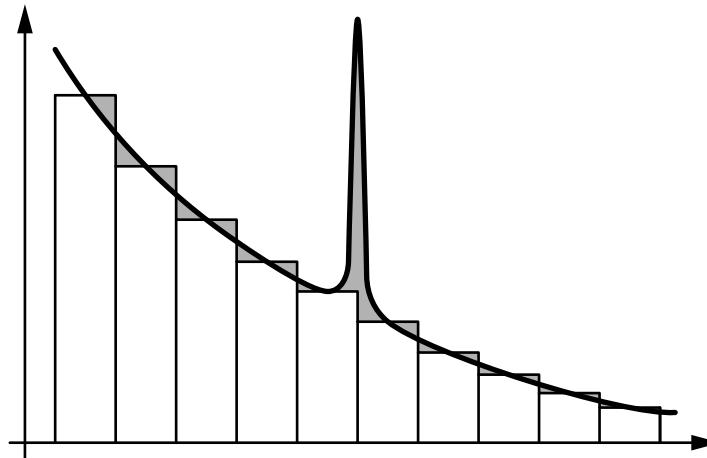


A neural network that computes the step function shown on the preceding slide.
According to the input value only one step is active at any time.
The output neuron has the identity as its activation and output functions.

Multi-layer Perceptrons: Function Approximation

Theorem: Any Riemann-integrable function can be approximated with arbitrary accuracy by a four-layer perceptron.

- But: Error is measured as the **area** between the functions.



- More sophisticated mathematical examination allows a stronger assertion:
With a three-layer perceptron any continuous function can be approximated with arbitrary accuracy (error: maximum function value difference).

Multi-layer Perceptrons as Universal Approximators

Universal Approximation Theorem [Hornik 1991]:

Let $\varphi(\cdot)$ be a continuous, bounded and nonconstant function,
let X denote an arbitrary compact subset of \mathbb{R}^m , and
let $C(X)$ denote the space of continuous functions on X .

Given any function $f \in C(X)$ and $\varepsilon > 0$, there exists an integer N , real constants $v_i, \theta_i \in \mathbb{R}$ and real vectors $\vec{w}_i \in \mathbb{R}^m$, $i = 1, \dots, N$, such that we may define

$$F(\vec{x}) = \sum_{i=1}^N v_i \varphi(\vec{w}_i^\top \vec{x} - \theta_i)$$

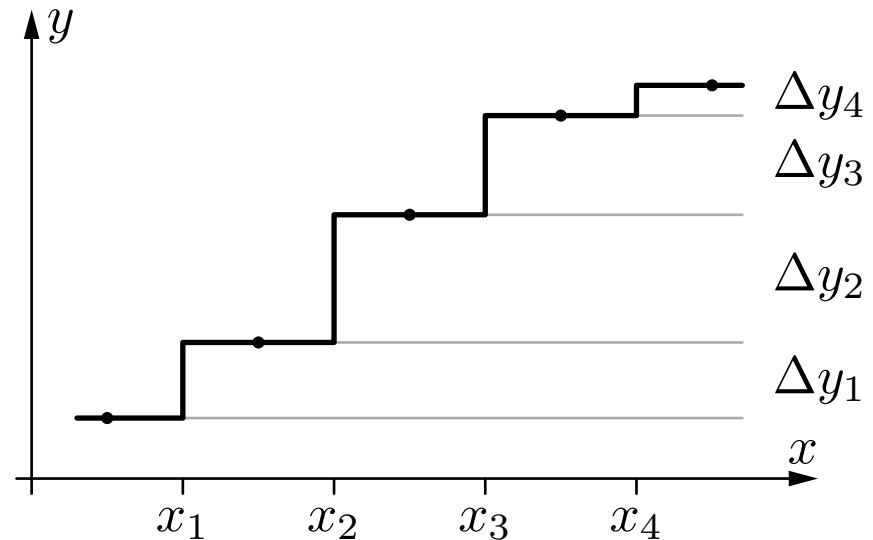
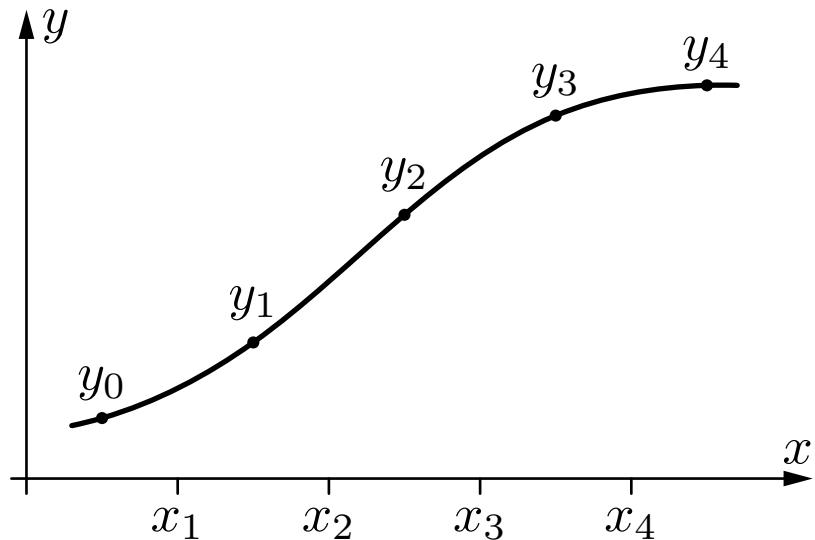
as an approximate realization of the function f where f is independent of φ . That is,

$$|F(\vec{x}) - f(\vec{x})| < \varepsilon$$

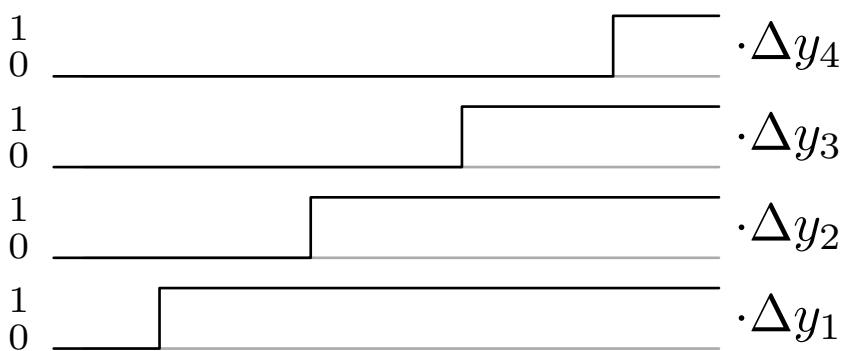
for all $\vec{x} \in X$. In other words, functions of the form $F(\vec{x})$ are dense in $C(X)$.

Note that it is *not* the shape of the activation function, but the layered structure of the feedforward network that renders multi-layer perceptrons universal approximators.

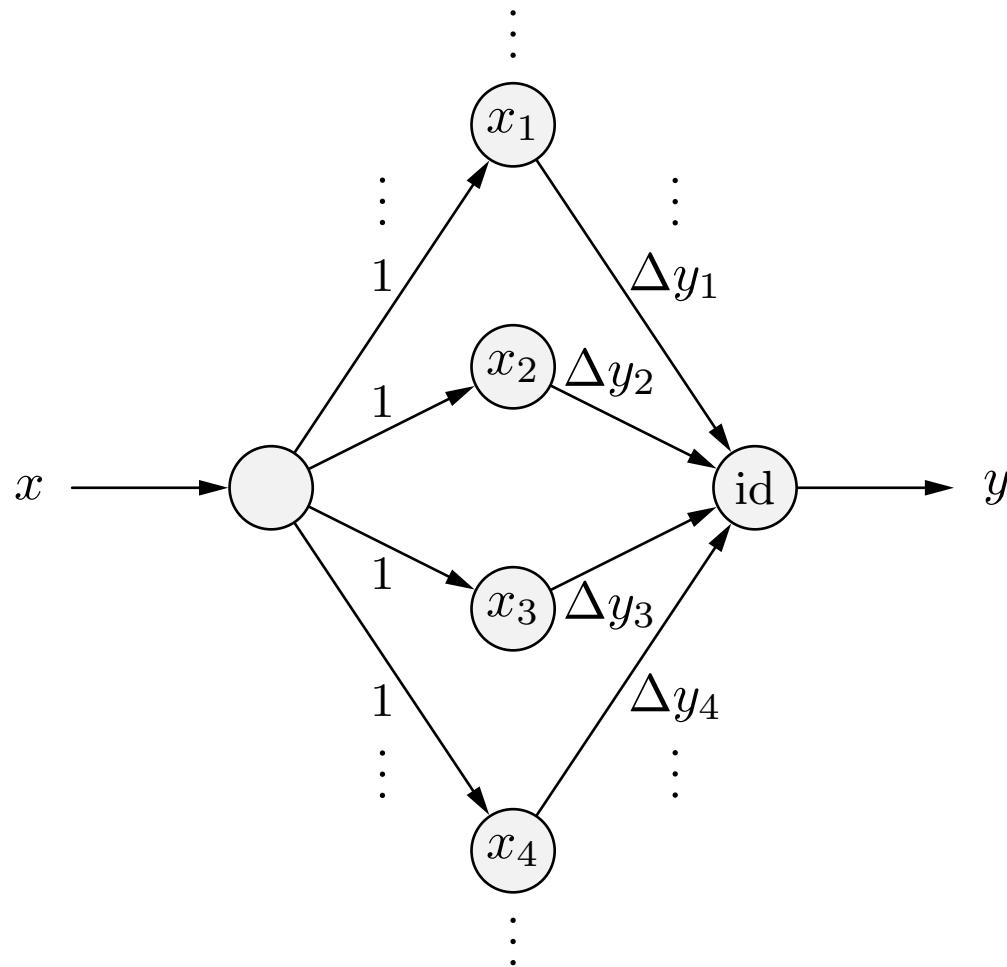
Multi-layer Perceptrons: Function Approximation



By using relative step heights
one layer can be saved.

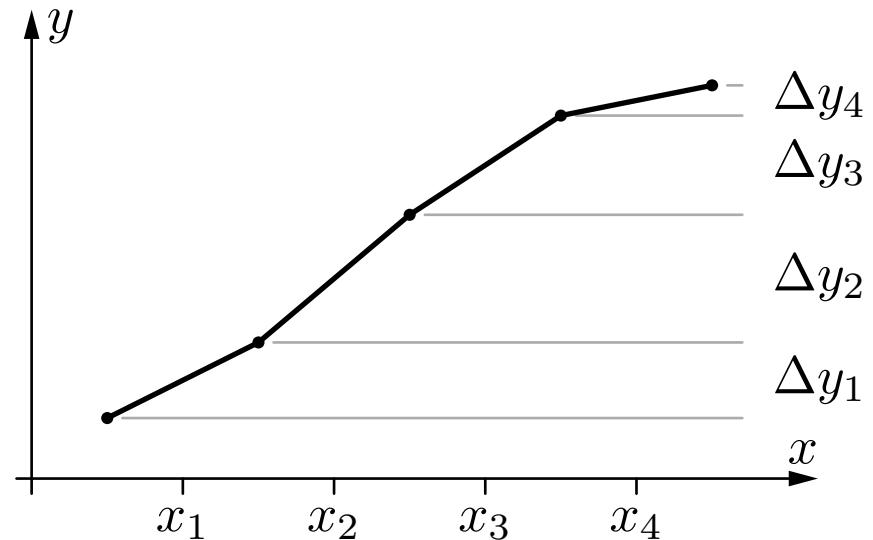
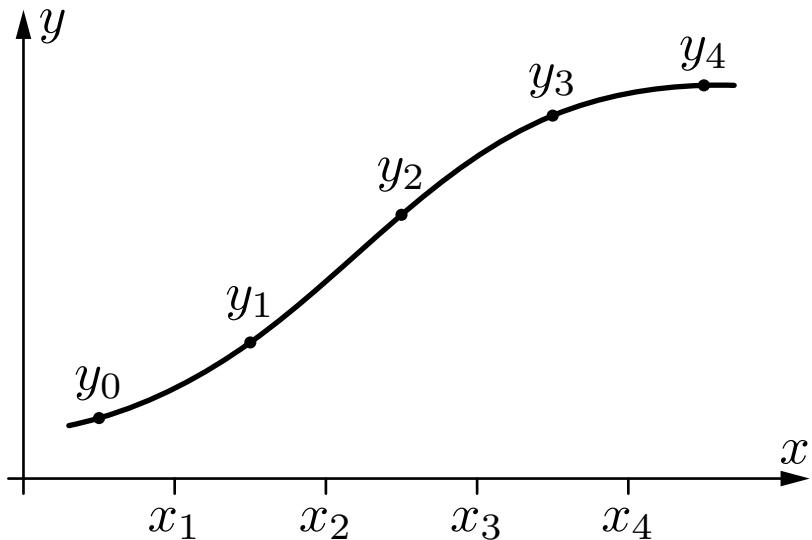


Multi-layer Perceptrons: Function Approximation

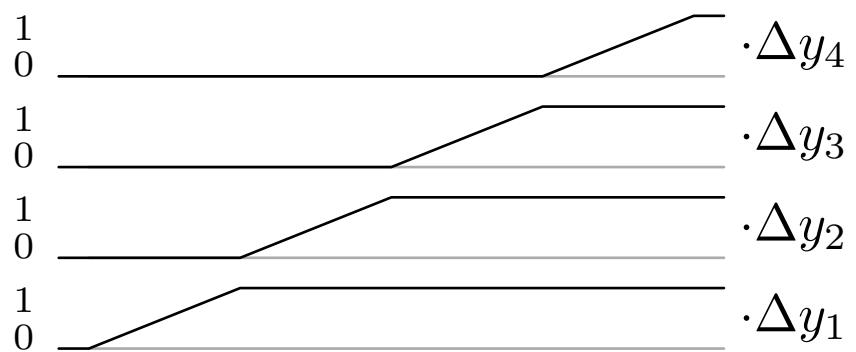


A neural network that computes the step function shown on the preceding slide. The output neuron has the identity as its activation and output functions.

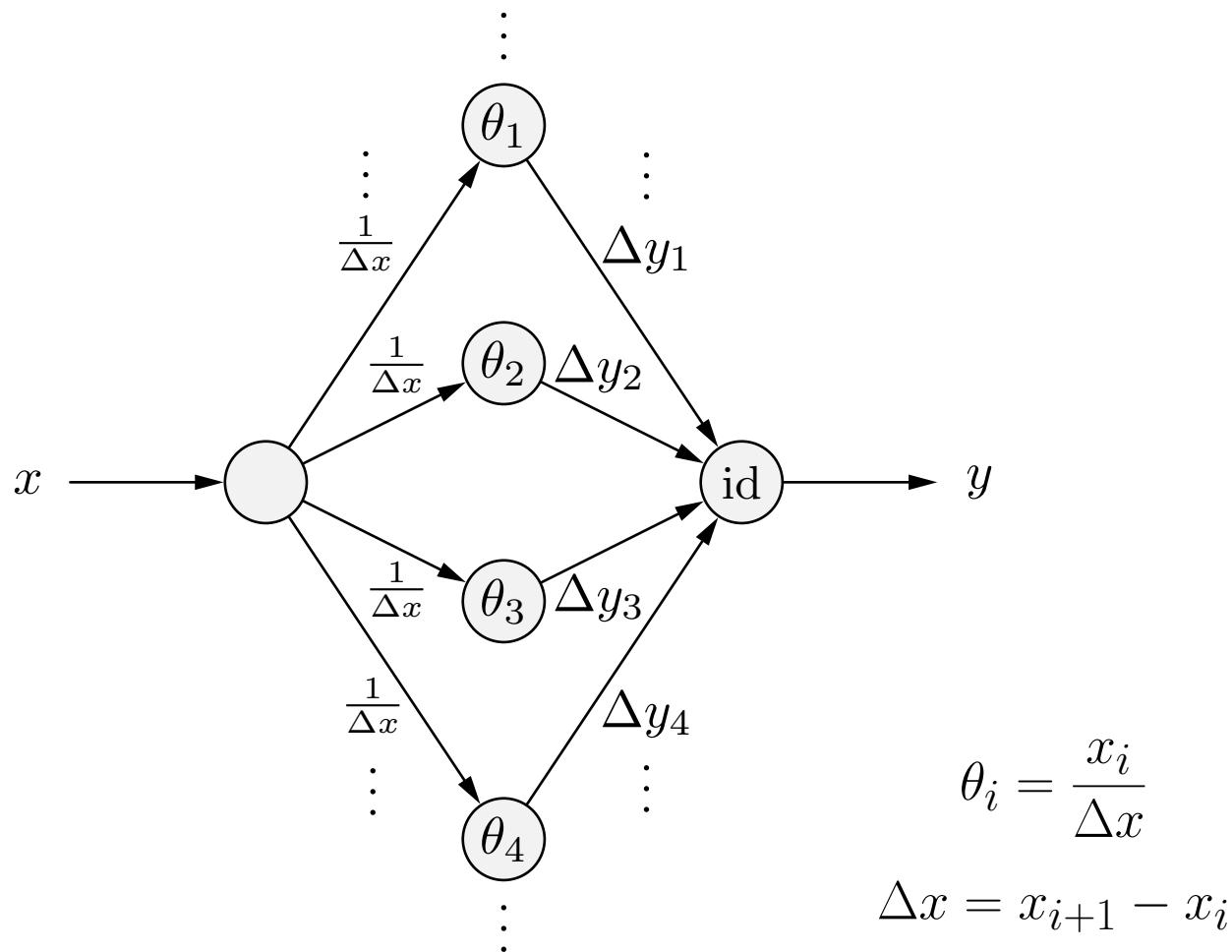
Multi-layer Perceptrons: Function Approximation



By using semi-linear functions the approximation can be improved.



Multi-layer Perceptrons: Function Approximation



A neural network that computes the step function shown on the preceding slide. The output neuron has the identity as its activation and output functions.

Training Multi-layer Perceptrons: Gradient Descent

- Problem of logistic regression: Works only for two-layer perceptrons.
- More general approach: **gradient descent**.
- Necessary condition: **differentiable activation and output functions**.

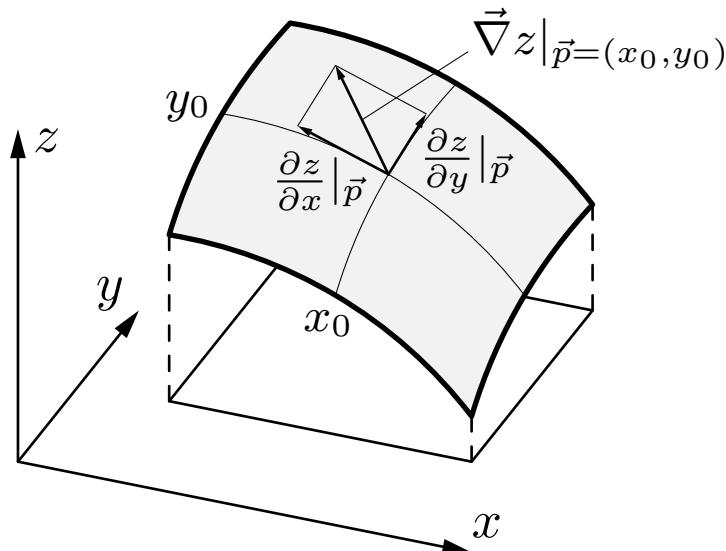


Illustration of the gradient of a real-valued function $z = f(x, y)$ at a point (x_0, y_0) .
It is $\vec{\nabla}z|_{(x_0,y_0)} = \left(\frac{\partial z}{\partial x}|_{x_0}, \frac{\partial z}{\partial y}|_{y_0}\right)$. ($\vec{\nabla}$ is a differential operator called “nabla” or “del”.)

Gradient Descent: Formal Approach

General Idea: Approach the minimum of the error function in small steps.

Error function:

$$e = \sum_{l \in L_{\text{fixed}}} e^{(l)} = \sum_{v \in U_{\text{out}}} e_v = \sum_{l \in L_{\text{fixed}}} \sum_{v \in U_{\text{out}}} e_v^{(l)},$$

Form gradient to determine the direction of the step
(here and in the following: extended weight vector $\vec{w}_u = (-\theta_u, w_{up_1}, \dots, w_{up_n})$):

$$\vec{\nabla}_{\vec{w}_u} e = \frac{\partial e}{\partial \vec{w}_u} = \left(-\frac{\partial e}{\partial \theta_u}, \frac{\partial e}{\partial w_{up_1}}, \dots, \frac{\partial e}{\partial w_{up_n}} \right).$$

Exploit the sum over the training patterns:

$$\vec{\nabla}_{\vec{w}_u} e = \frac{\partial e}{\partial \vec{w}_u} = \frac{\partial}{\partial \vec{w}_u} \sum_{l \in L_{\text{fixed}}} e^{(l)} = \sum_{l \in L_{\text{fixed}}} \frac{\partial e^{(l)}}{\partial \vec{w}_u}.$$

Gradient Descent: Formal Approach

Single pattern error depends on weights only through the network input:

$$\vec{\nabla}_{\vec{w}_u} e^{(l)} = \frac{\partial e^{(l)}}{\partial \vec{w}_u} = \frac{\partial e^{(l)}}{\partial \text{net}_u^{(l)}} \frac{\partial \text{net}_u^{(l)}}{\partial \vec{w}_u}.$$

Since $\text{net}_u^{(l)} = \vec{w}_u^\top \vec{\text{in}}_u^{(l)}$ (note: extended input vector $\vec{\text{in}}_u^{(l)} = (1, \text{in}_{p_1 u}^{(l)}, \dots, \text{in}_{p_n u}^{(l)})$), we have for the second factor

$$\frac{\partial \text{net}_u^{(l)}}{\partial \vec{w}_u} = \vec{\text{in}}_u^{(l)}.$$

For the first factor we consider the error $e^{(l)}$ for the training pattern $l = (\vec{v}^{(l)}, \vec{o}^{(l)})$:

$$e^{(l)} = \sum_{v \in U_{\text{out}}} e_v^{(l)} = \sum_{v \in U_{\text{out}}} \left(o_v^{(l)} - \text{out}_v^{(l)} \right)^2,$$

that is, the sum of the errors over all output neurons.

Gradient Descent: Formal Approach

Therefore we have

$$\frac{\partial e^{(l)}}{\partial \text{net}_u^{(l)}} = \frac{\partial \sum_{v \in U_{\text{out}}} \left(o_v^{(l)} - \text{out}_v^{(l)} \right)^2}{\partial \text{net}_u^{(l)}} = \sum_{v \in U_{\text{out}}} \frac{\partial \left(o_v^{(l)} - \text{out}_v^{(l)} \right)^2}{\partial \text{net}_u^{(l)}}.$$

Since only the actual output $\text{out}_v^{(l)}$ of an output neuron v depends on the network input $\text{net}_u^{(l)}$ of the neuron u we are considering, it is

$$\frac{\partial e^{(l)}}{\partial \text{net}_u^{(l)}} = -2 \underbrace{\sum_{v \in U_{\text{out}}} \left(o_v^{(l)} - \text{out}_v^{(l)} \right) \frac{\partial \text{out}_v^{(l)}}{\partial \text{net}_u^{(l)}}}_{\delta_u^{(l)}},$$

which also introduces the abbreviation $\delta_u^{(l)}$ for the important sum appearing here.

Gradient Descent: Formal Approach

Distinguish two cases:

- The neuron u is an **output neuron**.
- The neuron u is a **hidden neuron**.

In the first case we have

$$\forall u \in U_{\text{out}} : \quad \delta_u^{(l)} = \left(o_u^{(l)} - \text{out}_u^{(l)} \right) \frac{\partial \text{out}_u^{(l)}}{\partial \text{net}_u^{(l)}}$$

Therefore we have for the gradient

$$\forall u \in U_{\text{out}} : \quad \vec{\nabla}_{\vec{w}_u} e_u^{(l)} = \frac{\partial e_u^{(l)}}{\partial \vec{w}_u} = -2 \left(o_u^{(l)} - \text{out}_u^{(l)} \right) \frac{\partial \text{out}_u^{(l)}}{\partial \text{net}_u^{(l)}} \vec{\text{in}}_u^{(l)}$$

and thus for the weight change

$$\forall u \in U_{\text{out}} : \quad \Delta \vec{w}_u^{(l)} = -\frac{\eta}{2} \vec{\nabla}_{\vec{w}_u} e_u^{(l)} = \eta \left(o_u^{(l)} - \text{out}_u^{(l)} \right) \frac{\partial \text{out}_u^{(l)}}{\partial \text{net}_u^{(l)}} \vec{\text{in}}_u^{(l)}.$$

Gradient Descent: Formal Approach

Exact formulae depend on the choice of the activation and the output function, since it is

$$\text{out}_u^{(l)} = f_{\text{out}}(\text{act}_u^{(l)}) = f_{\text{out}}(f_{\text{act}}(\text{net}_u^{(l)})).$$

Consider the special case with

- output function is the identity,
- activation function is logistic, that is, $f_{\text{act}}(x) = \frac{1}{1+e^{-x}}$.

The first assumption yields

$$\frac{\partial \text{out}_u^{(l)}}{\partial \text{net}_u^{(l)}} = \frac{\partial \text{act}_u^{(l)}}{\partial \text{net}_u^{(l)}} = f'_{\text{act}}(\text{net}_u^{(l)}).$$

Gradient Descent: Formal Approach

For a logistic activation function we have

$$\begin{aligned} f'_{\text{act}}(x) &= \frac{d}{dx} (1 + e^{-x})^{-1} = - (1 + e^{-x})^{-2} (-e^{-x}) \\ &= \frac{1 + e^{-x} - 1}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \left(1 - \frac{1}{1 + e^{-x}} \right) \\ &= f_{\text{act}}(x) \cdot (1 - f_{\text{act}}(x)), \end{aligned}$$

and therefore

$$f'_{\text{act}}(\text{net}_u^{(l)}) = f_{\text{act}}(\text{net}_u^{(l)}) \cdot \left(1 - f_{\text{act}}(\text{net}_u^{(l)}) \right) = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)} \right).$$

The resulting weight change is therefore

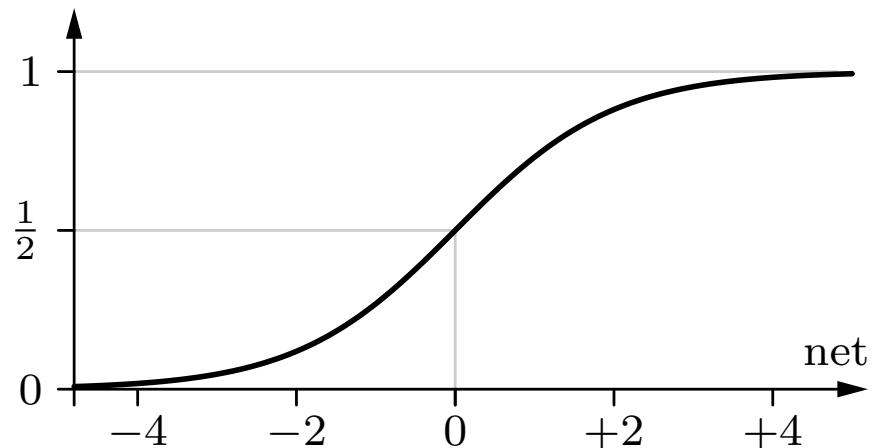
$$\Delta \vec{w}_u^{(l)} = \eta \left(o_u^{(l)} - \text{out}_u^{(l)} \right) \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)} \right) \vec{\text{in}}_u^{(l)},$$

which makes the computations very simple.

Gradient Descent: Formal Approach

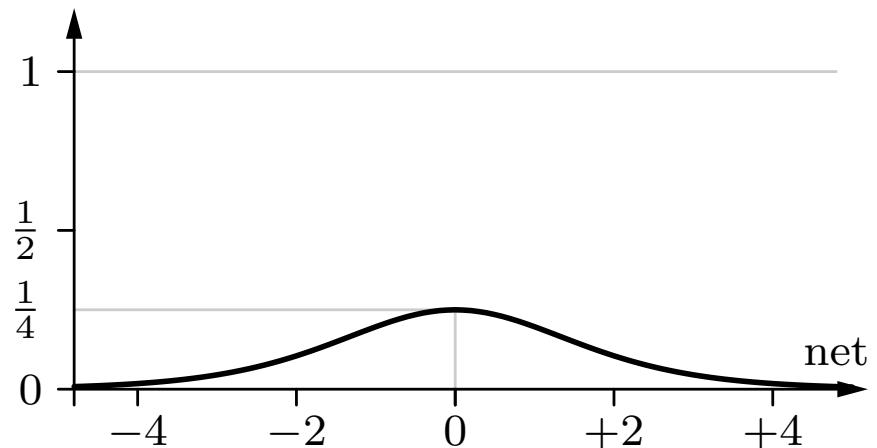
logistic activation function:

$$f_{\text{act}}(\text{net}_u^{(l)}) = \frac{1}{1 + e^{-\text{net}_u^{(l)}}}$$



derivative of logistic function:

$$f'_{\text{act}}(\text{net}_u^{(l)}) = f_{\text{act}}(\text{net}_u^{(l)}) \cdot (1 - f_{\text{act}}(\text{net}_u^{(l)}))$$

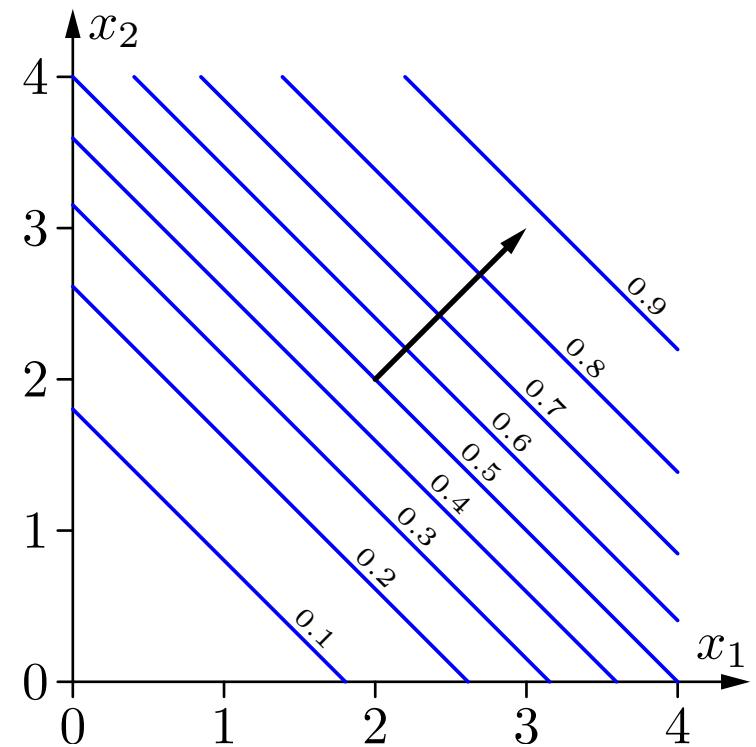
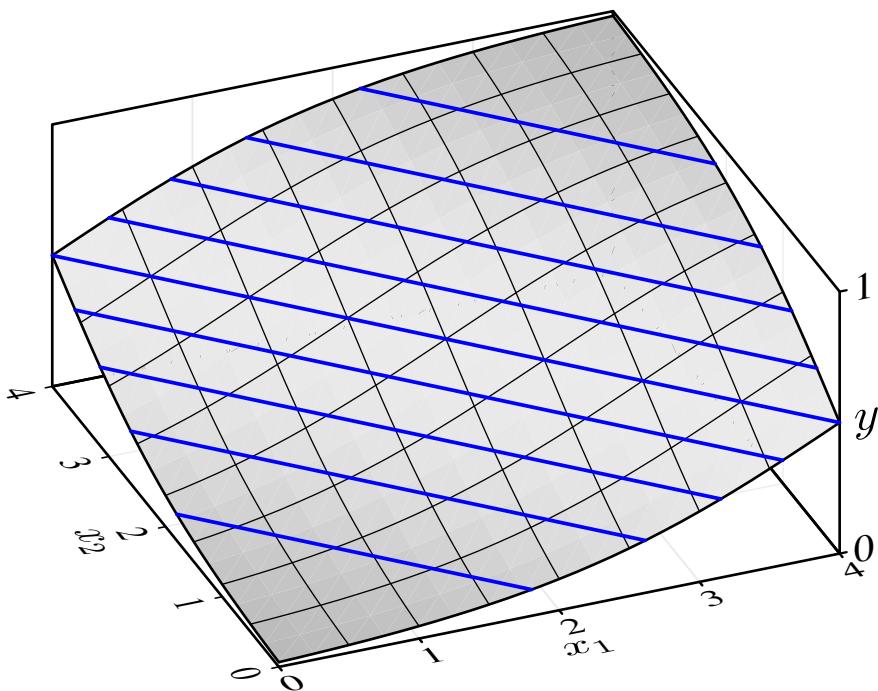


- If a logistic activation function is used (shown on left), the weight changes are proportional to $\lambda_u^{(l)} = \text{out}_u^{(l)} (1 - \text{out}_u^{(l)})$ (shown on right; see preceding slide).
- Weight changes are largest, and thus the training speed highest, in the vicinity of $\text{net}_u^{(l)} = 0$. Far away from $\text{net}_u^{(l)} = 0$, the gradient becomes (very) small (“saturation regions”) and thus training (very) slow.

Reminder: Two-dimensional Logistic Function

Example logistic function for two arguments x_1 and x_2 :

$$y = \frac{1}{1 + \exp(4 - x_1 - x_2)} = \frac{1}{1 + \exp(4 - (1, 1)^\top (x_1, x_2))}$$



The blue lines show where the logistic function has a certain value in $\{0.1, \dots, 0.9\}$.

Error Backpropagation

Consider now: The neuron u is a **hidden neuron**, that is, $u \in U_k$, $0 < k < r - 1$.

The output $\text{out}_v^{(l)}$ of an output neuron v depends on the network input $\text{net}_u^{(l)}$ only indirectly through the successor neurons $\text{succ}(u) = \{s \in U \mid (u, s) \in C\} = \{s_1, \dots, s_m\} \subseteq U_{k+1}$, namely through their network inputs $\text{net}_s^{(l)}$.

We apply the chain rule to obtain

$$\delta_u^{(l)} = \sum_{v \in U_{\text{out}}} \sum_{s \in \text{succ}(u)} (o_v^{(l)} - \text{out}_v^{(l)}) \frac{\partial \text{out}_v^{(l)}}{\partial \text{net}_s^{(l)}} \frac{\partial \text{net}_s^{(l)}}{\partial \text{net}_u^{(l)}}.$$

Exchanging the sums yields

$$\delta_u^{(l)} = \sum_{s \in \text{succ}(u)} \left(\sum_{v \in U_{\text{out}}} (o_v^{(l)} - \text{out}_v^{(l)}) \frac{\partial \text{out}_v^{(l)}}{\partial \text{net}_s^{(l)}} \right) \frac{\partial \text{net}_s^{(l)}}{\partial \text{net}_u^{(l)}} = \sum_{s \in \text{succ}(u)} \delta_s^{(l)} \frac{\partial \text{net}_s^{(l)}}{\partial \text{net}_u^{(l)}}.$$

Error Backpropagation

Consider the network input

$$\text{net}_s^{(l)} = \vec{w}_s^\top \vec{\text{in}}_s^{(l)} = \left(\sum_{p \in \text{pred}(s)} w_{sp} \text{out}_p^{(l)} \right) - \theta_s,$$

where one element of $\vec{\text{in}}_s^{(l)}$ is the output $\text{out}_u^{(l)}$ of the neuron u . Therefore it is

$$\frac{\partial \text{net}_s^{(l)}}{\partial \text{net}_u^{(l)}} = \left(\sum_{p \in \text{pred}(s)} w_{sp} \frac{\partial \text{out}_p^{(l)}}{\partial \text{net}_u^{(l)}} \right) - \frac{\partial \theta_s}{\partial \text{net}_u^{(l)}} = w_{su} \frac{\partial \text{out}_u^{(l)}}{\partial \text{net}_u^{(l)}},$$

The result is the recursive equation (error backpropagation)

$$\delta_u^{(l)} = \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su} \right) \frac{\partial \text{out}_u^{(l)}}{\partial \text{net}_u^{(l)}}.$$

Error Backpropagation

The resulting formula for the weight change is

$$\Delta \vec{w}_u^{(l)} = -\frac{\eta}{2} \vec{\nabla}_{\vec{w}_u} e^{(l)} = \eta \delta_u^{(l)} \vec{\text{in}}_u^{(l)} = \eta \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su} \right) \frac{\partial \text{out}_u^{(l)}}{\partial \text{net}_u^{(l)}} \vec{\text{in}}_u^{(l)}.$$

Consider again the special case with

- output function is the identity,
- activation function is logistic.

The resulting formula for the weight change is then

$$\Delta \vec{w}_u^{(l)} = \eta \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su} \right) \text{out}_u^{(l)} (1 - \text{out}_u^{(l)}) \vec{\text{in}}_u^{(l)}.$$

Error Backpropagation: Cookbook Recipe

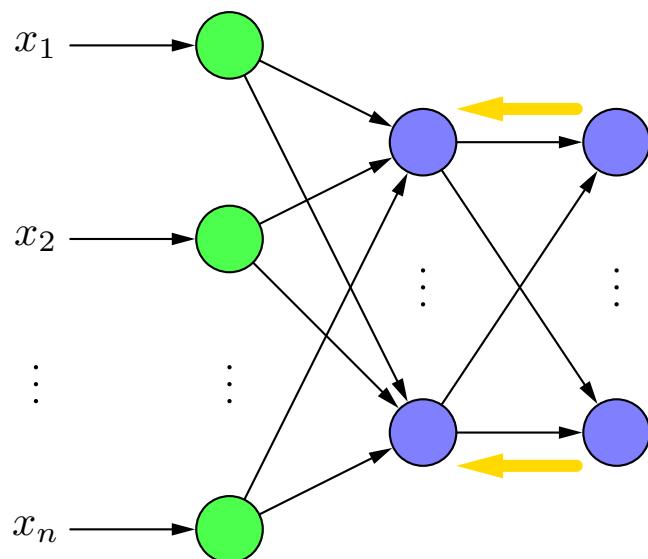
$$\forall u \in U_{\text{in}} : \quad$$

$$\text{out}_u^{(l)} = \text{ext}_u^{(l)}$$

forward propagation:

$$\forall u \in U_{\text{hidden}} \cup U_{\text{out}} : \quad$$

$$\text{out}_u^{(l)} = \left(1 + \exp \left(- \sum_{p \in \text{pred}(u)} w_{up} \text{out}_p^{(l)} \right) \right)^{-1}$$



backward

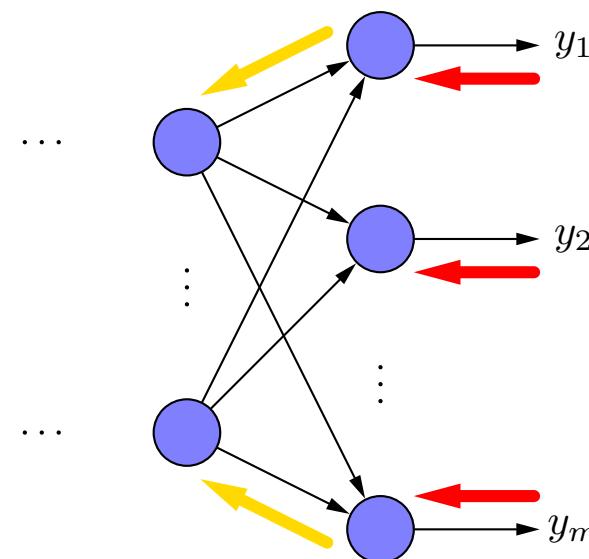
propagation:

$$\forall u \in U_{\text{hidden}} : \quad$$

$$\delta_u^{(l)} = \left(\sum_{s \in \text{succ}(u)} \delta_s^{(l)} w_{su} \right) \lambda_u^{(l)}$$

activation derivative:

$$\lambda_u^{(l)} = \text{out}_u^{(l)} \left(1 - \text{out}_u^{(l)} \right)$$



- logistic activation function

- implicit bias value

error factor:

$$\forall u \in U_{\text{out}} : \quad$$

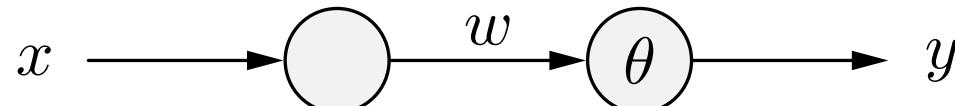
$$\delta_u^{(l)} = \left(o_u^{(l)} - \text{out}_u^{(l)} \right) \lambda_u^{(l)}$$

weight change:

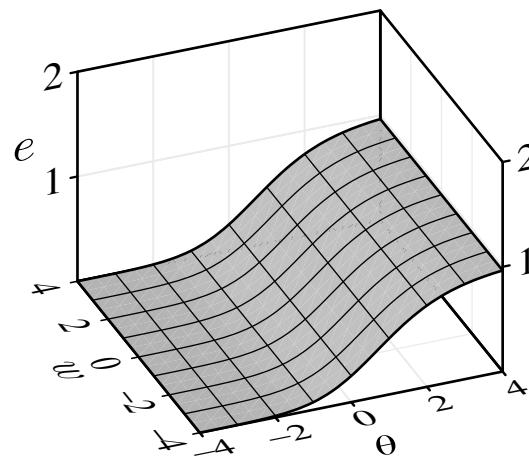
$$\Delta w_{up}^{(l)} = \eta \delta_u^{(l)} \text{out}_p^{(l)}$$

Gradient Descent: Examples

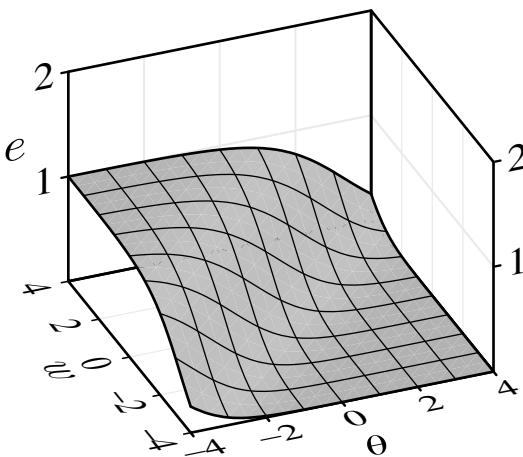
Gradient descent training for the negation $\neg x$



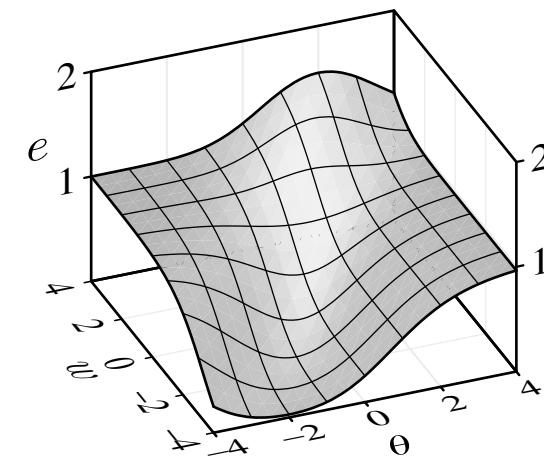
x	y
0	1
1	0



error for $x = 0$



error for $x = 1$



sum of errors

Note: error for $x = 0$ and $x = 1$ is effectively the squared logistic activation function!

Gradient Descent: Examples

epoch	θ	w	error
0	3.00	3.50	1.307
20	3.77	2.19	0.986
40	3.71	1.81	0.970
60	3.50	1.53	0.958
80	3.15	1.24	0.937
100	2.57	0.88	0.890
120	1.48	0.25	0.725
140	-0.06	-0.98	0.331
160	-0.80	-2.07	0.149
180	-1.19	-2.74	0.087
200	-1.44	-3.20	0.059
220	-1.62	-3.54	0.044

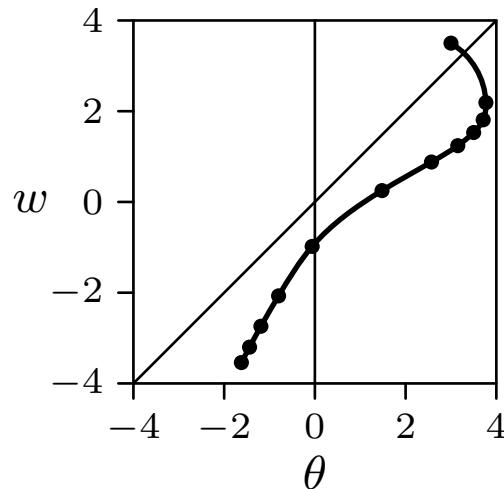
Online Training

epoch	θ	w	error
0	3.00	3.50	1.295
20	3.76	2.20	0.985
40	3.70	1.82	0.970
60	3.48	1.53	0.957
80	3.11	1.25	0.934
100	2.49	0.88	0.880
120	1.27	0.22	0.676
140	-0.21	-1.04	0.292
160	-0.86	-2.08	0.140
180	-1.21	-2.74	0.084
200	-1.45	-3.19	0.058
220	-1.63	-3.53	0.044

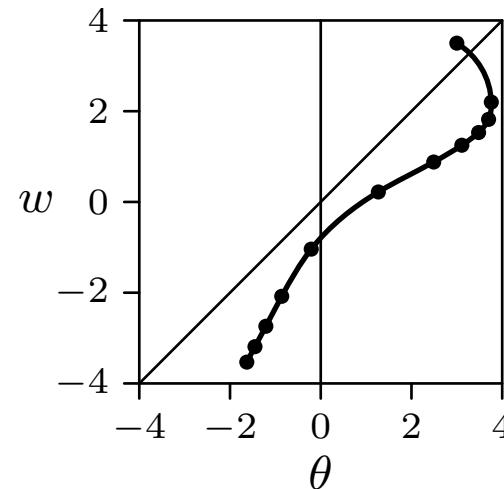
Batch Training

Gradient Descent: Examples

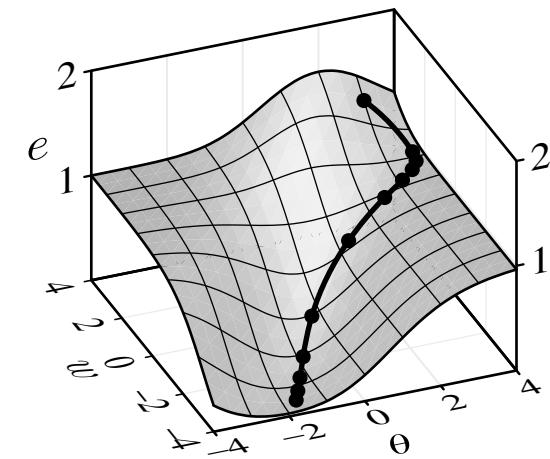
Visualization of gradient descent for the negation $\neg x$



Online Training



Batch Training



Batch Training

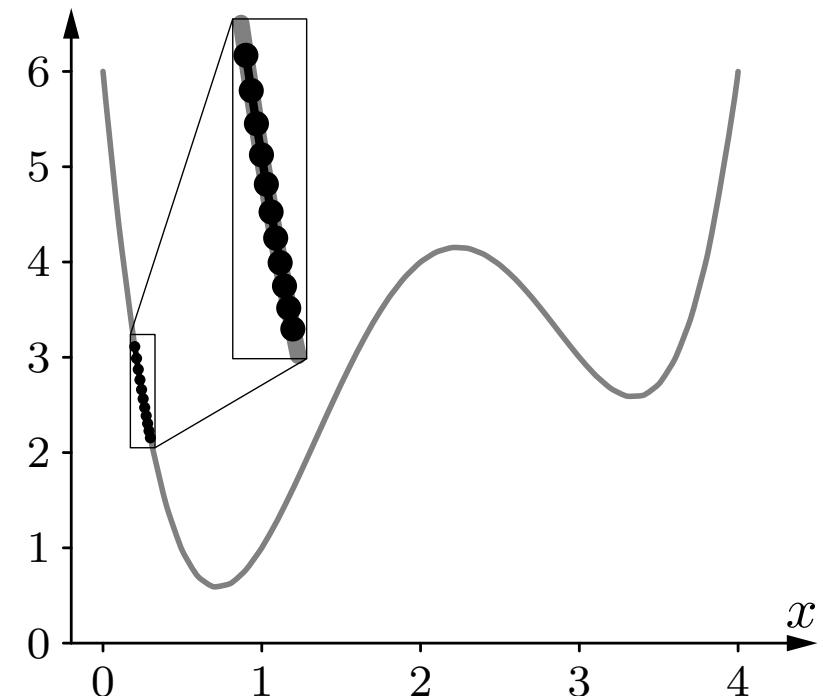
- Training is obviously successful.
- Error cannot vanish completely due to the properties of the logistic function.

Gradient Descent: Examples

Example function:

$$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	0.200	3.112	-11.147	0.011
1	0.211	2.990	-10.811	0.011
2	0.222	2.874	-10.490	0.010
3	0.232	2.766	-10.182	0.010
4	0.243	2.664	-9.888	0.010
5	0.253	2.568	-9.606	0.010
6	0.262	2.477	-9.335	0.009
7	0.271	2.391	-9.075	0.009
8	0.281	2.309	-8.825	0.009
9	0.289	2.233	-8.585	0.009
10	0.298	2.160		



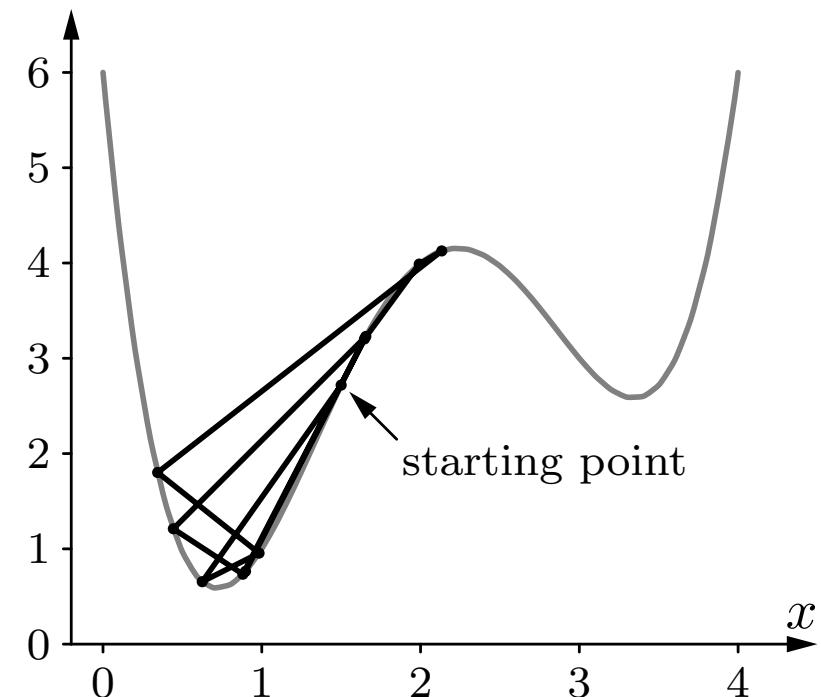
Gradient descent with initial value 0.2 and learning rate 0.001.

Gradient Descent: Examples

Example function:

$$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	1.500	2.719	3.500	-0.875
1	0.625	0.655	-1.431	0.358
2	0.983	0.955	2.554	-0.639
3	0.344	1.801	-7.157	1.789
4	2.134	4.127	0.567	-0.142
5	1.992	3.989	1.380	-0.345
6	1.647	3.203	3.063	-0.766
7	0.881	0.734	1.753	-0.438
8	0.443	1.211	-4.851	1.213
9	1.656	3.231	3.029	-0.757
10	0.898	0.766		



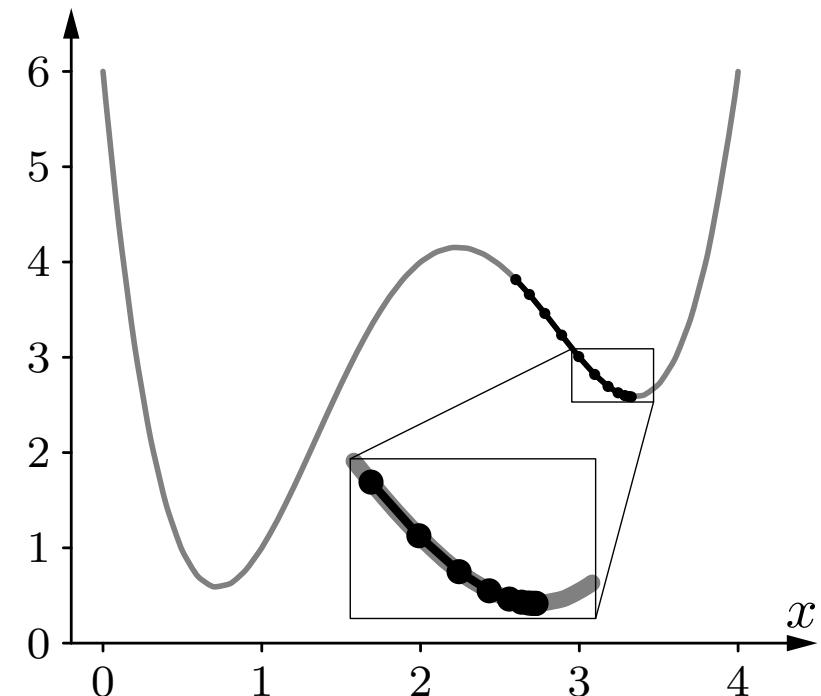
Gradient descent with initial value 1.5 and learning rate 0.25.

Gradient Descent: Examples

Example function:

$$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	2.600	3.816	-1.707	0.085
1	2.685	3.660	-1.947	0.097
2	2.783	3.461	-2.116	0.106
3	2.888	3.233	-2.153	0.108
4	2.996	3.008	-2.009	0.100
5	3.097	2.820	-1.688	0.084
6	3.181	2.695	-1.263	0.063
7	3.244	2.628	-0.845	0.042
8	3.286	2.599	-0.515	0.026
9	3.312	2.589	-0.293	0.015
10	3.327	2.585		



Gradient descent with initial value 2.6 and learning rate 0.05.

Gradient Descent: Variants

Weight update rule:

$$w(t + 1) = w(t) + \Delta w(t)$$

Standard backpropagation:

$$\Delta w(t) = -\frac{\eta}{2} \nabla_w e(t)$$

Manhattan training:

$$\Delta w(t) = -\eta \operatorname{sgn}(\nabla_w e(t)).$$

Fixed step width (grid), only sign of gradient (direction) is evaluated.

Momentum term:

$$\Delta w(t) = -\frac{\eta}{2} \nabla_w e(t) + \beta \Delta w(t - 1),$$

Part of previous change is added, may lead to accelerated training ($\beta \in [0.5, 0.95]$).

Gradient Descent: Variants

Self-adaptive error backpropagation:

$$\eta_w(t) = \begin{cases} c^- \cdot \eta_w(t-1), & \text{if } \nabla_w e(t) \cdot \nabla_w e(t-1) < 0, \\ c^+ \cdot \eta_w(t-1), & \text{if } \nabla_w e(t) \cdot \nabla_w e(t-1) > 0 \\ & \wedge \quad \nabla_w e(t-1) \cdot \nabla_w e(t-2) \geq 0, \\ \eta_w(t-1), & \text{otherwise.} \end{cases}$$

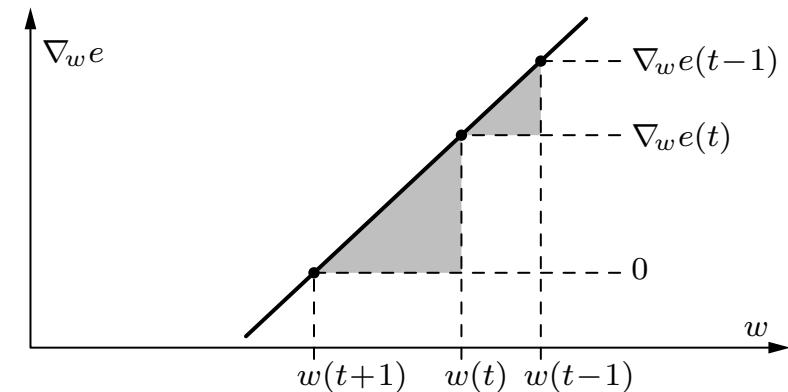
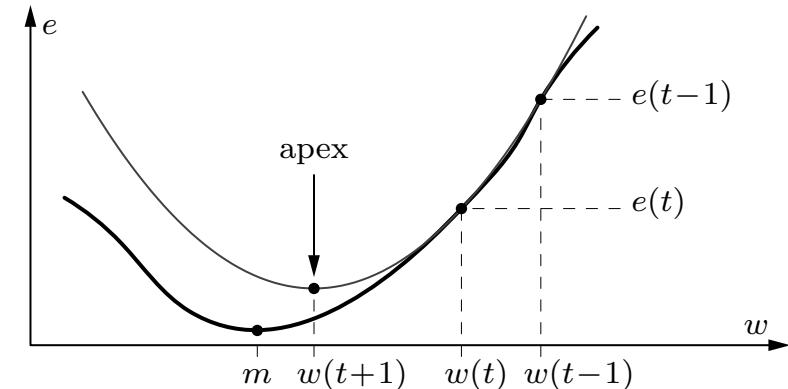
Resilient error backpropagation:

$$\Delta w(t) = \begin{cases} c^- \cdot \Delta w(t-1), & \text{if } \nabla_w e(t) \cdot \nabla_w e(t-1) < 0, \\ c^+ \cdot \Delta w(t-1), & \text{if } \nabla_w e(t) \cdot \nabla_w e(t-1) > 0 \\ & \wedge \quad \nabla_w e(t-1) \cdot \nabla_w e(t-2) \geq 0, \\ \Delta w(t-1), & \text{otherwise.} \end{cases}$$

Typical values: $c^- \in [0.5, 0.7]$ and $c^+ \in [1.05, 1.2]$.

Gradient Descent: Variants

Quickpropagation



The weight update rule can be derived from the triangles:

$$\Delta w(t) = \frac{\nabla_w e(t)}{\nabla_w e(t-1) - \nabla_w e(t)} \cdot \Delta w(t-1).$$

Gradient Descent: Examples

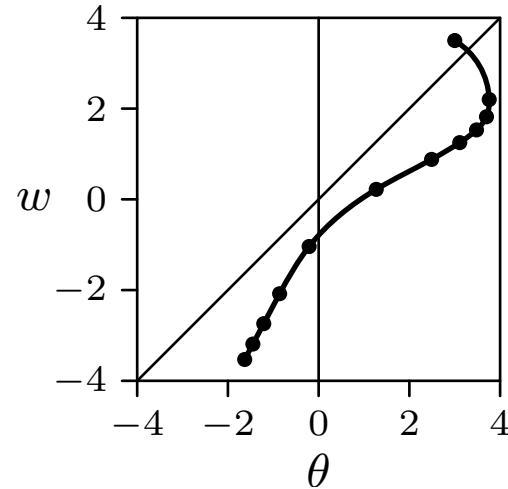
epoch	θ	w	error
0	3.00	3.50	1.295
20	3.76	2.20	0.985
40	3.70	1.82	0.970
60	3.48	1.53	0.957
80	3.11	1.25	0.934
100	2.49	0.88	0.880
120	1.27	0.22	0.676
140	-0.21	-1.04	0.292
160	-0.86	-2.08	0.140
180	-1.21	-2.74	0.084
200	-1.45	-3.19	0.058
220	-1.63	-3.53	0.044

without momentum term

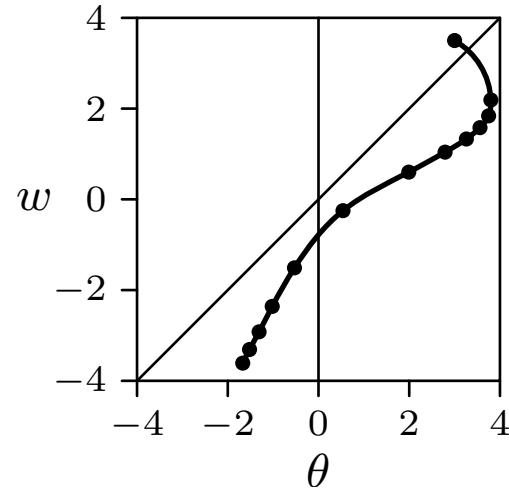
epoch	θ	w	error
0	3.00	3.50	1.295
10	3.80	2.19	0.984
20	3.75	1.84	0.971
30	3.56	1.58	0.960
40	3.26	1.33	0.943
50	2.79	1.04	0.910
60	1.99	0.60	0.814
70	0.54	-0.25	0.497
80	-0.53	-1.51	0.211
90	-1.02	-2.36	0.113
100	-1.31	-2.92	0.073
110	-1.52	-3.31	0.053
120	-1.67	-3.61	0.041

with momentum term ($\beta = 0.9$)

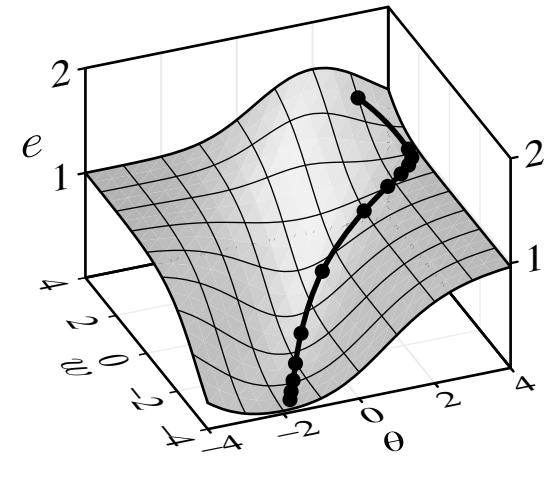
Gradient Descent: Examples



without momentum term



with momentum term



with momentum term

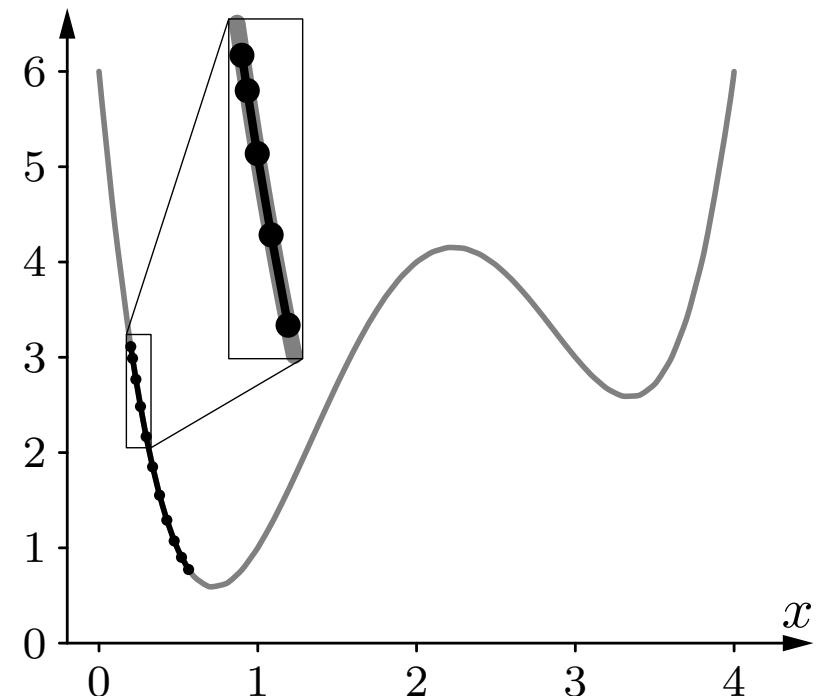
- Dots show position every 20 (without momentum term) or every 10 epochs (with momentum term).
- Learning with a momentum term ($\beta = 0.9$) is about twice as fast.

Gradient Descent: Examples

Example function:

$$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	0.200	3.112	-11.147	0.011
1	0.211	2.990	-10.811	0.021
2	0.232	2.771	-10.196	0.029
3	0.261	2.488	-9.368	0.035
4	0.296	2.173	-8.397	0.040
5	0.337	1.856	-7.348	0.044
6	0.380	1.559	-6.277	0.046
7	0.426	1.298	-5.228	0.046
8	0.472	1.079	-4.235	0.046
9	0.518	0.907	-3.319	0.045
10	0.562	0.777		



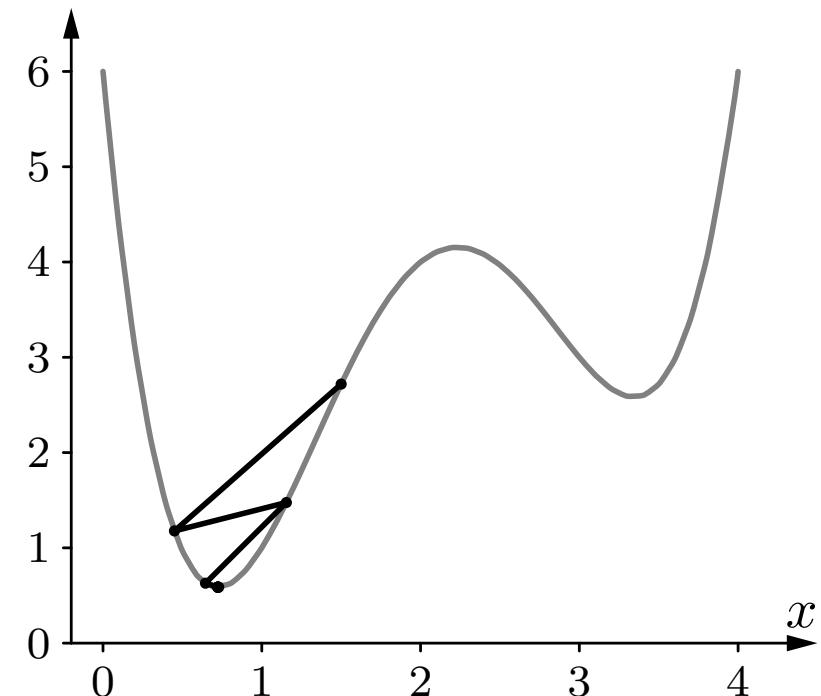
Gradient descent with initial value 0.2, learning rate 0.001
and momentum term $\beta = 0.9$.

Gradient Descent: Examples

Example function:

$$f(x) = \frac{5}{6}x^4 - 7x^3 + \frac{115}{6}x^2 - 18x + 6,$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	1.500	2.719	3.500	-1.050
1	0.450	1.178	-4.699	0.705
2	1.155	1.476	3.396	-0.509
3	0.645	0.629	-1.110	0.083
4	0.729	0.587	0.072	-0.005
5	0.723	0.587	0.001	0.000
6	0.723	0.587	0.000	0.000
7	0.723	0.587	0.000	0.000
8	0.723	0.587	0.000	0.000
9	0.723	0.587	0.000	0.000
10	0.723	0.587		



Gradient descent with initial value 1.5, initial learning rate 0.25, and self-adapting learning rate ($c^+ = 1.2$, $c^- = 0.5$).

Other Extensions of Error Backpropagation

Flat Spot Elimination:

$$\Delta w(t) = -\frac{\eta}{2} \nabla_w e(t) + \zeta$$

- Eliminates slow learning in saturation region of logistic function ($\zeta \approx 0.1$).
- Counteracts the decay of the error signals over the layers.

Weight Decay:

$$\Delta w(t) = -\frac{\eta}{2} \nabla_w e(t) - \xi w(t),$$

- Helps to improve the robustness of the training results ($\xi \leq 10^{-3}$).
- Can be derived from an extended error function penalizing large weights:

$$e^* = e + \frac{\xi}{2} \sum_{u \in U_{\text{out}} \cup U_{\text{hidden}}} \left(\theta_u^2 + \sum_{p \in \text{pred}(u)} w_{up}^2 \right).$$

Number of Hidden Neurons

- Note that the approximation theorem only states that there *exists* a number of hidden neurons and weight vectors \vec{v} and \vec{w}_i and thresholds θ_i , but not how they are to be chosen for a given ε of approximation accuracy.
- For a single hidden layer the following **rule of thumb** is popular:
number of hidden neurons = (number of inputs + number of outputs) / 2
- Better, though computationally expensive approach:
 - Randomly split the given data into two subsets of (about) equal size, the **training data** and the **validation data**.
 - Train multi-layer perceptrons with different numbers of hidden neurons on the training data and evaluate them on the validation data.
 - Repeat the random split of the data and training/evaluation many times and average the results over the same number of hidden neurons. Choose the number of hidden neurons with the best average error.
 - Train a final multi-layer perceptron on the whole data set.

Number of Hidden Neurons

Principle of training data/validation data approach:

- **Underfitting:** If the number of neurons in the hidden layer is too small, the multi-layer perceptron may not be able to capture the structure of the relationship between inputs and outputs precisely enough due to a lack of parameters.
- **Overfitting:** With a larger number of hidden neurons a multi-layer perceptron may adapt not only to the regular dependence between inputs and outputs, but also to the accidental specifics (errors and deviations) of the training data set.
- Overfitting will usually lead to the effect that the error a multi-layer perceptron yields on the validation data will be (possibly considerably) greater than the error it yields on the training data.

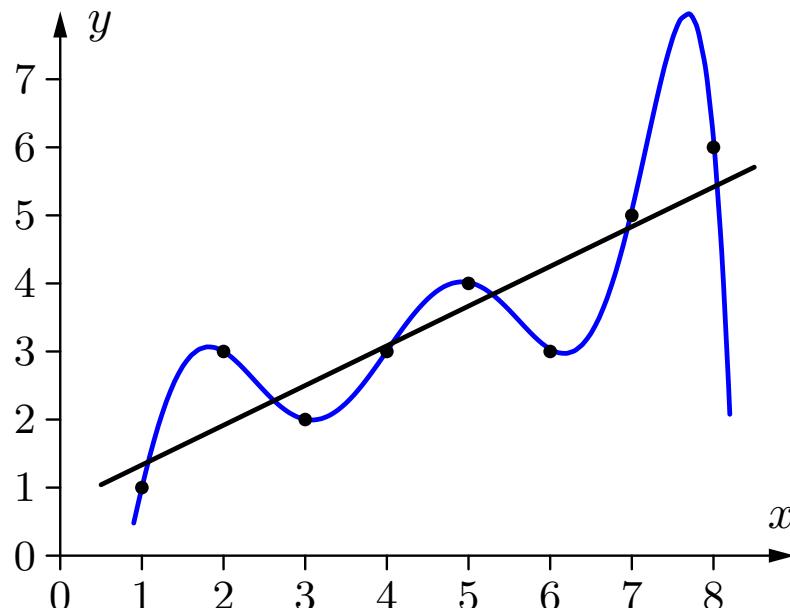
The reason is that the validation data set is likely distorted in a different fashion than the training data, since the errors and deviations are random.

- Minimizing the error on the validation data by properly choosing the number of hidden neurons prevents both under- and overfitting.

Number of Hidden Neurons: Avoid Overfitting

- Objective: select the model that best fits the data, **taking the model complexity into account.**

The more complex the model, the better it usually fits the data.



black line:
regression line
(2 free parameters)

blue curve:
7th order regression polynomial
(8 free parameters)

- The blue curve fits the data points perfectly, **but it is not a good model.**

Number of Hidden Neurons: Cross Validation

- The described method of iteratively splitting the data into training and validation data may be referred to as **cross validation**.
- However, this term is more often used for the following specific procedure:
 - The given data set is split into n parts or subsets (also called folds) of about equal size (so-called **n -fold cross validation**).
 - If the output is nominal (also sometimes called symbolic or categorical), this split is done in such a way that the relative frequency of the output values in the subsets/folds represent as well as possible the relative frequencies of these values in the data set as a whole. This is also called **stratification** (derived from the Latin *stratum*: layer, level, tier).
 - Out of these n data subsets (or folds) n pairs of training and validation data set are formed by using one fold as a validation data set while the remaining $n - 1$ folds are combined into a training data set.

Number of Hidden Neurons: Cross Validation

- The advantage of the cross validation method is that one random split of the data yields n different pairs of training and validation data set.
- An obvious disadvantage is that (except for $n = 2$) the size of the training and the test data set are considerably different, which makes the results on the validation data statistically less reliable.
- It is therefore only recommended for sufficiently large data sets or sufficiently small n , so that the validation data sets are of sufficient size.
- Repeating the split (either with $n = 2$ or greater n) has the advantage that one obtains many more training and validation data sets, leading to more reliable statistics (here: for the number of hidden neurons).
- The described approaches fall into the category of **resampling methods**.
- Other well-known statistical resampling methods are **bootstrap**, **jackknife**, **subsampling** and **permutation test**.

Avoiding Overfitting: Alternatives

- An alternative way to prevent overfitting is the following approach:
 - During training the performance of the multi-layer perceptron is evaluated after each epoch (or every few epochs) on a **validation data set**.
 - While the error on the training data set should always decrease with each epoch, the error on the validation data set should, after decreasing initially as well, increase again as soon as overfitting sets in.
 - At this moment training is terminated and either the current state or (if available) the state of the multi-layer perceptron, for which the error on the validation data reached a minimum, is reported as the training result.
- Furthermore a stopping criterion may be derived from the shape of the error curve on the training data over the training epochs, or the network is trained only for a fixed, relatively small number of epochs (also known as **early stopping**).
- Disadvantage: these methods stop the training of a complex network early enough, rather than adjust the complexity of the network to the “correct” level.

Multi-layer Perceptrons

- **Biological Background**
- **Threshold Logic Units**
 - Definition, Geometric Interpretation, Linear Separability
 - Training Threshold Logic Units, Limitations
 - Networks of Threshold Logic Units
- **Multilayer Perceptrons**
 - Definition of Multilayer Perceptrons
 - Why Non-linear Activation Functions?
 - Function Approximation
 - Training with Gradient Descent
 - Training Examples and Variants
- **Core Idea: Mimic Biological Neural Networks**

Ensemble Methods

Ensemble Methods

- **Fundamental Ideas**
 - Combine Multiple Classifiers/Predictors
 - Why Do Ensemble Methods Work?
- **Some Popular Ensemble Methods**
 - Bayesian Voting
 - Bagging
 - Random Subspace Selection
 - Injecting Randomness
 - Boosting (especially AdaBoost)
 - Mixture of Experts
 - Stacking
- **Summary**

Ensemble Methods: Fundamental Ideas

- It is well known from psychological studies of problem solving activities (but also highly plausible in itself) that a committee of (human) experts with different, but complementary skills usually produces better solutions than any individual.
- Ensemble methods combine several predictors (classifiers or numeric predictors) to improve the prediction quality over the performance of the individual predictors.
- Instead of using a single model to predict the target value, we employ an ensemble of predictors and combine their predictions in order to obtain a joint prediction.
- The core ingredients of ensemble methods are a **procedure to construct different predictors** and a **rule how to combine their results**.

Depending on the choices that are made for these two ingredients, a large variety of different ensemble methods has been suggested.

- While usually yielding higher accuracy than individual models, the fact that sometimes very large ensembles are employed makes the ensemble prediction mechanism difficult to interpret (even if the elements are simple).

Ensemble Methods: Fundamental Ideas

- A necessary and sufficient condition for an ensemble to out-perform the individuals is that the predictors are *reasonably accurate* and *diverse*.
- Technically, a predictor is already called **(reasonably) accurate** if it predicts the correct target value for a new input object better than random guessing.
Hence this is a pretty weak requirement that is easy to meet in practice.
- Two predictors are called **diverse** if they do not make the same mistakes.
That this requirement is essential is obvious: if the predictors always made the same mistakes, no improvement could possibly result from combining them.
- Consider the extreme case that the predictors in the ensemble are all identical: the combined prediction is necessarily the same as that of any individual predictor —regardless of how the individual predictions are combined.
- However, if the errors made by the individual predictors are uncorrelated, their combination will reduce these errors.

Ensemble Methods: Fundamental Ideas

- If we combine classifiers making independent mistakes by majority voting, the ensemble yields a wrong result only if more than half of the classifiers misclassify an input object, thus improving over the individuals.
- For instance, for 5 independent classifiers for a two-class problem, each having an error probability of 0.3, the probability that 3 or more yield a wrong result is

$$\sum_{i=3}^5 \binom{5}{i} \cdot 0.3^i \cdot 0.7^{5-i} = 0.08748.$$

- Note, however, that this holds only for the ideal case that the classifiers are fully independent, which is usually not the case in practice.
- Fortunately, though, improvements are also achieved if the dependence is sufficiently weak, although the gains are naturally smaller.
- Note also that even in the ideal case no gains result (but rather a degradation) if the error probability of an individual classifier exceeds 0.5, which substantiates the requirement that the individual predictors should be accurate.

Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are basically three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.

- **Statistical Reason:**

The statistical reason is that in practice any learning method has to work on a finite data set and thus may not be able to identify the correct predictor, even if this predictor lies within the set of models that the learning method can, in principle, return as a result. Rather it is to be expected that there are several predictors that yield similar accuracy.

- Since there is thus no sufficiently clear evidence which model is the correct or best one, there is a certain risk that the learning method selects a suboptimal model.
- By removing the requirement to produce a single model, it becomes possible to “average” over many or even all of the good models.
- This reduces the risk of excluding the best predictor and the influence of actually bad models.

Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are basically three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.
- **Computational Reason:**

The computational reason refers to the fact that learning algorithms usually cannot traverse the complete model space, but must use certain heuristics (greedy, hill climbing, gradient descent etc.) in order to find a model.
- Since these heuristics may yield suboptimal models (for example, local minima of the error function), a suboptimal model may be chosen.
- However, if several models constructed with heuristics are combined in an ensemble, the result may be a better approximation of the true dependence between the inputs and the target variable.

Why Do Ensemble Methods Work?

- According to [Dietterich 2000] there are basically three reasons why ensemble methods work: *statistical*, *computational*, and *representational*.

- **Representational Reason:**

The representational reason is that for basically all learning methods, even the most flexible ones, the class of models that can be learned is limited and thus it may be that the true model cannot be represented accurately.

- By combining several models in a predictor ensemble, the model space can be enriched, that is, the ensemble may be able to represent a dependence between the inputs and the target variable that cannot be expressed by any of the individual models the learning method is able to produce.
- From a representational point of view, ensemble methods make it possible to reduce the bias of a learning algorithm by extending its model space, while the statistical and computational reasons indicate that they can also reduce the variance.
- In this sense, ensemble methods are able to sever the usual link between bias and variance.

Ensemble Methods: Bayesian Voting

- In pure **Bayesian voting** the set of all possible models in a user-defined hypothesis space is enumerated to form the ensemble.
- The predictions of the individual models are combined weighted with the posterior probability of the model given the training data.
That is, models that are unlikely to be correct given the data have a low influence on the ensemble prediction, models that are likely have a high influence.

- The likelihood of the model given the data can often be computed as

$$P(M | D) \propto P(D | M)P(M),$$

where M is the model, D the data, $P(M)$ the prior probability of the model (often assumed to be the same for all models), and $P(D | M)$ the data likelihood given the model.

- Theoretically, Bayesian voting is the optimal combination method, because all possible models are considered and their relative influence reflects their likelihood given the data.

Ensemble Methods: Bayesian Voting

- Theoretically, Bayesian voting is the optimal combination method, because all possible models are considered and their relative influence reflects their likelihood given the data.
- In practice, however, it suffers from several drawbacks:
 - It is rarely possible to actually enumerate all models in the hypothesis space that is (implicitly) defined by a learning method.

For example, even if we restrict the tree size, it is usually infeasible to enumerate all decision trees that could be constructed for a given classification problem.
 - In order to overcome this problem, model sampling methods are employed, which ideally should select a model with a probability that corresponds to their likelihood given the data.
 - However, most such methods are biased and thus usually do not yield a representative sample of the total set of models, sometimes seriously degrading the ensemble performance.

Ensemble Methods: Bagging

- The method of **bagging** (bootstrap aggregating) predictors can be applied with basically any learning algorithm.
- The basic idea is to select a single learning algorithm (most studied in this respect are decision tree inducers) and to learn several models by providing it each time with a different random sample of the training data.
- The sampling is carried out with replacement (bootstrapping) with the sample size commonly chosen as $n(1 - 1/e) \approx 0.632n$, (where n is the number of training samples).
- Especially if the learning algorithm is unstable (like decision tree inducers, where a small change of the data can lead to a considerably different decision tree), the resulting models will usually be fairly diverse, thus satisfying one of the conditions needed for ensemble methods to work.
- The predictions of the individual models are then combined by simple majority voting or by averaging them (with the same weight for each model).

Ensemble Methods: Bagging

- Bagging effectively yields predictions from an “average model”, even though this model does not exist in simple form — it may not even lie in the hypothesis space of the learning algorithm.
- It has been shown that bagging reduces the risk of overfitting the training data (because each subsample has different special properties) and thus produces very robust predictions.
- Experiments show that bagging yields very good results especially for noisy data sets, where the sampling seems to be highly effective to avoid any adaptation to the noise data points.
- A closely related alternative to bagging are **cross-validated committees**.
- Instead of resampling the training data with replacement (bootstrapping) to generate the predictors of an ensemble, the predictors learned during a cross validation run are combined with equal weights in a majority vote or by averaging.

Ensemble Methods: Random Subspace Selection

- While bagging obtains a set of diverse predictors by randomly varying the training data, **random subspace selection** employs a random selection of the features for this purpose.
- That is, all data points are used in each training run, but the features the model construction algorithm can use are randomly selected.
- With a learning algorithm like a decision tree inducer (for which random subspace selection was first proposed), the available features may even be varied each time a split has to be chosen, so that the whole decision tree can potentially use all features.
- Combining random subspace selection with bagging is a highly effective and strongly recommended method if accuracy is the main goal.
- Applied to decision trees this method has been named **random forests**, which is known to be one of the most accurate classification methods to date.
- However, the often huge number of trees destroys the advantage that decision trees are easy to interpret and can be checked for plausibility.

Ensemble Methods: Injecting Randomness

- Both bagging and random subspace selection employ random processes in order to obtain diverse predictors.
- This approach can of course be generalized to the principle of **injecting randomness** into the learning process.
 - Bagging: select training data set randomly
 - Random Subspace Selection: select feature set randomly
- For example, such an approach is very natural and straightforward for **artificial neural networks**:
different initialization of the connection weights often yield different learning results (different local optima), which may then be used as the members of an ensemble.
- Alternatively, the network structure can be modified, for example, by randomly deleting a certain fraction of the connections between two consecutive layers.

Ensemble Methods: Boosting

- **Boosting** constructs predictors progressively, with the prediction results of the model learned last influencing the construction of the next model.
- Like bagging, boosting varies the training data. However, instead of drawing random samples, boosting always works on the complete training data set, and maintains and manipulates a data point weight for each training example.
- For low noise data, boosting clearly outperforms bagging and random subspace selection in experimental studies.
- However, if the training data contains noise, the performance of boosting can degrade quickly, because it tends to focus on the noise data points (which are necessarily difficult to classify and thus receive high weights after fairly few steps).
- As a consequence, boosting overfits the data.
For noisy data bagging and random subspace selection yield much better results.
- Boosting is usually described for classification problems with two classes, which are assumed to be coded by 1 and -1 .

Ensemble Methods: AdaBoost

- The best-known boosting approach is **AdaBoost** and works as follows:
- Initially, all data point weights are equal and therefore set to $w_i = 1/n$, $i = 1, \dots, n$, where n is the size of the data set.
- After a predictor M_t has been constructed in step t using the current weights $w_{i,t}$, $i = 1, \dots, n$, it is applied to the training data and

$$e_t = \frac{\sum_{i=1}^n w_{i,t} \cdot y_i \cdot M_t(\vec{x}_i)}{\sum_{i=1}^n w_{i,t}} \quad \text{and} \quad \alpha_t = \frac{1}{2} \ln \left(\frac{1 - e_t}{1 + e_t} \right)$$

are computed, where \vec{x}_i is the input vector, $y_i \in \{-1, 1\}$ the class of the i -th training example, and $M_t(\vec{x}_i)$ the prediction of the model for the input \vec{x}_i .

- The data point weights are then updated according to

$$w_{i,t+1} = c \cdot w_{i,t} \cdot \exp(-\alpha_t y_i M_t(\vec{x}_i)),$$

where c is a normalization constant chosen in such a way that $\sum_{i=1}^n w_{i,t+1} = 1$.

Ensemble Methods: AdaBoost

- The procedure of learning a predictor and updating the data point weights is repeated a user-specified number of times t_{\max} .
- The constructed ensemble classifies new data points by majority voting, with each model M_t weighted with α_t .
- That is, the joint prediction is

$$M_{\text{joint}}(\vec{x}_i) = \text{sign} \left(\sum_{t=1}^{t_{\max}} \alpha_t M_t(\vec{x}_i) \right).$$

- Since there is no convergence guarantee and the performance of the ensemble classifier can even degrade after a certain number of steps, the inflection point of the error curve over t is often chosen as the ensemble size.
- Reminder: if the training data contains noise, the performance of boosting can degrade quickly, because it tends to focus on the noise data points (which are necessarily difficult to classify and thus receive high weights after fairly few steps). As a consequence, boosting overfits the data.

Ensemble Methods: Mixture of Experts

- In the approach referred to as **mixture of experts** the individual predictors to combine are assumed as already given, for example, selected by a user.
- They may be, for instance, results of different learning algorithms, like a decision tree, neural networks with different network structure, a support vector machine, etc. — whatever the user sees as promising to solve the application task.
- Alternatively, they may be the set of models obtained from any of the ensemble methods described so far.
- The focus is then placed on finding an optimal rule to combine the predictions of the individual models.
- For classification tasks, for example, the input to this combination rule are the probability distributions over the classes that the individual classifiers yield.
- Note that this requires more than simple (weighted) majority voting, which only asks each classifier for its best guess of the class of a new input object: each classifier must assign a probability to each class.

Ensemble Methods: Mixture of Experts

- The most common rules to combine such class probabilities are
 - the so-called **sum rule**, which simply averages, for each class, the probabilities provided by the individual classifiers and
 - the so-called **product rule**, which assumes conditional independence of the classifiers given the class and therefore multiplies, for each class, the probabilities provided by the different classifiers.
- In both cases the class with the largest sum or product is chosen as the prediction of the ensemble.
- Experiments show that the sum rule is usually preferable, likely because due to the product a class that is seen as (very) unlikely by a single classifier has little chance of being predicted, even if several other classifiers assign a high probability to it.
- Both the sum rule and the product rule can be seen as special cases of a general family of combination rules that are known as f -means. Other such rules include Dempster–Shafer combination and rank based rules.

Ensemble Methods: Stacking

- Like a mixture of experts, **stacking** takes the set of predictors as already given and focuses on combining their individual predictions.
- The core idea is to view the outputs of the predictors as new features and to use a learning algorithm to find a model that combines them optimally.
- Technically, a new data table is set up with one row for each training example, the columns of which contain the predictions of the different (level-1) models for training example.
In addition, a final column states the true classes.
- With this new training data set a (level-2) model is learned, the output of which is the prediction of the ensemble.
- Note that the level-2 model may be of the same or of a different type than the level-1 models.
- For example, the output of several regression trees (e.g. a random forest) may be combined with a linear regression, or with a neural network.

Ensemble Methods: Summary

- **Fundamental Ideas**
 - Combine Multiple Classifiers/Predictors
 - Why Do Ensemble Methods Work?
- **Some Popular Ensemble Methods**
 - Bayesian Voting
 - Bagging
 - Random Subspace Selection
 - Injecting Randomness
 - Boosting (especially AdaBoost)
 - Mixture of Experts
 - Stacking
- **Ingredients: Predictor Construction + Combination Rule**

Clustering

Clustering

- **General Idea of Clustering**
 - Similarity and distance measures
- **Prototype-based Clustering**
 - Classical c -means (or k -means) clustering
 - Learning vector quantization (“online” c -means clustering)
 - Fuzzy c -means clustering
 - Expectation maximization for Gaussian mixtures
- **Hierarchical Agglomerative Clustering**
 - Merging clusters: Dendograms
 - Measuring the distance of clusters
 - Choosing the clusters
- **Summary**

General Idea of Clustering

- Goal: Arrange the given data tuples into **classes** or **clusters**.
- Data tuples assigned to the **same cluster** should be as **similar** as possible.
- Data tuples assigned to **different clusters** should be as **dissimilar** as possible.
- Similarity is most often measured with the help of a distance function.
(The smaller the distance, the more similar the data tuples.)
- Often: restriction to data points in \mathbb{R}^m (although this is not mandatory).

$d : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}_0^+$ is a **distance function** if it satisfies $\forall \vec{x}, \vec{y}, \vec{z} \in \mathbb{R}^m :$

$$(i) \quad d(\vec{x}, \vec{y}) = 0 \iff \vec{x} = \vec{y},$$

$$(ii) \quad d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x}) \quad (\text{symmetry}),$$

$$(iii) \quad d(\vec{x}, \vec{z}) \leq d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z}) \quad (\text{triangle inequality}).$$

Distance Functions

Illustration of distance functions: Minkowski Family

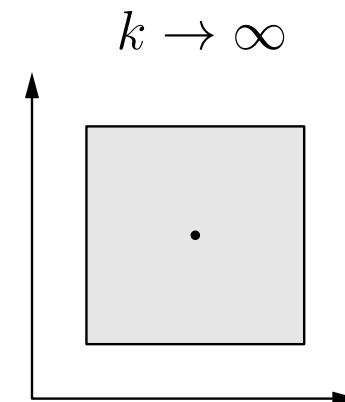
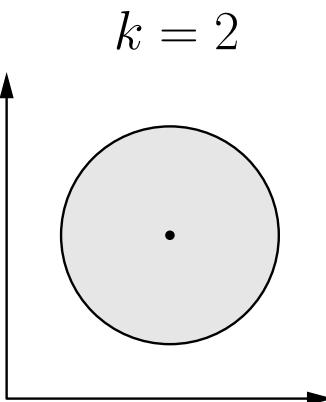
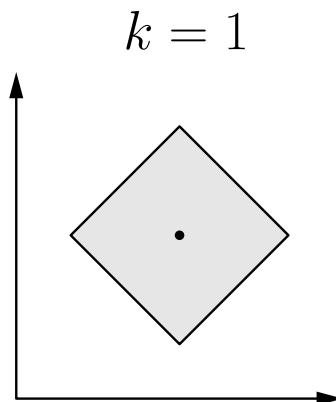
$$d_k(\vec{x}, \vec{y}) = \sqrt[k]{\sum_{i=1}^n |x_i - y_i|^k} = \left(\sum_{i=1}^n |x_i - y_i|^k \right)^{\frac{1}{k}}$$

Well-known special cases from this family are:

$k = 1$: Manhattan or city block distance,

$k = 2$: Euclidean distance (the only isotropic distance),

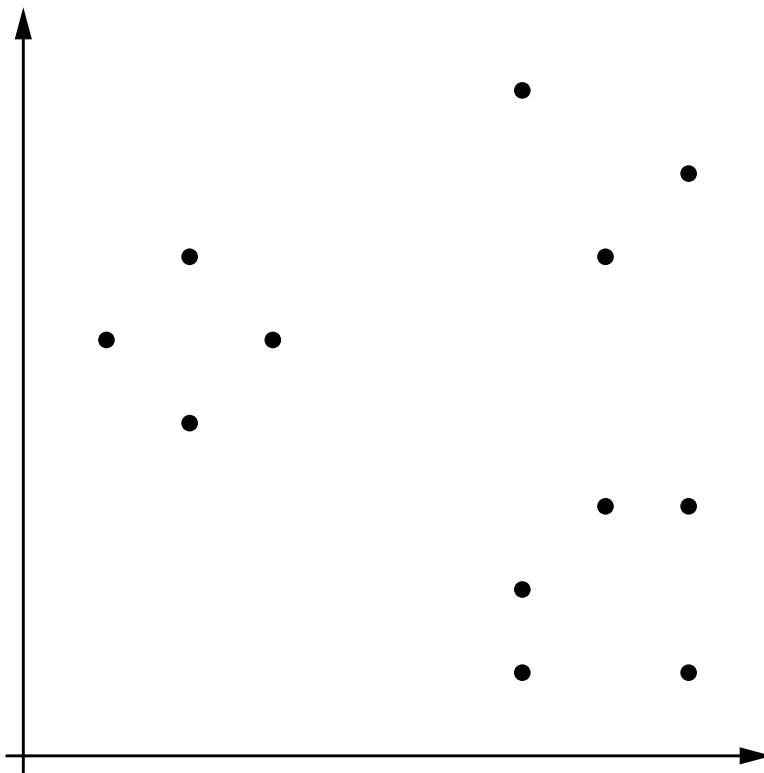
$k \rightarrow \infty$: maximum distance, i.e. $d_\infty(\vec{x}, \vec{y}) = \max_{i=1}^n |x_i - y_i|$.



c -Means Clustering (a.k.a. k -Means Clustering)

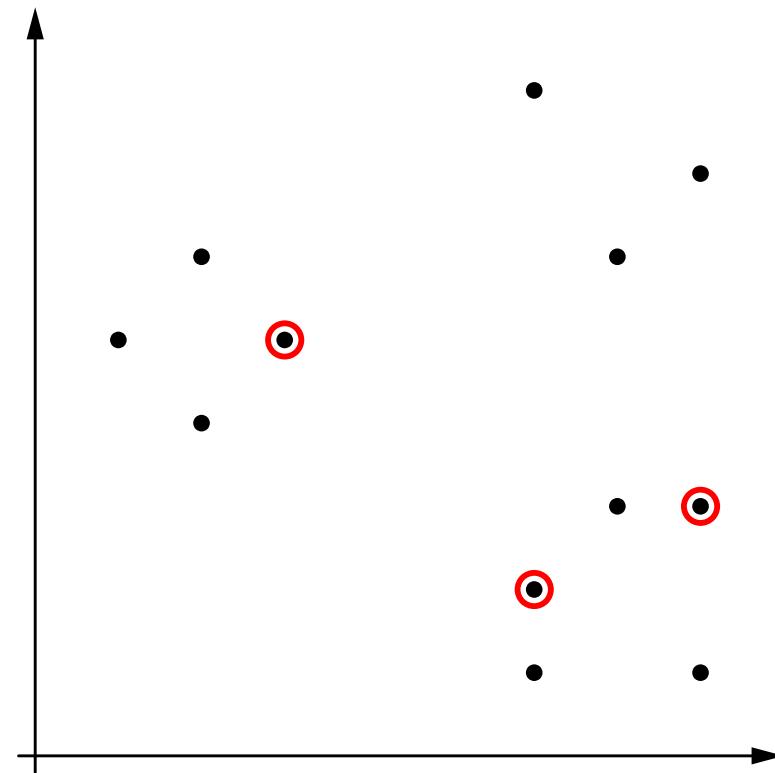
- Choose a number c (or k) of clusters to be found (user input).
- Initialize the cluster centers randomly
(for instance, by randomly selecting c data points — more details later).
- **Data point assignment:**
Assign each data point to the cluster center that is closest to it
(i.e. closer than any other cluster center).
- **Cluster center update:**
Compute new cluster centers as the mean vectors of the assigned data points.
(Intuitively: center of gravity if each data point has unit weight.)
- Repeat these two steps (data point assignment and cluster center update)
until the clusters centers do not change anymore (convergence).
- It can be shown that this scheme must converge,
i.e., the update of the cluster centers cannot go on forever.

c -Means Clustering: Example



Data set to cluster.

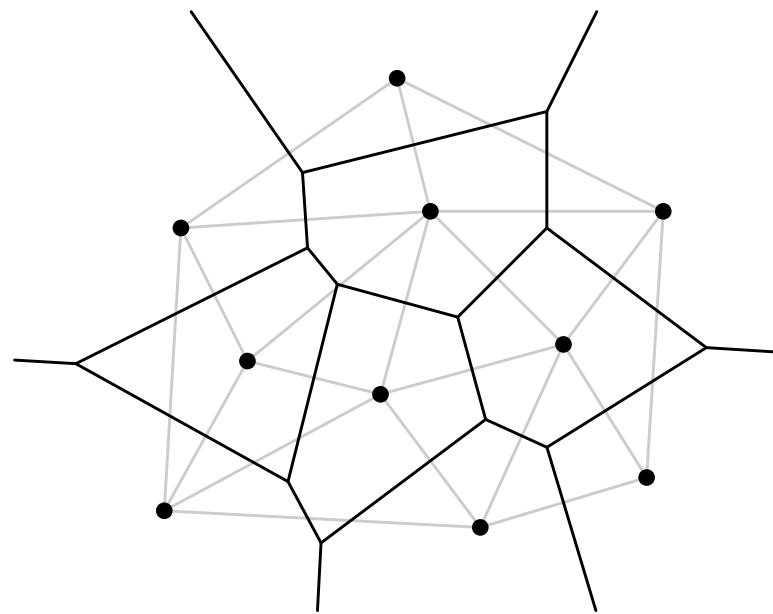
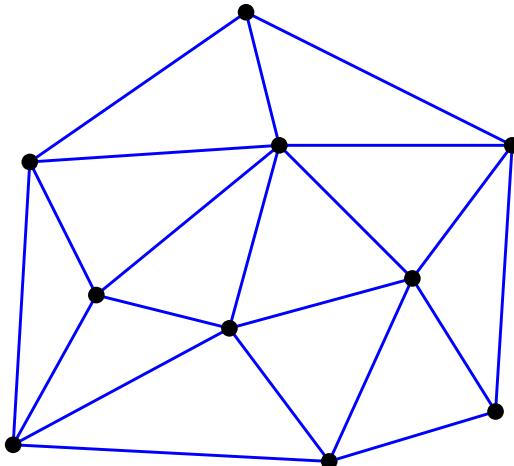
Choose $c = 3$ clusters.
(From visual inspection, can be difficult to determine in general.)



Initial position of cluster centers.

Randomly selected data points.
(Alternative methods include e.g. latin hypercube sampling)

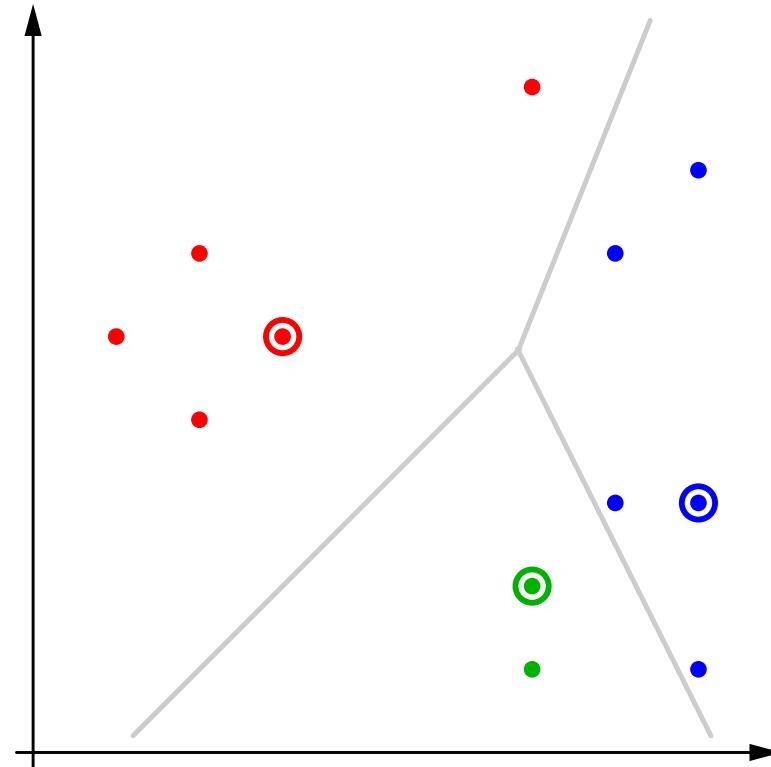
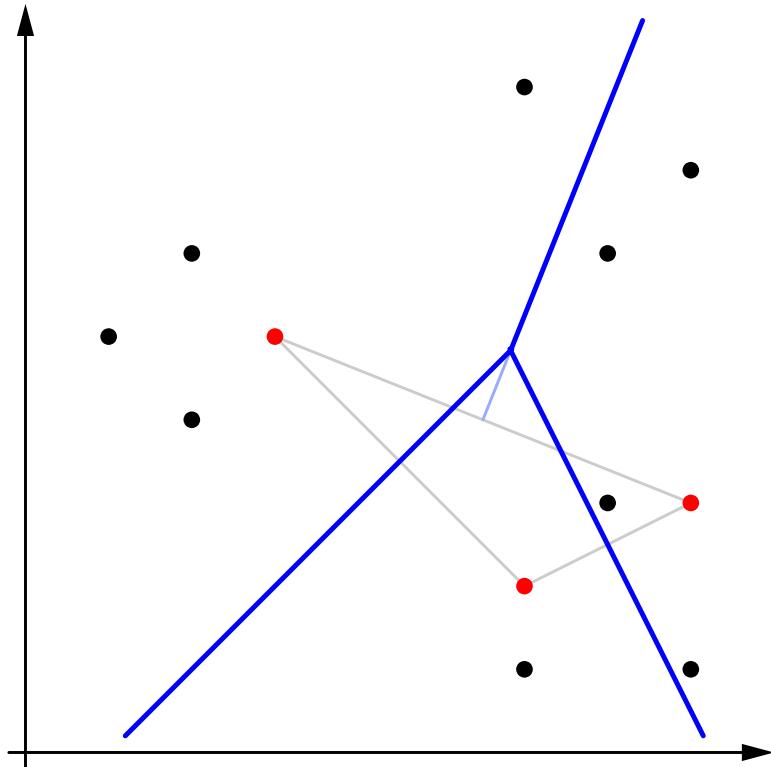
Delaunay Triangulations and Voronoi Diagrams



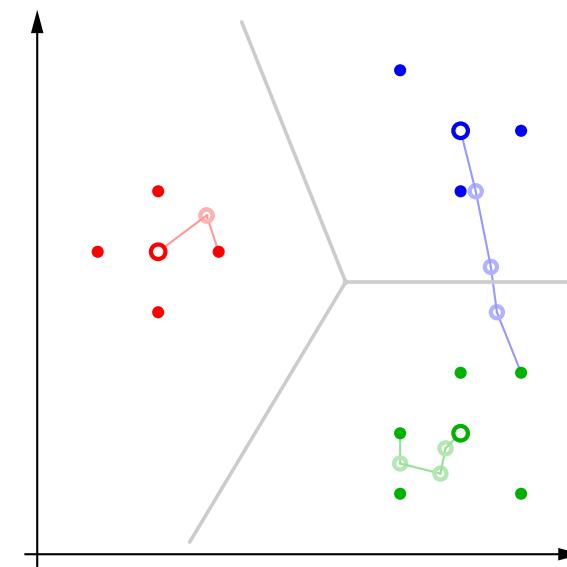
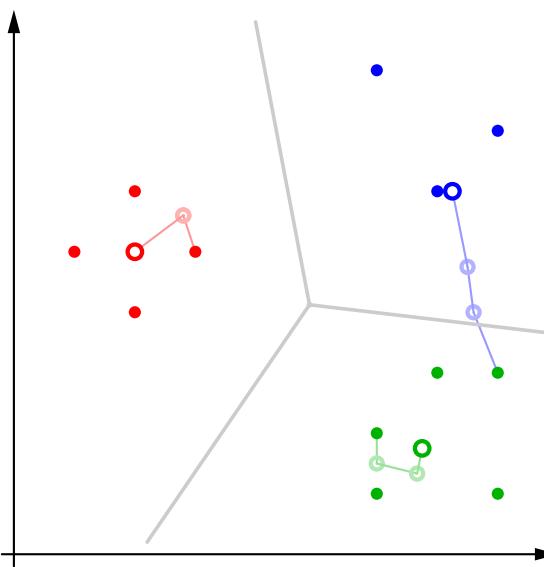
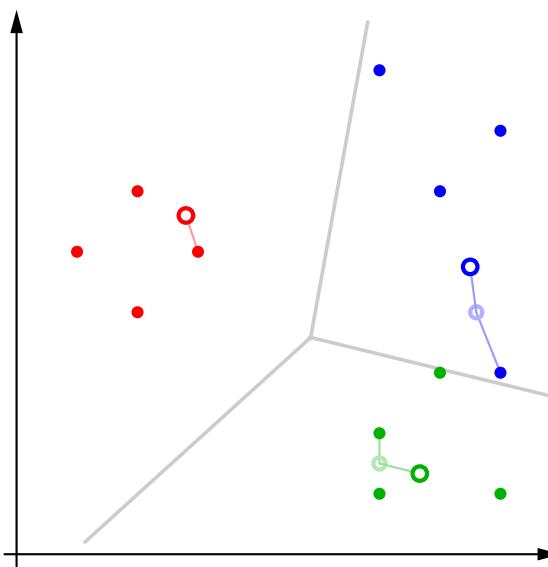
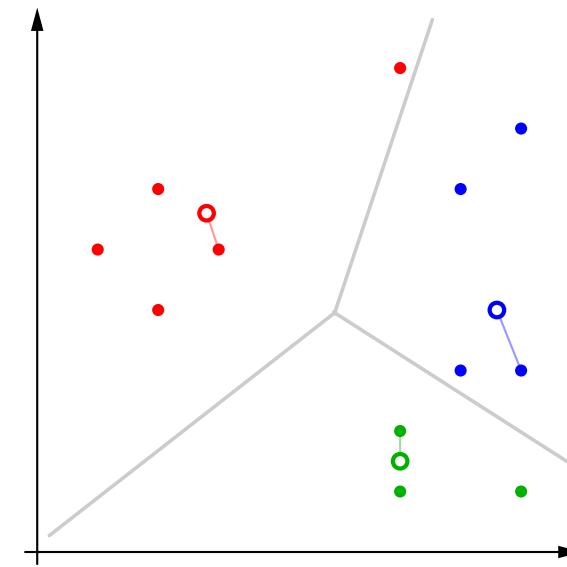
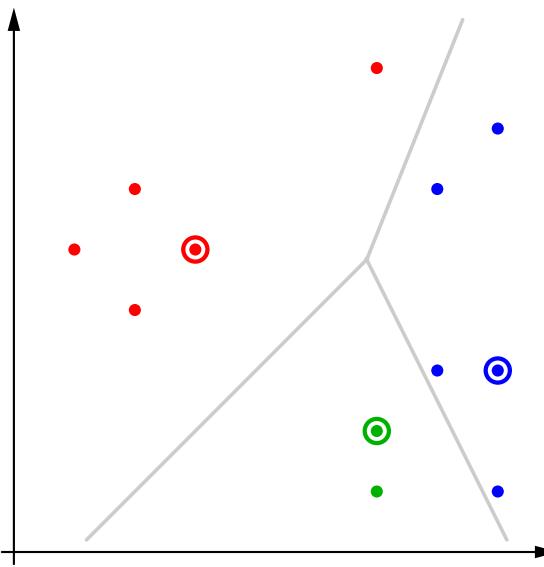
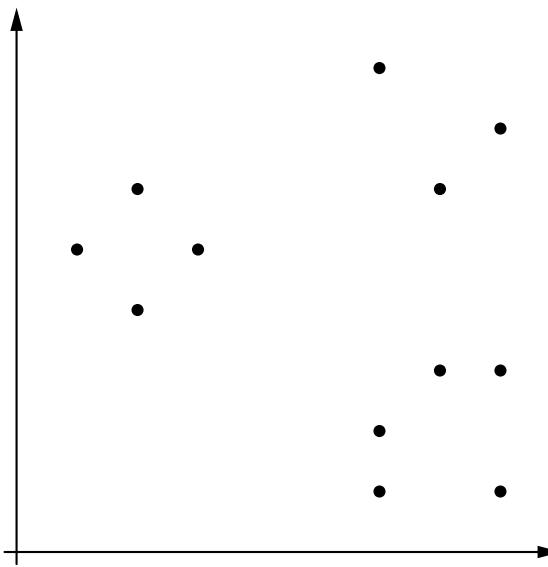
- Dots represent cluster centers (quantization vectors).
- Left: **Delaunay Triangulation**
(The circle through the corners of a triangle does not contain another point.)
- Right: **Voronoi Diagram**
(Midperpendiculars of the Delaunay triangulation: boundaries of the regions of points that are closest to the enclosed cluster center (Voronoi cells)).

Delaunay Triangulations and Voronoi Diagrams

- **Delaunay Triangulation:** simple triangle (shown in grey on the left)
- **Voronoi Diagram:** midperpendiculars of the triangle's edges (shown in blue on the left, in grey on the right)



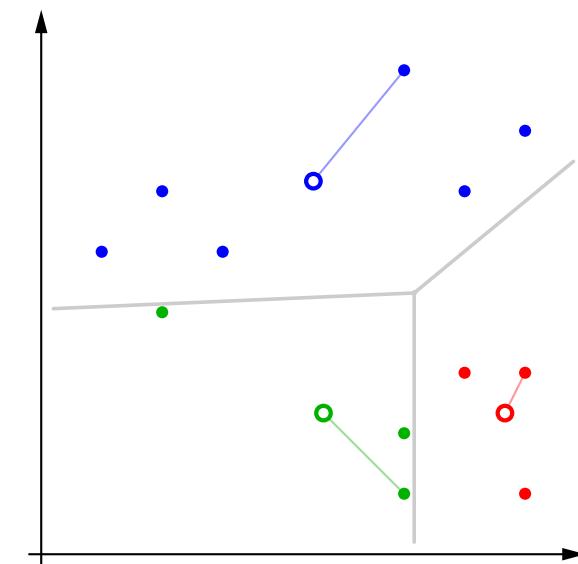
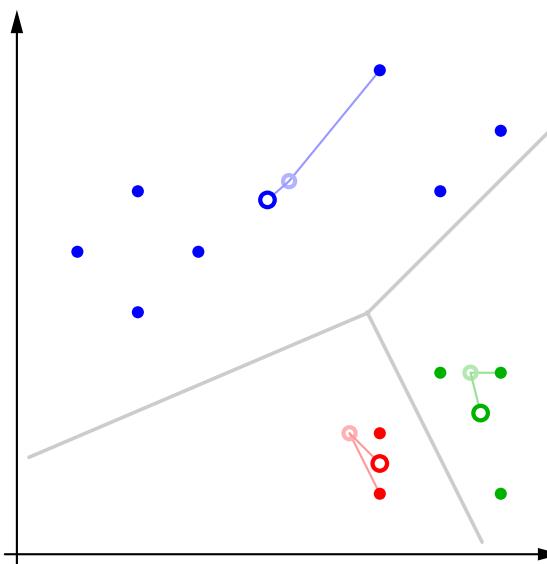
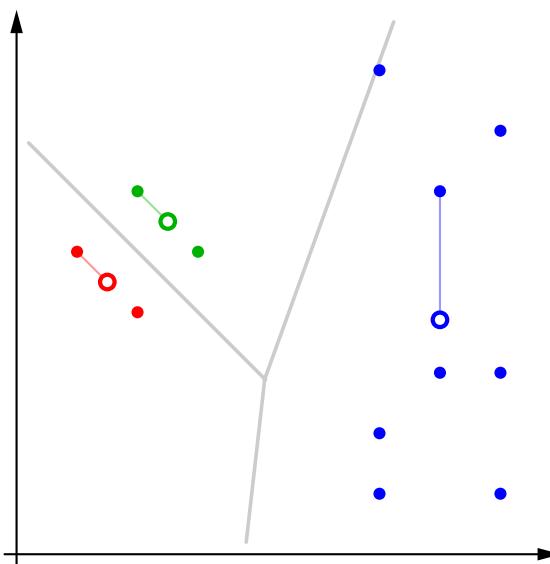
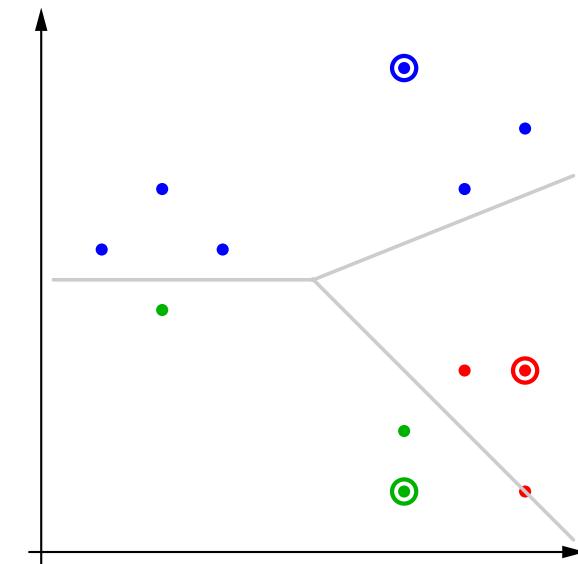
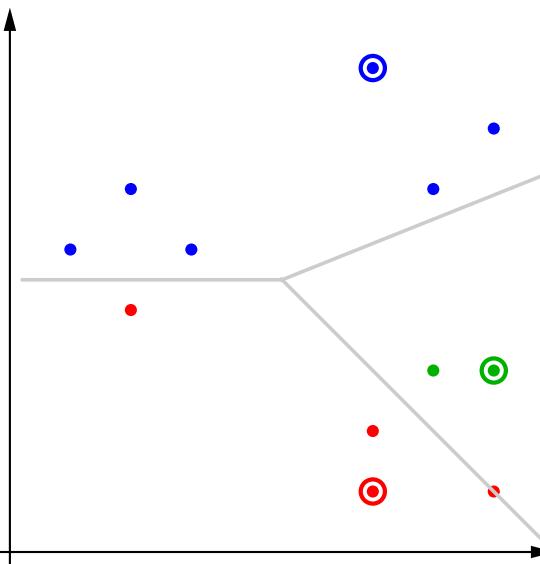
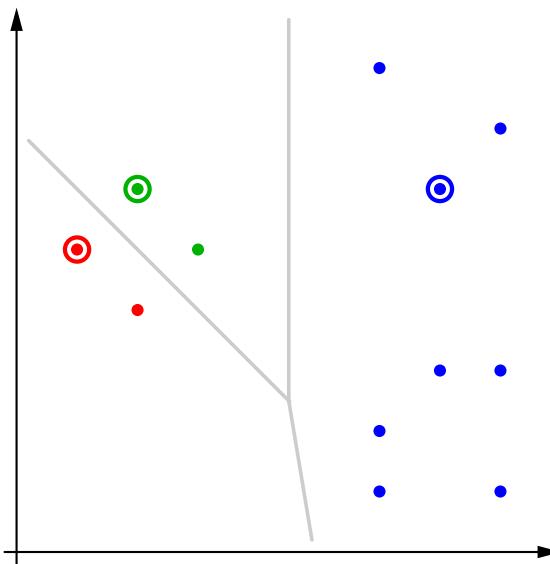
c -Means Clustering: Example



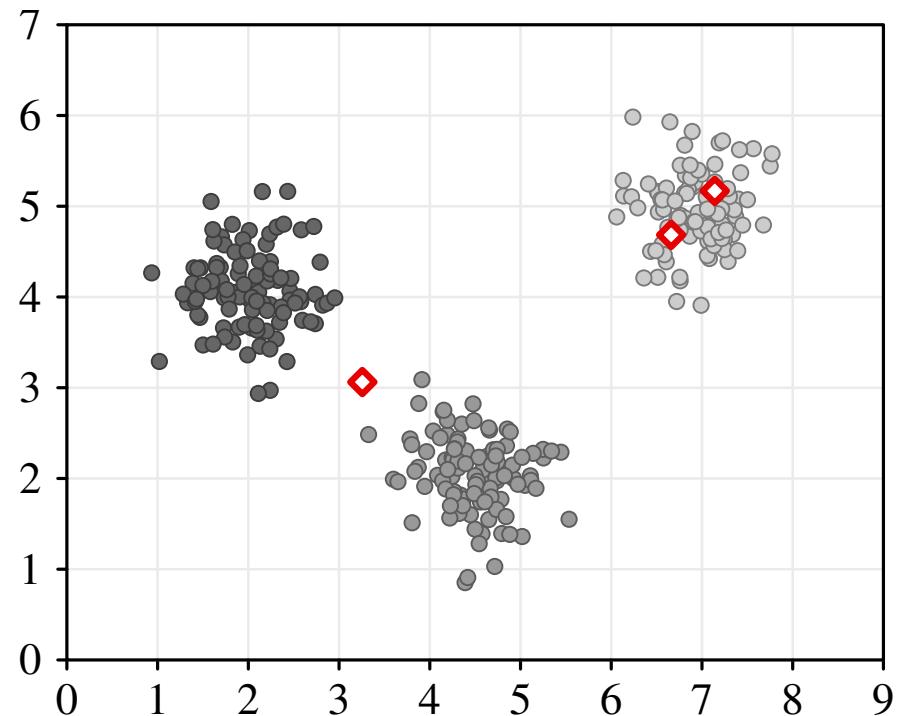
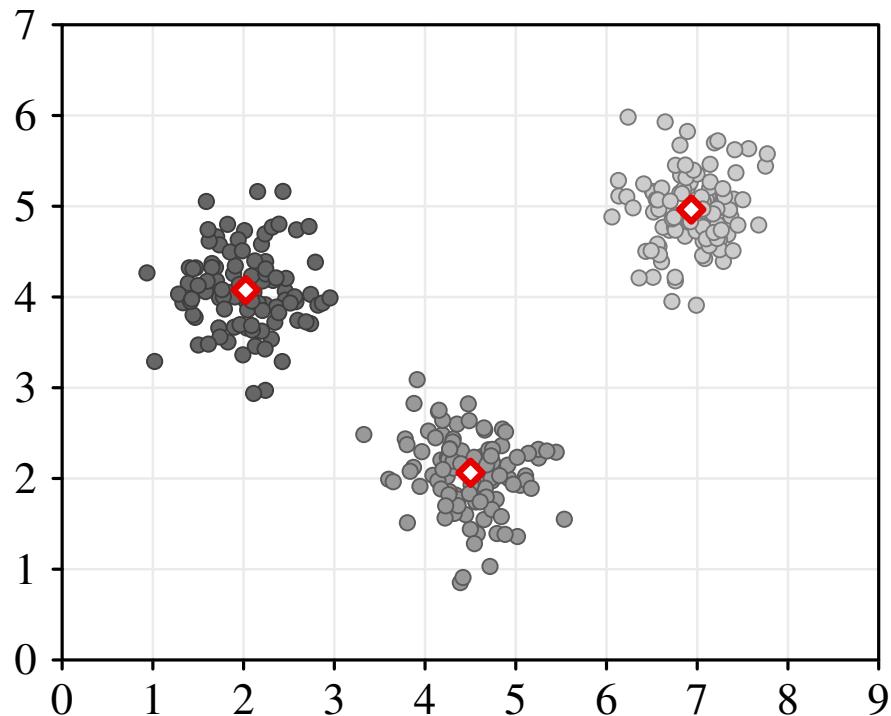
c -Means Clustering: Local Minima

- Clustering is successful in this example:
The clusters found are those that would have been formed intuitively.
- Convergence is achieved after only 5 steps.
(This is typical: convergence is usually very fast.)
- However: The clustering result is fairly **sensitive to the initial positions** of the cluster centers (see examples on next slides).
- With a bad initialization clustering may fail
(the alternating update process gets stuck in a local minimum).
- Fuzzy c -means clustering and the estimation of a mixture of Gaussians are much more robust (to be discussed later).
- Research issue: Can we determine the number of clusters automatically?
(Some approaches exist, resampling is most successful.)

c -Means Clustering: Local Minima



c -Means Clustering: Local Minima



A simple data set with three clusters and 300 data points (100 per cluster).

Result of a successful c -means clustering (left) and a local optimum (right).

Red diamonds mark cluster centers.

In an unsuccessful clustering, actual clusters are split or merged.

c -Means Clustering: Formal Description

- We are given a data set $\mathbf{X} = \{\vec{x}_1, \dots, \vec{x}_n\}$ with n data points.
Each data point is an m -dimensional real-valued vector,
that is, $\forall j; 1 \leq j \leq n : \vec{x}_j = (x_{j1}, \dots, x_{jm}) \in \mathbb{R}^m$.
- These data points are to be grouped into c clusters,
each of which is described by a prototype \mathbf{c}_i , $i = 1, \dots, c$.
The set of all prototypes is denoted by $\mathbf{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_c\}$.
- We (first) confine ourselves here to cluster prototypes that consist merely
of a cluster center, that is, $\forall i; 1 \leq i \leq c : \mathbf{c}_i = \vec{c}_i = (c_{i1}, \dots, c_{im}) \in \mathbb{R}^m$.
- The assignment of data points to cluster centers is encoded as a $c \times n$ matrix
 $\mathbf{U} = (u_{ij})_{1 \leq i \leq c; 1 \leq j \leq n}$, which is often called the *partition matrix*.
- In the crisp case, a matrix element $u_{ij} \in \{0, 1\}$ states whether data point \vec{x}_j
belongs to cluster \vec{c}_i or not.
In the fuzzy case (discussed later), $u_{ij} \in [0, 1]$ states the degree to which \vec{x}_j
belongs to \vec{c}_i (degree of membership).

c -Means Clustering: Formal Description

- We confine ourselves (first) to the (squared) Euclidean distance as the measure for the distance between a data point \vec{x}_j and a cluster center \vec{c}_i , that is,

$$d_{ij}^2 = d^2(\vec{c}_i, \vec{x}_j) = (\vec{x}_j - \vec{c}_i)^\top (\vec{x}_j - \vec{c}_i) = \sum_{k=1}^m (x_{jk} - c_{ik})^2.$$

- The objective of (hard) c -means clustering is to minimize the objective function

$$J(\mathbf{X}, \mathbf{C}, \mathbf{U}) = \sum_{i=1}^c \sum_{j=1}^n u_{ij} d_{ij}^2$$

under the constraints

- $\forall j; 1 \leq j \leq n : \quad \forall i; 1 \leq i \leq c : \quad u_{ij} \in \{0, 1\}$ and
- $\forall j; 1 \leq j \leq n : \quad \sum_{i=1}^c u_{ij} = 1.$

⇒ hard/crisp data point assignment:

each data point is assigned to one cluster and one cluster only.

(“Fuzzy” and probabilistic data point assignments are considered later.)

c -Means Clustering: Formal Description

- Since the minimum cannot be found directly using analytical means, an **alternating optimization** scheme is employed.
- At the beginning the cluster centers are initialized (discussed in more detail later).
- Then the two steps of *partition matrix update* (data point assignment) and *cluster center update* are iterated until convergence, that is, until the cluster centers do not change anymore.
- The **partition matrix update** assigns each data point \vec{x}_j to the cluster \mathbf{c}_i , the center \vec{c}_i of which is closest to it:

$$u_{ij} = \begin{cases} 1, & \text{if } i = \operatorname{argmin}_{k=1}^c d_{kj}^2, \\ 0, & \text{otherwise.} \end{cases}$$

- The **cluster center update** recomputes each cluster center as the *mean* of the data points that were assigned to it, that is,

$$\vec{c}_i = \frac{\sum_{j=1}^n u_{ij} \vec{x}_j}{\sum_{j=1}^n u_{ij}}.$$

Cluster Center Initialization

- **Random Data Points** (this was assumed up to now)
 - Choose c data points uniformly at random without replacement.

Advantages:

- Very simple and efficient
(time complexity $O(c)$, no distance computations needed).

Disadvantages:

- Prone to the local optimum problem:
Often leads to fairly bad clustering results.
- May actually be less efficient than other initialization methods,
because it usually requires more update steps until convergence.
- Repeating the initialization (and clustering) several times helps,
but results are usually still inferior compared to other methods.

Cluster Center Initialization

- **Maximin Initialization** [Hathaway, Bezdek, and Huband 2006]

- Choose first cluster center uniformly at random from the data points.
- For the remaining centers always choose the data point that has the maximum minimum distance to the already chosen cluster centers.
- Formally: For $0 < k \leq c$, define $\forall j; 1 \leq j \leq n : d_j(k) = \min_{i=1}^k d_{ij}$.
Then choose $\vec{c}_{k+1} = \vec{x}_\ell$ where $\ell = \operatorname{argmax}_{j=1}^n d_j(k)$.

Advantages:

- Cluster centers are guaranteed to have some distance from each other.
- Guarantees a fairly good coverage of the data points.
(Reduces the tendency to find local optima.)

Disadvantages:

- Tends to choose outliers/extreme data points as cluster centers.

Cluster Center Initialization

- **Maximin Initialization**

- A naive implementation (following the above formal description) needs $O(nc)$ distance computations (time complexity $O(ncm)$).
- An improved computation relies on the following simple insight:

$$\forall r; 1 \leq r \leq k : d_j(k) \leq d_{rj} \quad \text{and} \quad \forall I \subseteq \{1, \dots, k\} : d_j(k) \leq \min_{r \in I} d_{rj}.$$

- For $t \in \{1, \dots, n\}$, define $d_k(t) = \max_{j=1}^t d_j(k)$.
If $d_k(t) > d_{t+1}(s)$ for some $s \leq k$,
then \vec{x}_{t+1} will certainly not be chosen as the next cluster center.
- Implementation: for each data point \vec{x}_j , note s_j and $d_j(s_j)$.
- Traverse the data points and in each step update t and $d_k(t)$.
- As long as $s_{t+1} < k$ and $d_{t+1}(s_{t+1}) > d_k(t)$ (possible maximin point),
set $d_{t+1}(s_{t+1} + 1) = \min\{d_{t+1}(s_{t+1}), d_{(s_{t+1}+1)j}\}$ and increment s_{t+1} .
Finally compute $d_k(t + 1)$ accordingly.

Cluster Center Initialization

- **Maximin Initialization:** Python code (using NumPy)

```
ctrss[0] = data[npr.randint(n)] # choose first cluster center randomly
dsts = np.array([dist(ctrss[0], x) for x in data])
cids = np.zeros(n, dtype=int)    # init. highest used cluster indices
for i in range(1,c):
    dmax = m = 0                # init. max. distance and corresp. index
    for j in range(n):           # traverse the data points
        if dsts[j] <= dmax:     # if less than current maximum,
            continue             # data point will not be selected
        while cids[j] < i-1:     # traverse skipped clusters
            cids[j] += 1          # go to the next cluster
            d = dist(ctrss[cids[j]], data[j])
            if d < dsts[j]:       # if less than known distance,
                dsts[j] = d        # update the minimum distance
            if d < dmax:          # if less than current maximum,
                break               # data point will not be selected
            if dsts[j] > dmax:     # if larger than current maximum
                dmax = dsts[j]      # note new maximum distance and
                m = j                 # corresponding data point index
    dsts[m] = 0.0                  # mark the data point as selected
    ctrss[i] = data[m]              # and add it to the set of centers
```

Cluster Center Initialization

- **kmeans++ (or cmeans++) Initialization** [Arthur and Vassilvitskii 2007]

- Choose first cluster center uniformly at random from the data points.
- For the remaining centers sample data points according to the squared distance a data point has to its closest already chosen cluster center.
- Formally: For $1 \leq k \leq c$, define $\forall j; 1 \leq j \leq n : d_j(k) = \min_{i=1}^k d_{ij}$.

Then sample \vec{c}_{k+1} from the data points according to $P_{k+1}(\vec{x}_j) = \frac{d_j^2(k)}{\sum_{r=1}^n d_r^2(k)}$.

Advantages:

- Selects, with fairly high probability, cluster centers that have some distance from each other and thus centers that cover the data fairly well.
- Less likely to select outliers/extreme data points than maximin.

Disadvantages:

- High costs: needs $O(nc)$ distance computations (time complexity $O(ncm)$).

Cluster Center Initialization

- **kmeans++ (or cmeans++) Initialization:** Python code (using NumPy)

```
ctrss[0] = data[npr.randint(n)] # choose first cluster center randomly
dsts = np.array([sqrdist(ctrss[0], x) for x in data])
                           # compute distances to first center
for i in range(1,c):
    if i > 1:
        dsts = np.minimum(dsts, [sqrdist(ctrss[i], x) for x in data])
    dcum = np.cumsum(dsts)
           # compute cumulated distance sums
    m    = np.searchsorted(dcum, dcum[-1] *npr.random())
    if m >= size:
        m = size-1
    dsts[m] = 0.0
    ctrss[i] = data[m]
                           # mark the data point as selected
                           # and add it to the set of centers
```

- The computational costs may be reduced by using a Markov Chain Monte Carlo approach for the sampling.

[Bachem, Lucic, Hassani and Krause 2016]

This approach avoids having to compute all $n \cdot c$ distances.

Learning Vector Quantization

Adaptation of reference vectors / codebook vectors

- May be seen as an “online” version of (batch) c -means clustering.
- For each training pattern find the closest reference vector.
- Adapt only this reference vector (winner neuron).
- For classified data the class may be taken into account:
Each reference vector is assigned to a class.

Attraction rule (data point and reference vector have same class)

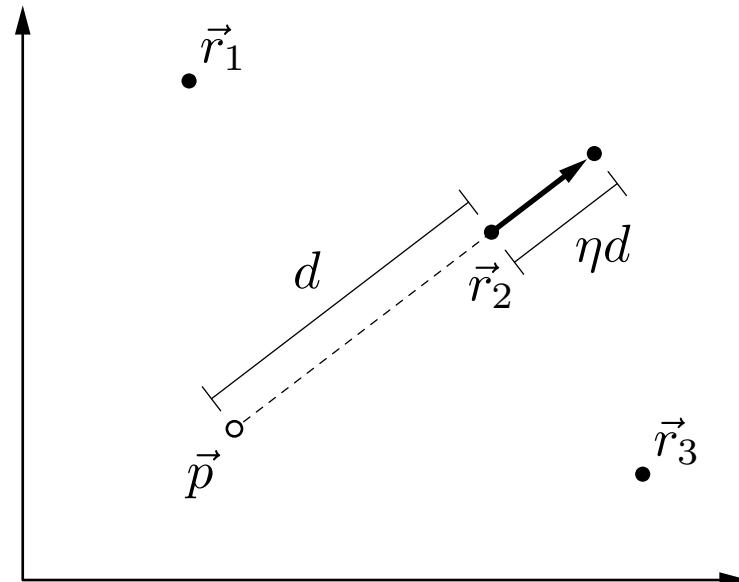
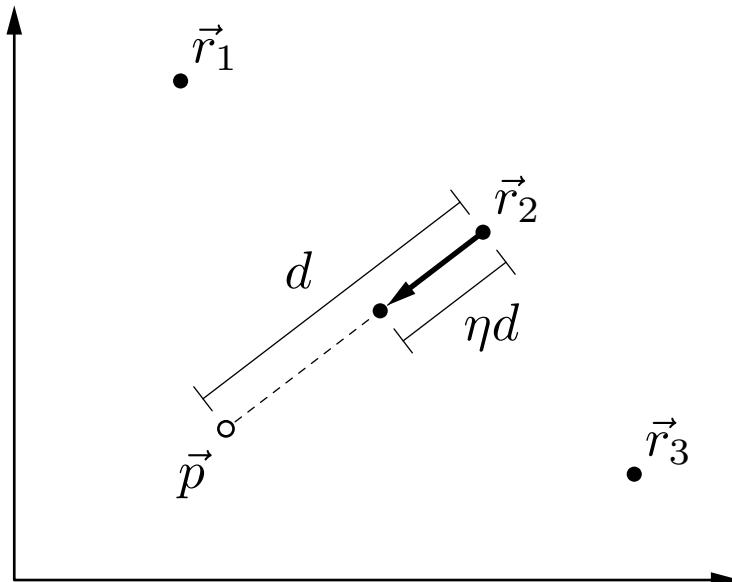
$$\vec{r}^{(\text{new})} = \vec{r}^{(\text{old})} + \eta(\vec{x} - \vec{r}^{(\text{old})}),$$

Repulsion rule (data point and reference vector have different class)

$$\vec{r}^{(\text{new})} = \vec{r}^{(\text{old})} - \eta(\vec{x} - \vec{r}^{(\text{old})}).$$

Learning Vector Quantization

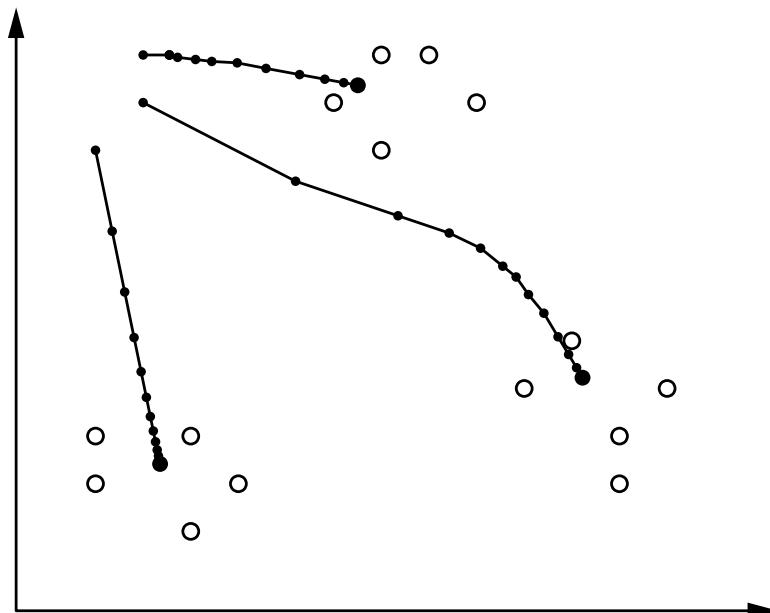
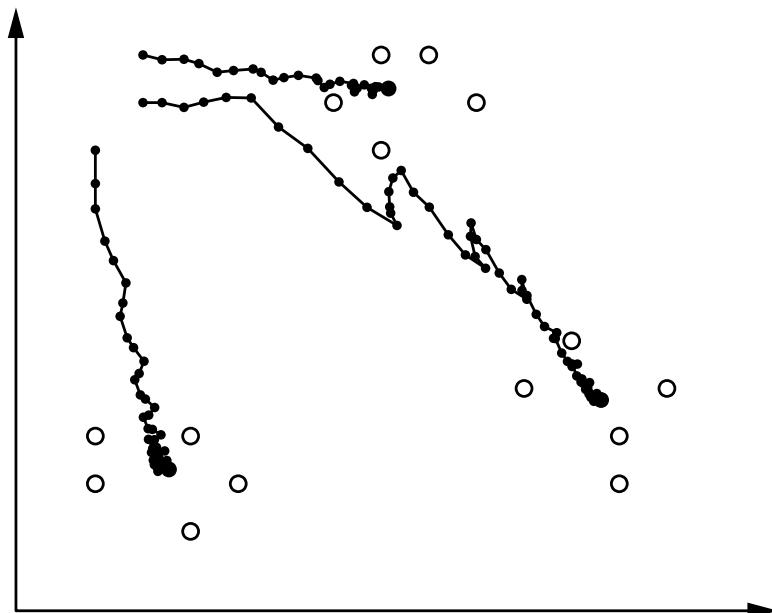
Adaptation of reference vectors / codebook vectors



- \vec{x} : data point, \vec{r}_i : reference vector
- $\eta = 0.4$ (learning rate)

Learning Vector Quantization: Example

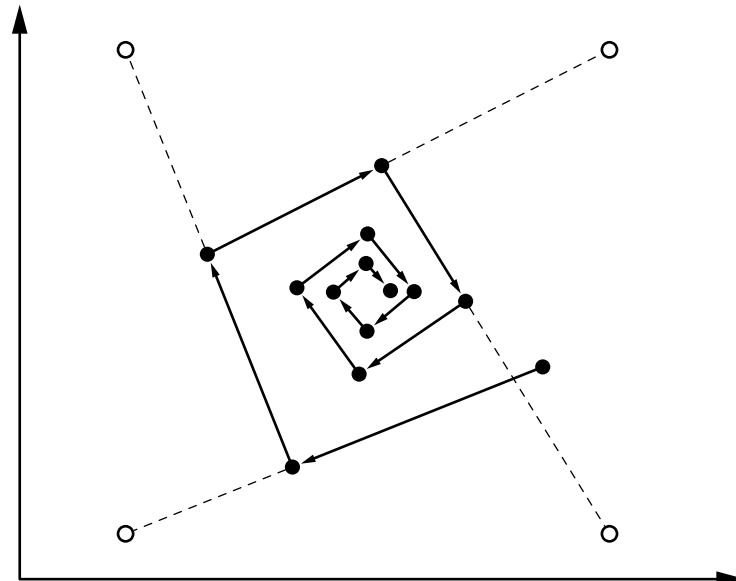
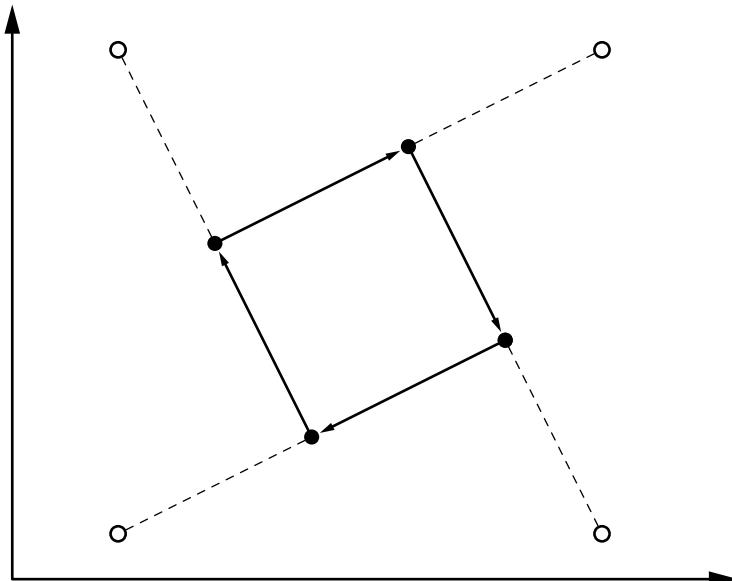
Adaptation of reference vectors / codebook vectors



- Left: Online training with learning rate $\eta = 0.1$,
- Right: Batch training with learning rate $\eta = 0.05$.

Learning Vector Quantization: Learning Rate Decay

Problem: fixed learning rate can lead to oscillations



Solution: **time dependent learning rate**

$$\eta(t) = \eta_0 \alpha^t, \quad 0 < \alpha < 1, \quad \text{or} \quad \eta(t) = \eta_0 t^\kappa, \quad \kappa > 0.$$

Learning Vector Quantization: Classified Data

Improved update rule for classified data

- **Idea:** Update not only the one reference vector that is closest to the data point (the winner neuron), but **update the two closest reference vectors.**
- Let \vec{x} be the currently processed data point and c its class.
Let \vec{r}_j and \vec{r}_k be the two closest reference vectors and z_j and z_k their classes.
- Reference vectors are updated only if $z_j \neq z_k$ and either $c = z_j$ or $c = z_k$.
(Without loss of generality we assume $c = z_j$.)
The **update rules** for the two closest reference vectors are:

$$\begin{aligned}\vec{r}_j^{(\text{new})} &= \vec{r}_j^{(\text{old})} + \eta(\vec{x} - \vec{r}_j^{(\text{old})}) \quad \text{and} \\ \vec{r}_k^{(\text{new})} &= \vec{r}_k^{(\text{old})} - \eta(\vec{x} - \vec{r}_k^{(\text{old})}),\end{aligned}$$

while all other reference vectors remain unchanged.

Learning Vector Quantization: Window Rule

- It was observed in practical tests that standard learning vector quantization may drive the reference vectors further and further apart.
- To counteract this undesired behavior a **window rule** was introduced: update only if the data point \vec{x} is close to the classification boundary.
- “Close to the boundary” is made formally precise by requiring

$$\min \left(\frac{d(\vec{x}, \vec{r}_j)}{d(\vec{x}, \vec{r}_k)}, \frac{d(\vec{x}, \vec{r}_k)}{d(\vec{x}, \vec{r}_j)} \right) > \theta, \quad \text{where} \quad \theta = \frac{1 - \xi}{1 + \xi}.$$

ξ is a parameter that has to be specified by a user.

- Intuitively, ξ describes the “width” of the window around the classification boundary, in which the data point has to lie in order to lead to an update.
- Using it prevents divergence, because the update ceases for a data point once the classification boundary has been moved far enough away.

Fuzzy Clustering

- Allow degrees of membership of a datum to different clusters.
(Classical c -means clustering assigns data crisply.)

Objective Function: (to be minimized)

$$J(\mathbf{X}, \mathbf{B}, \mathbf{U}) = \sum_{i=1}^c \sum_{j=1}^n u_{ij}^w d^2(\beta_i, \vec{x}_j)$$

- $\mathbf{U} = [u_{ij}]$ is the $c \times n$ fuzzy partition matrix,
 $u_{ij} \in [0, 1]$ is the membership degree of the data point \vec{x}_j to the i -th cluster.
- $\mathbf{B} = \{\beta_1, \dots, \beta_c\}$ is the set of cluster prototypes.
- w is the so-called “fuzzifier” (the higher w , the softer the cluster boundaries).
- Constraints:

$$\forall i \in \{1, \dots, c\} : \sum_{j=1}^n u_{ij} > 0 \quad \text{and} \quad \forall j \in \{1, \dots, n\} : \sum_{i=1}^c u_{ij} = 1.$$

Relation to Classical c -Means Clustering:

- Classical c -means clustering can be seen as optimizing the objective function

$$J(\mathbf{X}, \mathbf{B}, \mathbf{U}) = \sum_{i=1}^c \sum_{j=1}^n u_{ij} d^2(\beta_i, \vec{x}_j),$$

where $\forall i, j : u_{ij} \in \{0, 1\}$ (i.e. hard assignment of the data points) and the cluster prototypes β_i consist only of cluster centers.

- To obtain a fuzzy assignment of the data points, it is not enough to extend the range of values for the u_{ij} to the unit interval $[0, 1]$: The objective function J is optimized for a hard assignment (each data point is assigned to the closest cluster center).
- **To achieve actual degrees of membership:**
Apply a convex function $h : [0, 1] \rightarrow [0, 1]$ to the membership degrees u_{ij} .
Most common choice: $h(u) = u^w$, usually with $w = 2$.

Reminder: Function Optimization

Task: Find values $\vec{x} = (x_1, \dots, x_m)$ such that $f(\vec{x}) = f(x_1, \dots, x_m)$ is optimal.

Often feasible approach:

- A necessary condition for a (local) optimum (maximum or minimum) is that the partial derivatives w.r.t. the parameters vanish (Pierre Fermat).
- Therefore: (Try to) solve the equation system that results from setting all partial derivatives w.r.t. the parameters equal to zero.

Example task: Minimize $f(x, y) = x^2 + y^2 + xy - 4x - 5y$.

Solution procedure:

1. Take the partial derivatives of the objective function and set them to zero:

$$\frac{\partial f}{\partial x} = 2x + y - 4 = 0, \quad \frac{\partial f}{\partial y} = 2y + x - 5 = 0.$$

2. Solve the resulting (here: linear) equation system: $x = 1, \quad y = 2$.

Function Optimization with Constraints

Often a function has to be optimized subject to certain **constraints**.

Here: restriction to k **equality constraints** $C_i(\vec{x}) = 0, i = 1, \dots, k$.

Note: the equality constraints describe a subspace of the domain of the function.

Problem of optimization with constraints:

- The gradient of the objective function f may vanish outside the constrained subspace, leading to an unacceptable solution (violating the constraints).
- At an optimum *in the constrained subspace* the derivatives need not vanish.

One way to handle this problem are **generalized coordinates**:

- Exploit the dependence between the parameters specified in the constraints to express some parameters in terms of the others and thus reduce the set \vec{x} to a set \vec{x}' of independent parameters (*generalized coordinates*).
- Problem: Can be clumsy and cumbersome, if possible at all, because the form of the constraints may not allow for expressing some parameters as proper functions of the others.

Function Optimization with Constraints

A much more elegant approach is based on the following nice insights:

Let \vec{x}^* be a (local) optimum of $f(\vec{x})$ *in the constrained subspace*. Then:

- The gradient $\nabla_{\vec{x}} f(\vec{x}^*)$, if it does not vanish, must be **perpendicular** to the constrained subspace. (If $\nabla_{\vec{x}} f(\vec{x}^*)$ had a component in the constrained subspace, \vec{x}^* would not be a (local) optimum in this subspace.)
- The gradients $\nabla_{\vec{x}} C_j(\vec{x}^*)$, $1 \leq j \leq k$, must all be **perpendicular** to the constrained subspace, because they are constant, namely 0, in this subspace. Together they span the subspace perpendicular to the constrained subspace.
- Therefore it must be possible to find values λ_j , $1 \leq j \leq k$, such that

$$\nabla_{\vec{x}} f(\vec{x}^*) + \sum_{j=1}^s \lambda_j \nabla_{\vec{x}} C_j(\vec{x}^*) = 0.$$

If the constraints (and thus their gradients) are linearly independent, the values λ_j are uniquely determined. This equation can be used to **compensate the gradient** of $f(\vec{x}^*)$ so that it vanishes at \vec{x}^* .

Function Optimization: Lagrange Theory

As a consequence of these insights we obtain the

Method of Lagrange Multipliers:

Given:

- a function $f(\vec{x})$, which is to be optimized,
- k equality constraints $C_j(\vec{x}) = 0$, $1 \leq j \leq k$.

Procedure:

1. Construct the so-called **Lagrange function** by incorporating the equality constraints C_i , $i = 1, \dots, k$, with (unknown) **Lagrange multipliers** λ_i :

$$L(\vec{x}, \lambda_1, \dots, \lambda_k) = f(\vec{x}) + \sum_{i=1}^k \lambda_i C_i(\vec{x}).$$

2. Set the partial derivatives of the Lagrange function equal to zero:

$$\frac{\partial L}{\partial x_1} = 0, \quad \dots, \quad \frac{\partial L}{\partial x_m} = 0, \quad \frac{\partial L}{\partial \lambda_1} = 0, \quad \dots, \quad \frac{\partial L}{\partial \lambda_k} = 0.$$

3. (Try to) solve the resulting equation system.

Function Optimization: Lagrange Theory

Observations:

- Due to the representation of the gradient of $f(\vec{x})$ at a local optimum \vec{x}^* in the constrained subspace (see above) the gradient of L w.r.t. \vec{x} vanishes at \vec{x}^* .
→ The standard approach works again!
- If the constraints are satisfied, the additional terms have no influence.
→ The original task is not modified (same objective function).
- Taking the partial derivative w.r.t. a Lagrange multiplier reproduces the corresponding equality constraint:

$$\forall j; 1 \leq j \leq k : \quad \frac{\partial}{\partial \lambda_j} L(\vec{x}, \lambda_1, \dots, \lambda_k) = C_j(\vec{x}),$$

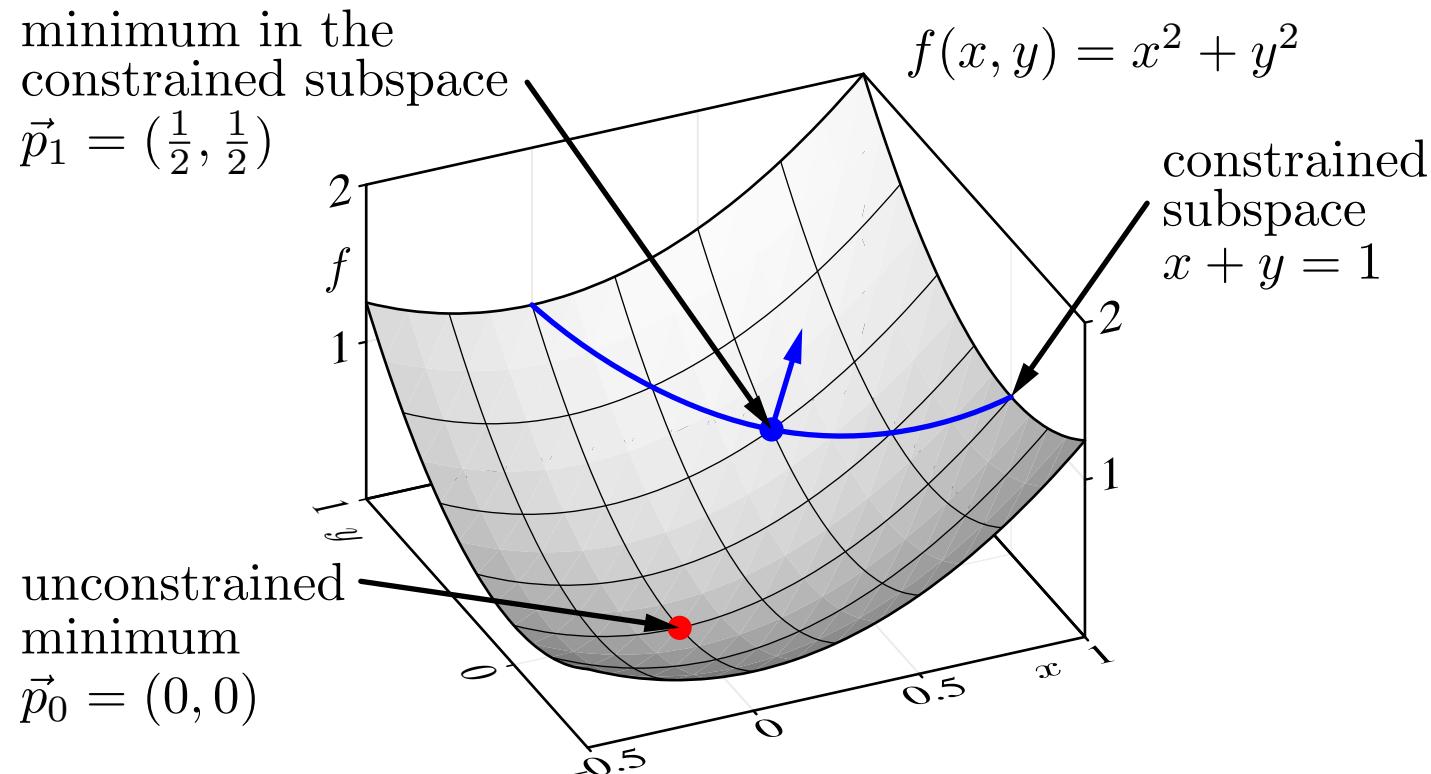
→ Constraints enter the equation system to solve in a natural way.

Remark:

- **Inequality** constraints can be handled with the **Kuhn–Tucker theory**.

Lagrange Theory: Example 1

Example task: Minimize $f(x, y) = x^2 + y^2$ subject to $x + y = 1$.



The unconstrained minimum is not in the constrained subspace, and at the minimum in the constrained subspace the gradient does not vanish.

Lagrange Theory: Example 1

Example task: Minimize $f(x, y) = x^2 + y^2$ subject to $x + y = 1$.

Solution procedure:

1. Rewrite the constraint, so that one side gets zero: $x + y - 1 = 0$.
2. Construct the Lagrange function by incorporating the constraint into the objective function with a Lagrange multiplier λ :

$$L(x, y, \lambda) = x^2 + y^2 + \lambda(x + y - 1).$$

3. Take the partial derivatives of the Lagrange function and set them to zero (necessary conditions for a minimum):

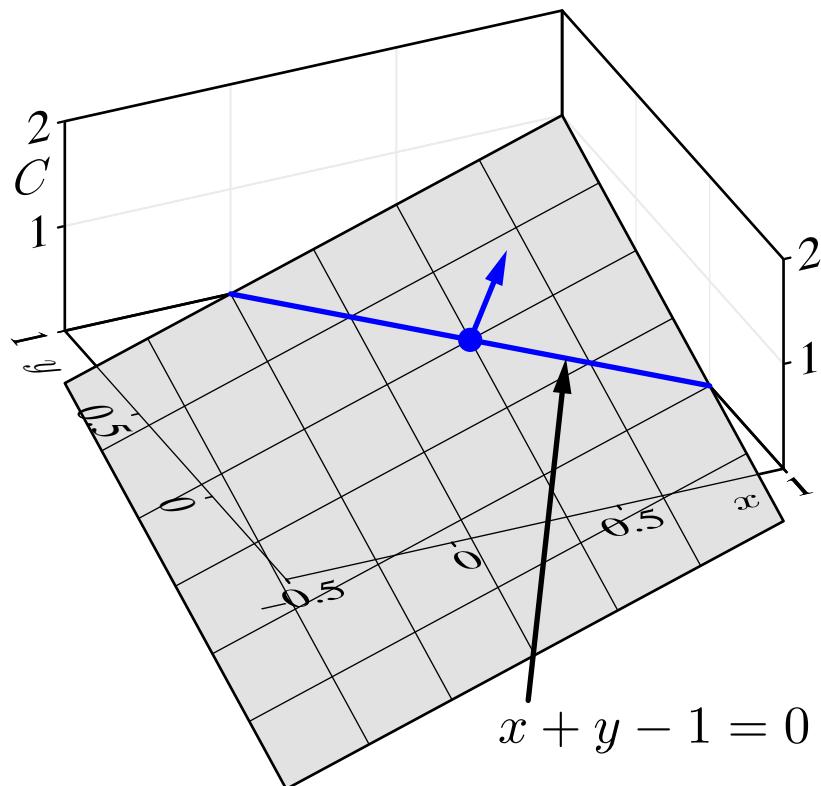
$$\frac{\partial L}{\partial x} = 2x + \lambda = 0, \quad \frac{\partial L}{\partial y} = 2y + \lambda = 0, \quad \frac{\partial L}{\partial \lambda} = x + y - 1 = 0.$$

4. Solve the resulting (here: linear) equation system:

$$\lambda = -1, \quad x = y = \frac{1}{2}.$$

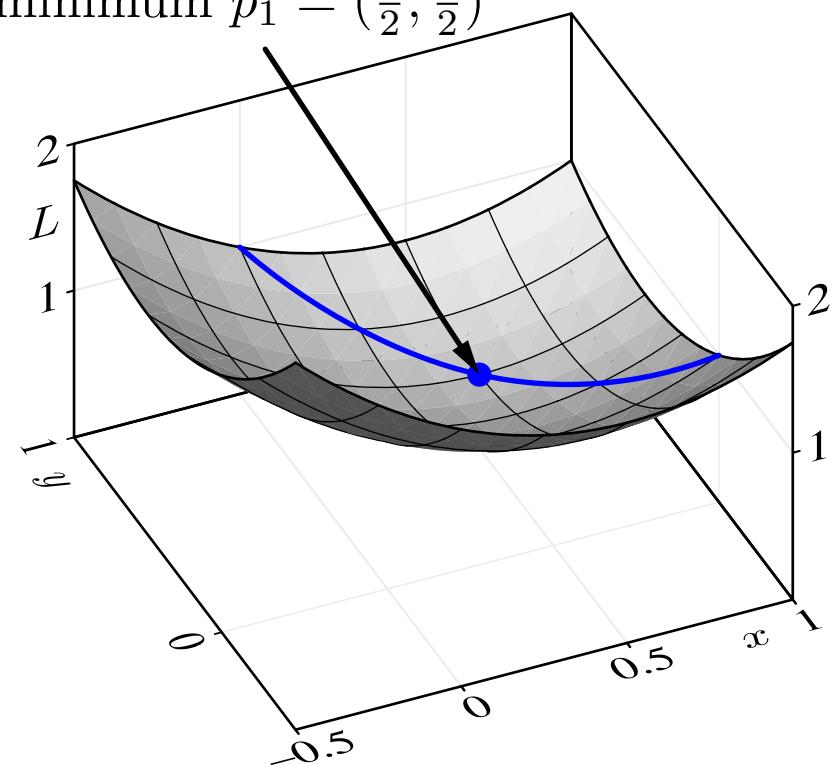
Lagrange Theory: Example 1

$$C(x, y) = x + y - 1$$



$$L(x, y, -1) = x^2 + y^2 - (x + y - 1)$$

$$\text{minimum } \vec{p}_1 = (\frac{1}{2}, \frac{1}{2})$$



The gradient of the constraint is perpendicular to the constrained subspace.
The (unconstrained) minimum of the Lagrange function $L(x, y, \lambda)$
is the minimum of the objective function $f(x, y)$ in the constrained subspace.

Lagrange Theory: Example 2

Example task: Find the side lengths x, y, z of a box with maximum volume for a given area S of the surface.

Formally: Maximize $f(x, y, z) = xyz$
 subject to $2xy + 2xz + 2yz = S.$

Solution procedure:

1. The constraint is $C(x, y, z) = 2xy + 2xz + 2yz - S = 0.$
2. The Lagrange function is

$$L(x, y, z, \lambda) = xyz + \lambda(2xy + 2xz + 2yz - S).$$

3. Taking the partial derivatives yields (in addition to the constraint):

$$\frac{\partial L}{\partial x} = yz + 2\lambda(y + z) = 0, \quad \frac{\partial L}{\partial y} = xz + 2\lambda(x + z) = 0, \quad \frac{\partial L}{\partial z} = xy + 2\lambda(x + y) = 0.$$

4. The solution is: $\lambda = -\frac{1}{4}\sqrt{\frac{S}{6}}, \quad x = y = z = \sqrt{\frac{S}{6}}$ (i.e., the box is a cube).

Fuzzy Clustering: Alternating Optimization

Objective function: (to be minimized)

$$J(\mathbf{X}, \mathbf{B}, \mathbf{U}) = \sum_{i=1}^c \sum_{j=1}^n u_{ij}^w d^2(\vec{x}_j, \beta_i)$$

Constraints:

$$\forall i \in \{1, \dots, c\} : \sum_{j=1}^n u_{ij} > 0 \quad \text{and} \quad \forall j \in \{1, \dots, n\} : \sum_{i=1}^c u_{ij} = 1.$$

- **Problem:** The objective function J cannot be minimized directly.
- Therefore: **Alternating Optimization**
 - Optimize membership degrees for fixed cluster parameters.
 - Optimize cluster parameters for fixed membership degrees.
(Update formulae are derived by differentiating the objective function J)
 - Iterate until convergence (checked, e.g., by change of cluster center).

Fuzzy Clustering: Alternating Optimization

First Step: Fix the cluster parameters.

Introduce Lagrange multipliers λ_j , $0 \leq j \leq n$, to incorporate the constraints $\forall j; 1 \leq j \leq n : \sum_{i=1}^c u_{ij} = 1$. This yields the Lagrange function (to be minimized)

$$L(\mathbf{X}, \mathbf{B}, \mathbf{U}, \Lambda) = \underbrace{\sum_{i=1}^c \sum_{j=1}^n u_{ij}^w d_{ij}^2}_{=J(\mathbf{X}, \mathbf{B}, \mathbf{U})} + \sum_{j=1}^n \lambda_j \left(1 - \sum_{i=1}^c u_{ij} \right),$$

A necessary condition for the minimum is that the partial derivatives of the Lagrange function w.r.t. the membership degrees vanish, i.e.,

$$\frac{\partial}{\partial u_{kl}} L(\mathbf{X}, \mathbf{B}, \mathbf{U}, \Lambda) = w u_{kl}^{w-1} d_{kl}^2 - \lambda_l \stackrel{!}{=} 0,$$

which leads to

$$\forall i; 1 \leq i \leq c : \forall j; 1 \leq j \leq n : \quad u_{ij} = \left(\frac{\lambda_j}{w d_{ij}^2} \right)^{\frac{1}{w-1}}.$$

Fuzzy Clustering: Alternating Optimization

Summing these equations over the clusters (in order to be able to exploit the corresponding constraints on the membership degrees), we get

$$1 = \sum_{i=1}^c u_{ij} = \sum_{i=1}^c \left(\frac{\lambda_j}{w d_{ij}^2} \right)^{\frac{1}{w-1}}.$$

Consequently the λ_j , $1 \leq j \leq n$, are

$$\lambda_j = \left(\sum_{i=1}^c \left(w d_{ij}^2 \right)^{\frac{1}{1-w}} \right)^{1-w}.$$

Inserting this into the equation for the membership degrees yields

$$\forall i; 1 \leq i \leq c : \forall j; 1 \leq j \leq n : \quad u_{ij} = \frac{d_{ij}^{\frac{2}{1-w}}}{\sum_{k=1}^c d_{kj}^{\frac{2}{1-w}}}.$$

This update formula results regardless of the distance measure.

Standard Fuzzy Clustering Algorithms

Fuzzy C-Means Algorithm: Euclidean distance

$$d_{\text{fcm}}^2(\vec{x}_j, \beta_i) = (\vec{x}_j - \vec{\mu}_i)^\top (\vec{x}_j - \vec{\mu}_i)$$

Necessary condition for a minimum: gradients w.r.t. cluster centers vanish.

$$\begin{aligned} \nabla_{\vec{\mu}_k} J_{\text{fcm}}(\mathbf{X}, \mathbf{B}, \mathbf{U}) &= \nabla_{\vec{\mu}_k} \sum_{i=1}^c \sum_{j=1}^n u_{ij}^w (\vec{x}_j - \vec{\mu}_i)^\top (\vec{x}_j - \vec{\mu}_i) \\ &= \sum_{j=1}^n u_{kj}^w \nabla_{\vec{\mu}_k} (\vec{x}_j - \vec{\mu}_k)^\top (\vec{x}_j - \vec{\mu}_k) \\ &= -2 \sum_{j=1}^n u_{kj}^w (\vec{x}_j - \vec{\mu}_k) \stackrel{!}{=} \vec{0} \end{aligned}$$

Resulting update rule for the cluster centers (**second step** of alt. optimization):

$$\forall i; 1 \leq i \leq c : \quad \vec{\mu}_i = \frac{\sum_{j=1}^n u_{ij}^w \vec{x}_j}{\sum_{j=1}^n u_{ij}^w}$$

Standard Fuzzy Clustering Algorithms

Gustafson–Kessel Algorithm: Mahalanobis distance

$$d_{\text{gk}}^2(\vec{x}_j, \beta_i) = (\vec{x}_j - \vec{\mu}_i)^\top \mathbf{C}_i^{-1} (\vec{x}_j - \vec{\mu}_i)$$

Additional constraints: $|\mathbf{C}_i| = 1$ (all cluster have unit size).

These constraints are incorporated again by Lagrange multipliers.

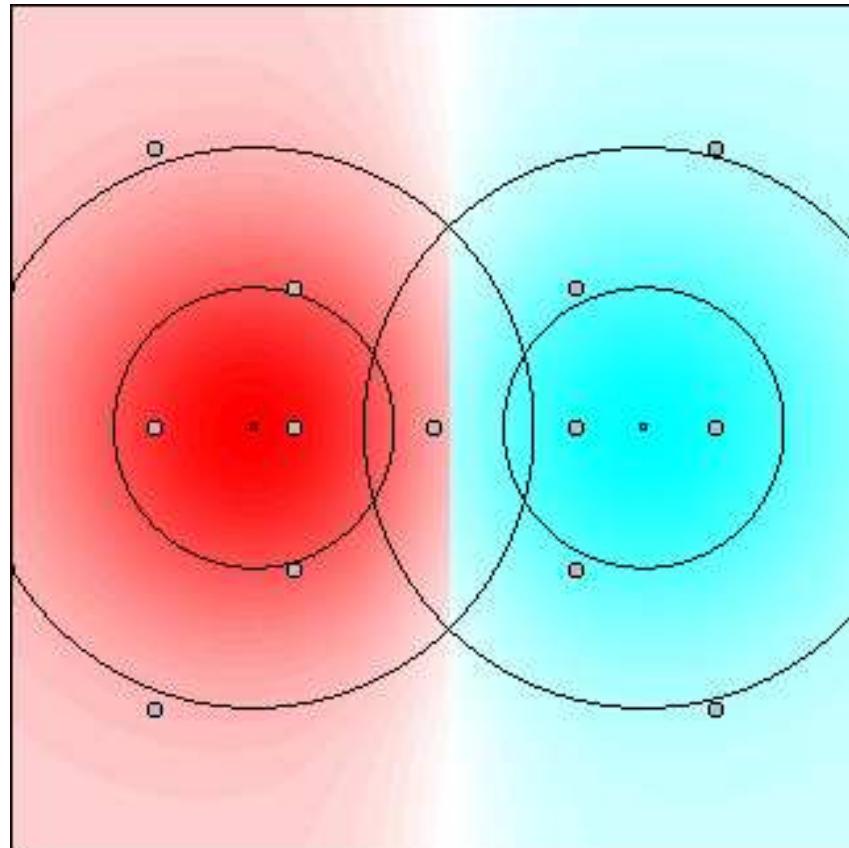
A similar derivation as for the fuzzy c -means algorithm yields the same update rule for the cluster centers:

$$\forall i; 1 \leq i \leq c : \quad \vec{\mu}_i = \frac{\sum_{j=1}^n u_{ij}^w \vec{x}_j}{\sum_{j=1}^n u_{ij}^w}$$

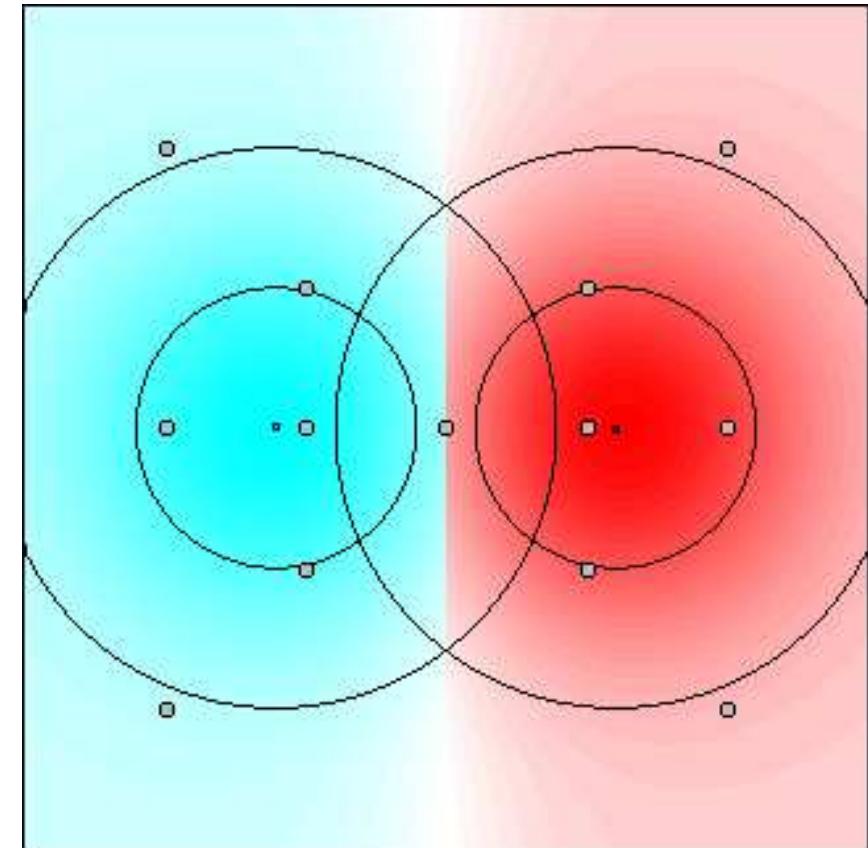
Update rule for the covariance matrices (m is the number of dimensions):

$$\mathbf{C}_i = \frac{1}{\sqrt[m]{|\Sigma_i|}} \Sigma_i \quad \text{where} \quad \Sigma_i = \sum_{j=1}^n u_{ij}^w (\vec{x}_j - \vec{\mu}_i)(\vec{x}_j - \vec{\mu}_i)^\top.$$

Fuzzy Clustering: Overlapping Clusters

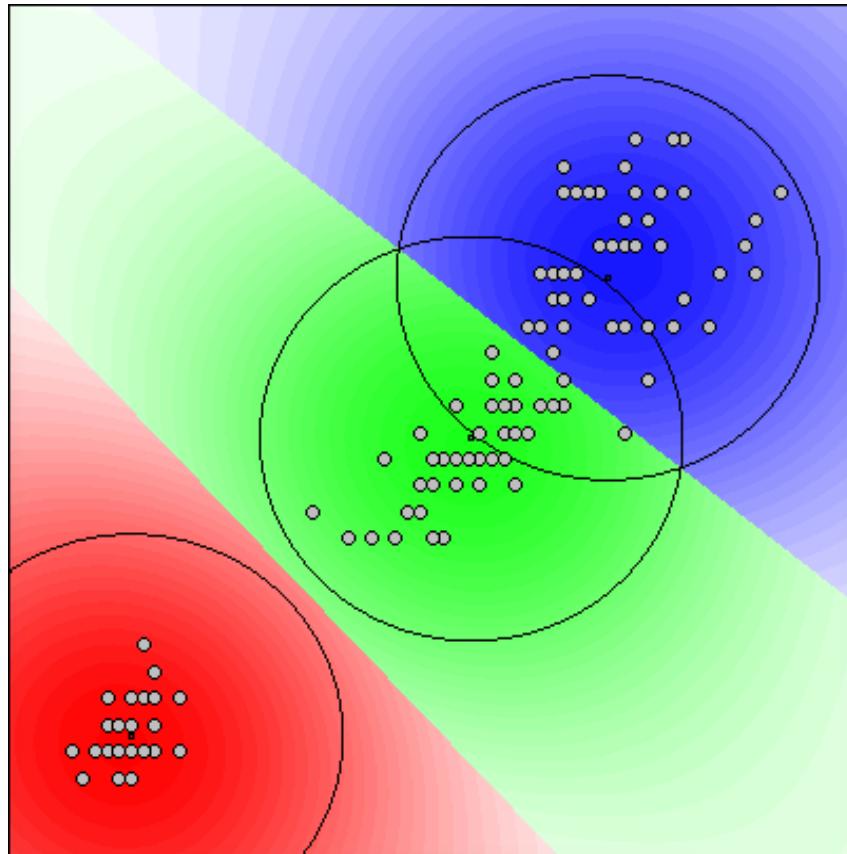


Classical c -Means

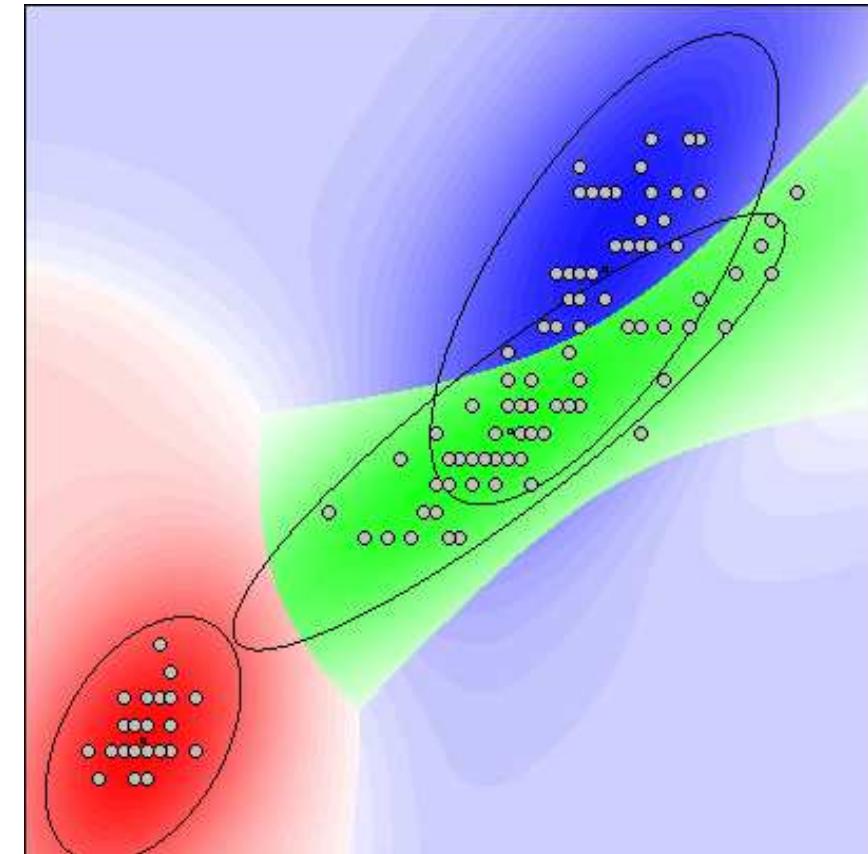


Fuzzy c -Means

Fuzzy Clustering of the Iris Data



Fuzzy c -Means



Gustafson–Kessel

Expectation Maximization: Mixture of Gaussians

- **Assumption:** Data was generated by sampling a set of normal distributions.
(The probability density is a mixture of Gaussian distributions.)
- **Formally:** We assume that the probability density can be described as

$$f_{\vec{X}}(\vec{x}; \mathbf{C}) = \sum_{y=1}^c f_{\vec{X}, Y}(\vec{x}, y; \mathbf{C}) = \sum_{y=1}^c p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}|y; \mathbf{C}).$$

\mathbf{C}	is the set of cluster parameters
\vec{X}	is a random vector that has the data space as its domain
Y	is a random variable that has the cluster indices as possible values (i.e., $\text{dom}(\vec{X}) = \mathbb{R}^m$ and $\text{dom}(Y) = \{1, \dots, c\}$)
$p_Y(y; \mathbf{C})$	is the probability that a data point belongs to (is generated by) the y -th component of the mixture
$f_{\vec{X} Y}(\vec{x} y; \mathbf{C})$	is the conditional probability density function of a data point given the cluster (specified by the cluster index y)

Expectation Maximization

- **Basic idea:** Do a maximum likelihood estimation of the cluster parameters.
- **Problem:** The likelihood function,

$$L(\mathbf{X}; \mathbf{C}) = \prod_{j=1}^n f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}) = \prod_{j=1}^n \sum_{y=1}^c p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}_j|y; \mathbf{C}),$$

is difficult to optimize, even if one takes the natural logarithm (cf. the maximum likelihood estimation of the parameters of a normal distribution), because

$$\ln L(\mathbf{X}; \mathbf{C}) = \sum_{j=1}^n \ln \sum_{y=1}^c p_Y(y; \mathbf{C}) \cdot f_{\vec{X}|Y}(\vec{x}_j|y; \mathbf{C})$$

contains the natural logarithms of complex sums.

- **Approach:** Assume that there are “hidden” variables Y_j stating the clusters that generated the data points \vec{x}_j , so that the sums reduce to one term.
- **Problem:** Since the Y_j are hidden, we do not know their values.

Expectation Maximization

- **Formally:** Maximize the likelihood of the “completed” data set (\mathbf{X}, \vec{y}) , where $\vec{y} = (y_1, \dots, y_n)$ combines the values of the variables Y_j . That is,

$$L(\mathbf{X}, \vec{y}; \mathbf{C}) = \prod_{j=1}^n f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) = \prod_{j=1}^n p_{Y_j}(y_j; \mathbf{C}) \cdot f_{\vec{X}_j|Y_j}(\vec{x}_j|y_j; \mathbf{C}).$$

- **Problem:** Since the Y_j are hidden, the values y_j are unknown (and thus the factors $p_{Y_j}(y_j; \mathbf{C})$ cannot be computed).
- **Approach to find a solution nevertheless:**
 - See the Y_j as random variables (the values y_j are not fixed) and consider a probability distribution over the possible values.
 - As a consequence $L(\mathbf{X}, \vec{y}; \mathbf{C})$ becomes a random variable, even for a fixed data set \mathbf{X} and fixed cluster parameters \mathbf{C} .
 - Try to **maximize the expected value** of $L(\mathbf{X}, \vec{y}; \mathbf{C})$ or $\ln L(\mathbf{X}, \vec{y}; \mathbf{C})$ (hence the name **expectation maximization**).

Expectation Maximization

- **Formally:** Find the cluster parameters as

$$\hat{\mathbf{C}} = \underset{\mathbf{C}}{\operatorname{argmax}} E([\ln]L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}),$$

that is, maximize the expected likelihood

$$E(L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1, \dots, c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y} \mid \mathbf{X}; \mathbf{C}) \cdot \prod_{j=1}^n f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C})$$

or, alternatively, maximize the expected log-likelihood

$$E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}) = \sum_{\vec{y} \in \{1, \dots, c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y} \mid \mathbf{X}; \mathbf{C}) \cdot \sum_{j=1}^n \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}).$$

- Unfortunately, these functionals are still difficult to optimize directly.
- **Solution:** Use the equation as an iterative scheme, fixing \mathbf{C} in some terms (iteratively compute better approximations, similar to Heron's algorithm).

Excursion: Heron's Algorithm

- **Task:** Find the square root of a given number x , i.e., find $y = \sqrt{x}$.

- **Approach:** Rewrite the defining equation $y^2 = x$ as follows:

$$y^2 = x \Leftrightarrow 2y^2 = y^2 + x \Leftrightarrow y = \frac{1}{2y}(y^2 + x) \Leftrightarrow y = \frac{1}{2} \left(y + \frac{x}{y} \right).$$

- Use the resulting equation as an iteration formula, i.e., compute the sequence

$$y_{k+1} = \frac{1}{2} \left(y_k + \frac{x}{y_k} \right) \quad \text{with} \quad y_0 = 1.$$

- It can be shown that $0 \leq y_k - \sqrt{x} \leq y_{k-1} - y_n$ for $k \geq 2$.

Therefore this iteration formula provides increasingly better approximations of the square root of x and thus is a safe and simple way to compute it.

Ex.: $x = 2$: $y_0 = 1$, $y_1 = 1.5$, $y_2 \approx 1.41667$, $y_3 \approx 1.414216$, $y_4 \approx 1.414213$.

- Heron's algorithm converges very quickly and is often used in pocket calculators and microprocessors to implement the square root.

Expectation Maximization

- **Iterative scheme for expectation maximization:**

Choose some initial set \mathbf{C}_0 of cluster parameters and then compute

$$\begin{aligned}\mathbf{C}_{k+1} &= \operatorname{argmax}_{\mathbf{C}} E(\ln L(\mathbf{X}, \vec{y}; \mathbf{C}) \mid \mathbf{X}; \mathbf{C}_k) \\ &= \operatorname{argmax}_{\mathbf{C}} \sum_{\vec{y} \in \{1, \dots, c\}^n} p_{\vec{Y}|\mathcal{X}}(\vec{y} | \mathbf{X}; \mathbf{C}_k) \sum_{j=1}^n \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\ &= \operatorname{argmax}_{\mathbf{C}} \sum_{\vec{y} \in \{1, \dots, c\}^n} \left(\prod_{l=1}^n p_{Y_l|\vec{X}_l}(y_l | \vec{x}_l; \mathbf{C}_k) \right) \sum_{j=1}^n \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\ &= \operatorname{argmax}_{\mathbf{C}} \sum_{i=1}^c \sum_{j=1}^n p_{Y_j|\vec{X}_j}(i | \vec{x}_j; \mathbf{C}_k) \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}).\end{aligned}$$

- It can be shown that each EM iteration increases the likelihood of the data and that the algorithm converges to a local maximum of the likelihood function (i.e., EM is a safe way to maximize the likelihood function).

Expectation Maximization

Justification of the last step on the previous slide:

$$\begin{aligned}
 & \sum_{\vec{y} \in \{1, \dots, c\}^n} \left(\prod_{l=1}^n p_{Y_l|\vec{X}_l}(y_l | \vec{x}_l; \mathbf{C}_k) \right) \sum_{j=1}^n \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, y_j; \mathbf{C}) \\
 &= \sum_{y_1=1}^c \cdots \sum_{y_n=1}^c \left(\prod_{l=1}^n p_{Y_l|\vec{X}_l}(y_l | \vec{x}_l; \mathbf{C}_k) \right) \sum_{j=1}^n \sum_{i=1}^c \delta_{i,y_j} \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}) \\
 &= \sum_{i=1}^c \sum_{j=1}^n \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}) \sum_{y_1=1}^c \cdots \sum_{y_n=1}^c \delta_{i,y_j} \prod_{l=1}^n p_{Y_l|\vec{X}_l}(y_l | \vec{x}_l; \mathbf{C}_k) \\
 &= \sum_{i=1}^c \sum_{j=1}^n p_{Y_j|\vec{X}_j}(i | \vec{x}_j; \mathbf{C}_k) \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}) \\
 &\quad \underbrace{\sum_{y_1=1}^c \cdots \sum_{y_{j-1}=1}^c \sum_{y_{j+1}=1}^c \cdots \sum_{y_n=1}^c \prod_{l=1, l \neq j}^n p_{Y_l|\vec{X}_l}(y_l | \vec{x}_l; \mathbf{C}_k)}_{= \prod_{l=1, l \neq j}^n \sum_{y_l=1}^c p_{Y_l|\vec{X}_l}(y_l | \vec{x}_l; \mathbf{C}_k)} \\
 &= \prod_{l=1, l \neq j}^n 1 = 1
 \end{aligned}$$

Expectation Maximization

- The probabilities $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$ are computed as

$$p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) = \frac{f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C}_k)}{f_{\vec{X}_j}(\vec{x}_j; \mathbf{C}_k)} = \frac{f_{\vec{X}_j|Y_j}(\vec{x}_j|i; \mathbf{C}_k) \cdot p_{Y_j}(i; \mathbf{C}_k)}{\sum_{l=1}^c f_{\vec{X}_j|Y_j}(\vec{x}_j|l; \mathbf{C}_k) \cdot p_{Y_j}(l; \mathbf{C}_k)},$$

that is, as the relative probability densities of the different clusters (as specified by the cluster parameters) at the location of the data points \vec{x}_j .

- The $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$ are the posterior probabilities of the clusters given the data point \vec{x}_j and a set of cluster parameters \mathbf{C}_k .
- They can be seen as **case weights** of a “completed” data set:
 - Split each data point \vec{x}_j into c data points (\vec{x}_j, i) , $i = 1, \dots, c$.
 - Distribute the unit weight of the data point \vec{x}_j according to the above probabilities, i.e., assign to (\vec{x}_j, i) the weight $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$, $i = 1, \dots, c$.

Expectation Maximization: Cookbook Recipe

Core Iteration Formula

$$\mathbf{C}_{k+1} = \operatorname{argmax}_{\mathbf{C}} \sum_{i=1}^c \sum_{j=1}^n p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k) \cdot \ln f_{\vec{X}_j, Y_j}(\vec{x}_j, i; \mathbf{C})$$

Expectation Step

- For all data points \vec{x}_j :
Compute for each normal distribution the probability $p_{Y_j|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}_k)$ that the data point was generated from it
(ratio of probability densities at the location of the data point).
→ “weight” of the data point for the estimation.

Maximization Step

- For all normal distributions:
Estimate the parameters by standard maximum likelihood estimation using the probabilities (“weights”) assigned to the data points w.r.t. the distribution in the expectation step.

Expectation Maximization: Mixture of Gaussians

Expectation Step: Use Bayes' rule to compute

$$p_{C|\vec{X}}(i|\vec{x}; \mathbf{C}) = \frac{p_C(i; \mathbf{c}_i) \cdot f_{\vec{X}|C}(\vec{x}|i; \mathbf{c}_i)}{f_{\vec{X}}(\vec{x}; \mathbf{C})} = \frac{p_C(i; \mathbf{c}_i) \cdot f_{\vec{X}|C}(\vec{x}|i; \mathbf{c}_i)}{\sum_{k=1}^c p_C(k; \mathbf{c}_k) \cdot f_{\vec{X}|C}(\vec{x}|k; \mathbf{c}_k)}.$$

→ “weight” of the data point \vec{x} for the estimation.

Maximization Step: Use maximum likelihood estimation to compute

$$\varrho_i^{(t+1)} = \frac{1}{n} \sum_{j=1}^n p_{C|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)}), \quad \vec{\mu}_i^{(t+1)} = \frac{\sum_{j=1}^n p_{C|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)}) \cdot \vec{x}_j}{\sum_{j=1}^n p_{C|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)})},$$

$$\text{and } \Sigma_i^{(t+1)} = \frac{\sum_{j=1}^n p_{C|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)}) \cdot (\vec{x}_j - \vec{\mu}_i^{(t+1)}) (\vec{x}_j - \vec{\mu}_i^{(t+1)})^\top}{\sum_{j=1}^n p_{C|\vec{X}_j}(i|\vec{x}_j; \mathbf{C}^{(t)})}$$

Iterate until convergence (checked, e.g., by change of mean vector).

Expectation Maximization: Technical Problems

- If a fully general mixture of Gaussian distributions is used, the likelihood function is truly optimized if
 - all normal distributions except one are contracted to single data points and
 - the remaining normal distribution is the maximum likelihood estimate for the remaining data points.
- This undesired result is rare, because the algorithm gets stuck in a local optimum.
- Nevertheless it is recommended to take countermeasures, which consist mainly in reducing the degrees of freedom, like
 - Fix the determinants of the covariance matrices to equal values.
 - Use a diagonal instead of a general covariance matrix.
 - Use an isotropic variance instead of a covariance matrix.
 - Fix the prior probabilities of the clusters to equal values.

Hierarchical Agglomerative Clustering

- Start with every data point in its own cluster.
(i.e., start with so-called **singletons**: single element clusters)
- In each step merge those two clusters that are closest to each other.
- Keep on merging clusters until all data points are contained in one cluster.
- The result is a hierarchy of clusters that can be visualized in a tree structure
(a so-called **dendrogram** — from the Greek $\delta\acute{\epsilon}\nu\tau\varrho\omega\nu$ (dendron): tree)
- **Measuring the Distances**
 - The distance between singletons is simply the distance between the (single) data points contained in them.
 - However: How do we compute the distance between clusters that contain more than one data point?

Measuring the Distance between Clusters

- **Centroid**

(red)

Distance between the centroids (mean value vectors) of the two clusters.

- **Average Linkage**

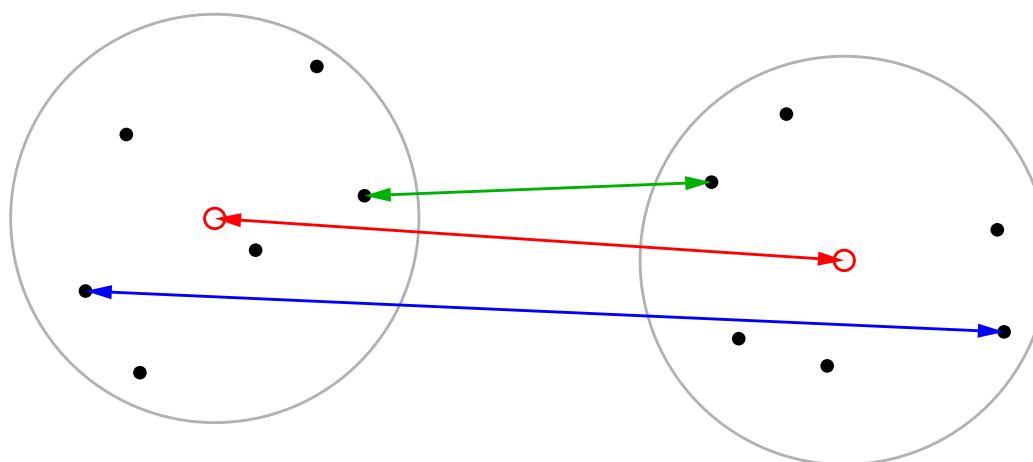
Average distance between two points of the two clusters.

- **Single Linkage** (green)

Distance between the two closest points of the two clusters.

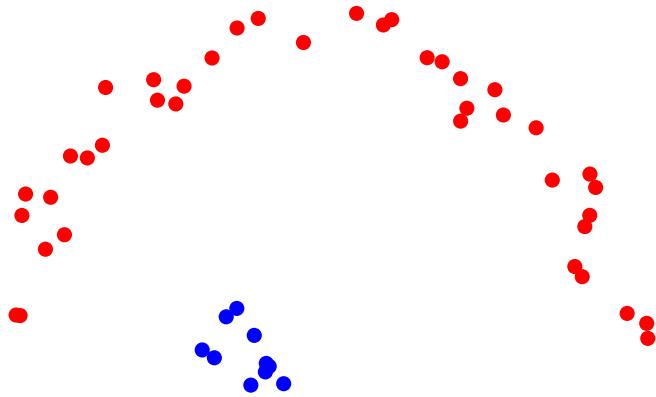
- **Complete Linkage** (blue)

Distance between the two farthest points of the two clusters.

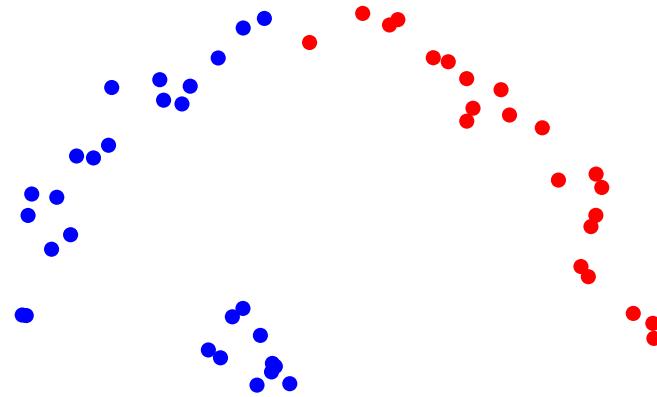


Measuring the Distance between Clusters

- Single linkage can “follow chains” in the data (may be desirable in certain applications).
- Complete linkage leads to very compact clusters, but may also “bridge gaps.”
- Average linkage also tends clearly towards compact clusters.



Single Linkage

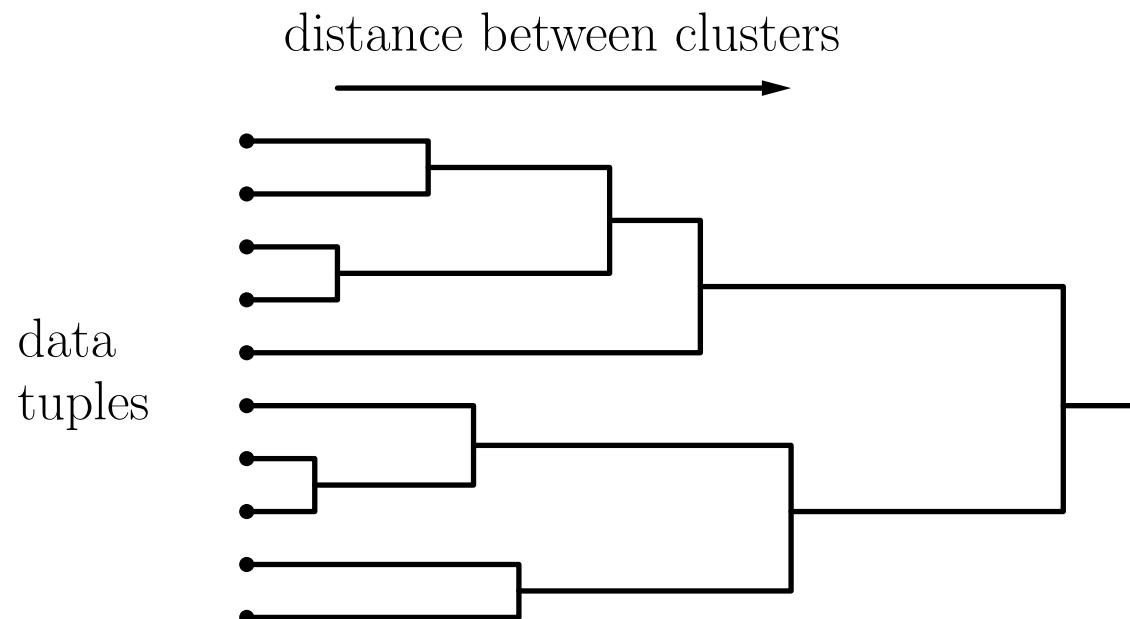


Complete Linkage

(These are the actual results that are computed for this data set; see also below.)

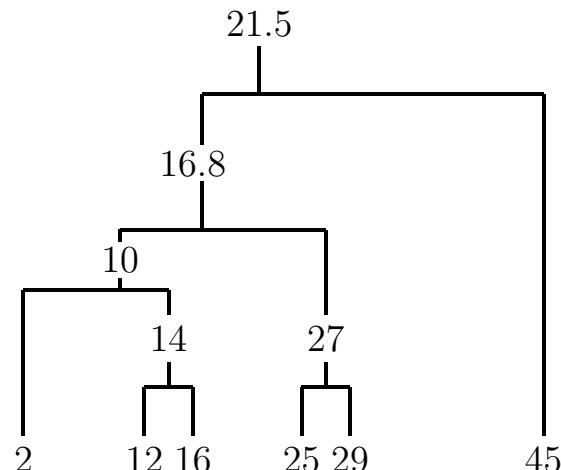
Dendrograms

- The cluster merging process arranges the data points in a binary tree.
- Draw the data tuples at the bottom or on the left (equally spaced if they are multi-dimensional).
- Draw a connection between clusters that are merged, with the distance to the data points representing the distance between the clusters.

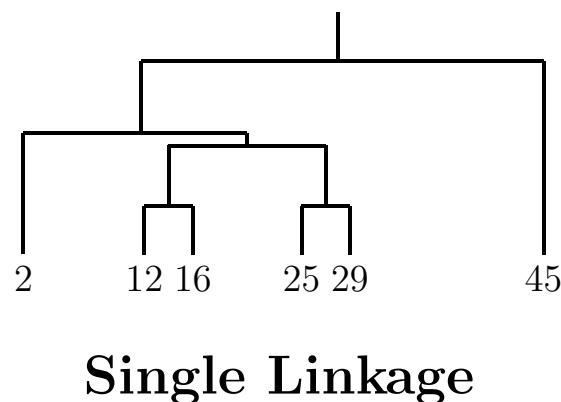


Dendrograms

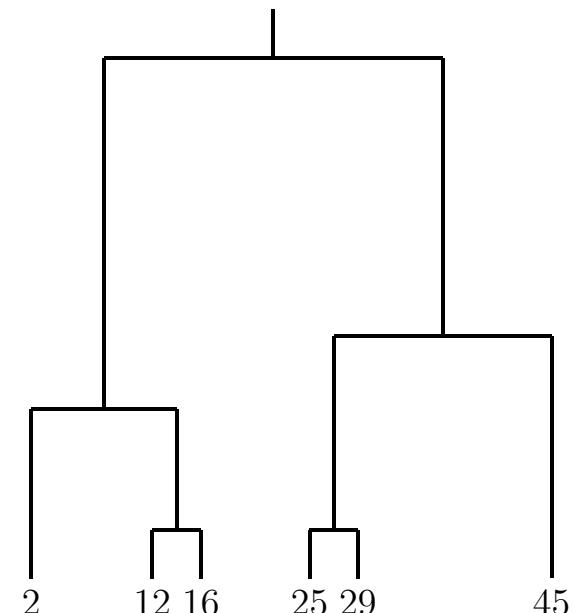
- Example: Clustering of the 1-dimensional data set $\{2, 12, 16, 25, 29, 45\}$.
- All three approaches to measure the distance between clusters lead to different dendograms.



Centroid



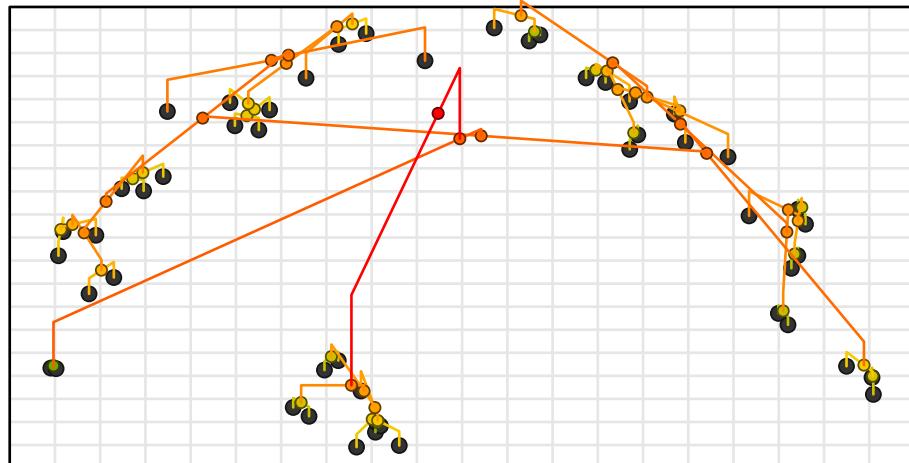
Single Linkage



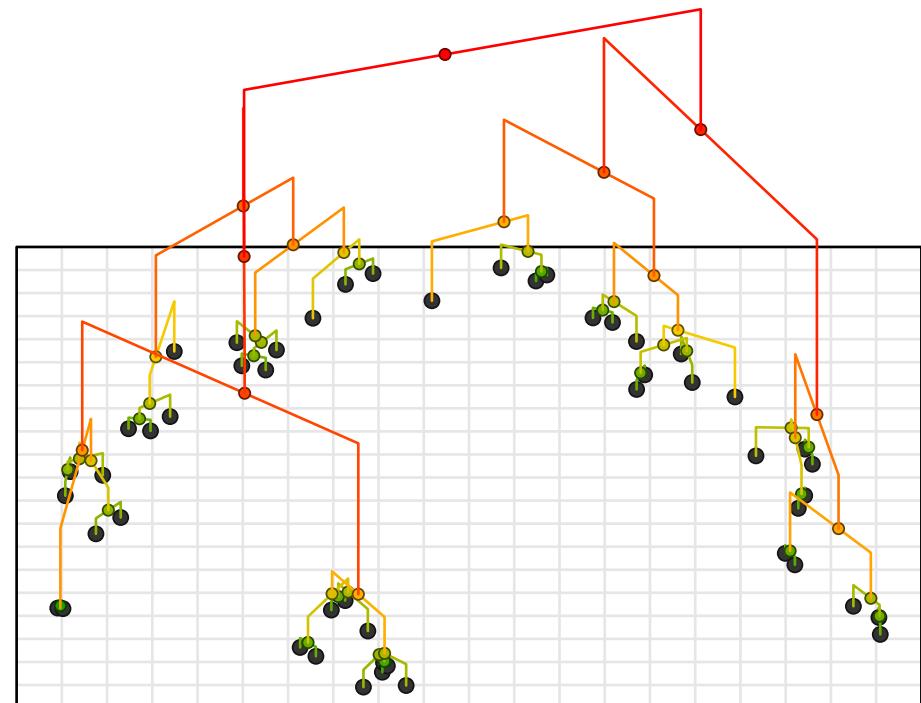
Complete Linkage

Dendrograms for “Half Circle” Data

- Single linkage can “follow chains” in the data.
- Complete linkage leads to very compact clusters, but may also “bridge gaps.”
- These dendograms use centroids.



Single Linkage



Complete Linkage

Implementation Aspects

- Hierarchical agglomerative clustering can be implemented by processing the matrix $\mathbf{D} = (d_{ij}^\kappa)_{1 \leq i,j \leq n}$ containing the pairwise (squared) distances of the data points. (The data points themselves are actually not needed.)
- In each step the rows and columns corresponding to the two clusters that are closest to each other are deleted.
- A new row and column corresponding to the cluster formed by merging these clusters is added to the matrix.
- The elements of this new row/column are computed according to ($\kappa \in \{1, 2\}$):

$$\forall k : \quad d_{k*}^\kappa = d_{*k}^\kappa = \alpha_i d_{ik}^\kappa + \alpha_j d_{jk}^\kappa + \beta d_{ij}^\kappa + \gamma |d_{ik}^\kappa - d_{jk}^\kappa|$$

i, j

indices of the two clusters that are merged

k

indices of the old clusters that are *not* merged

*

index of the new cluster (result of merger)

$\alpha_i, \alpha_j, \beta, \gamma$

parameters specifying the method (single linkage etc.)

Implementation Aspects

- The parameters defining the different methods are [Lance & Williams 1967] (n_i, n_j, n_k are the numbers of data points in the clusters):

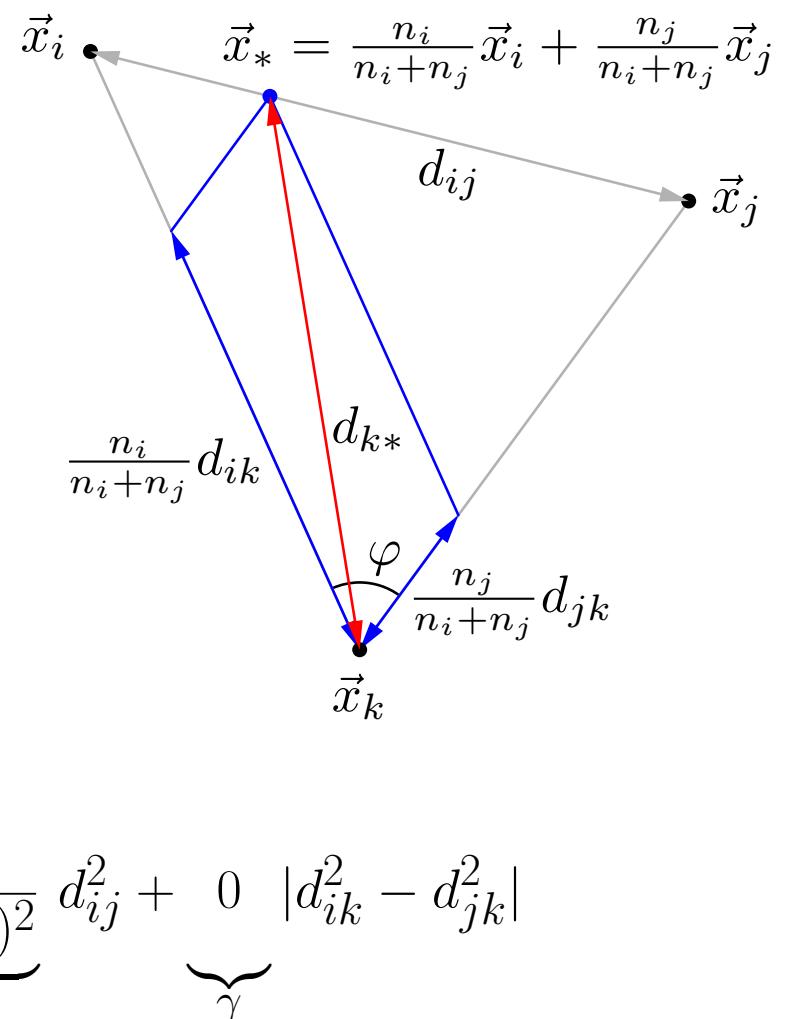
method	κ	α_i	α_j	β	γ
centroid method	2	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$-\frac{n_i n_j}{(n_i+n_j)^2}$	0
median method	1	$\frac{1}{2}$	$\frac{1}{2}$	$-\frac{1}{4}$	0
single linkage	1 or 2	$\frac{1}{2}$	$\frac{1}{2}$	0	$-\frac{1}{2}$
complete linkage	1 or 2	$\frac{1}{2}$	$\frac{1}{2}$	0	$+\frac{1}{2}$
average linkage	1	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Ward's method	2	$\frac{n_i+n_k}{n_i+n_j+n_k}$	$\frac{n_j+n_k}{n_i+n_j+n_k}$	0	0

Implementation Aspects: Centroid Formula

Application of the (planar) cosine theorem
in its two forms (parallelogram diagonals):

$$(a) d_{ij}^2 = d_{ik}^2 + d_{jk}^2 - 2d_{ik}d_{jk} \cos \varphi \\ \Leftrightarrow 2d_{ik}d_{jk} \cos \varphi = d_{ik}^2 + d_{jk}^2 - d_{ij}^2$$

$$(b) d_{k*}^2 = \left(\frac{n_i}{n_i+n_j} d_{ik} \right)^2 + \left(\frac{n_j}{n_i+n_j} d_{jk} \right)^2 \\ + \underbrace{2 \frac{n_i}{n_i+n_j} d_{ik} \frac{n_j}{n_i+n_j} d_{jk}}_{= \frac{n_i n_j}{(n_i+n_j)^2} (d_{ik}^2 + d_{jk}^2 - d_{ij}^2)} \cos \varphi \\ = \underbrace{\frac{n_i}{n_i+n_j} d_{ik}^2}_{\alpha_i} + \underbrace{\frac{n_j}{n_i+n_j} d_{jk}^2}_{\alpha_j} - \underbrace{\frac{n_i n_j}{(n_i+n_j)^2} d_{ij}^2}_{\beta} + \underbrace{0}_{\gamma} |d_{ik}^2 - d_{jk}^2|$$



Implementation Aspects: Single/Complete Linkage

Single Linkage:

$$\begin{aligned} d_{k*}^{\kappa} &= \frac{1}{2}d_{ik}^{\kappa} + \frac{1}{2}d_{jk}^{\kappa} - \frac{1}{2}|d_{ik}^{\kappa} - d_{jk}^{\kappa}| \\ &= \frac{1}{2}d_{ik}^{\kappa} + \frac{1}{2}d_{jk}^{\kappa} + \begin{cases} \frac{1}{2}d_{ik}^{\kappa} - \frac{1}{2}d_{jk}^{\kappa} & \text{if } d_{ik}^{\kappa} > d_{jk}^{\kappa}, \\ \frac{1}{2}d_{jk}^{\kappa} - \frac{1}{2}d_{ik}^{\kappa} & \text{otherwise} \end{cases} \\ &= \min\{d_{ik}^{\kappa}, d_{jk}^{\kappa}\} \end{aligned}$$

Complete Linkage:

$$\begin{aligned} d_{k*}^{\kappa} &= \frac{1}{2}d_{ik}^{\kappa} + \frac{1}{2}d_{jk}^{\kappa} + \frac{1}{2}|d_{ik}^{\kappa} - d_{jk}^{\kappa}| \\ &= \frac{1}{2}d_{ik}^{\kappa} + \frac{1}{2}d_{jk}^{\kappa} + \begin{cases} \frac{1}{2}d_{jk}^{\kappa} - \frac{1}{2}d_{ik}^{\kappa} & \text{if } d_{ik}^{\kappa} > d_{jk}^{\kappa}, \\ \frac{1}{2}d_{ik}^{\kappa} - \frac{1}{2}d_{jk}^{\kappa} & \text{otherwise} \end{cases} \\ &= \max\{d_{ik}^{\kappa}, d_{jk}^{\kappa}\} \end{aligned}$$

Choosing the Clusters

- **Simplest Approach:**

- Specify a minimum desired distance between clusters.
- Stop merging clusters if the closest two clusters are farther apart than this distance.

- **Visual Approach:**

- Merge clusters until all data points are combined into one cluster.
- Draw the dendrogram and find a good cut level.
- Advantage: Cut need not be strictly horizontal.

- **More Sophisticated Approaches:**

- Analyze the sequence of distances in the merging process.
- Try to find a step in which the distance between the two clusters merged is considerably larger than the distance of the previous step.
- Several heuristic criteria exist for this step selection.

Summary Clustering

- **Prototype-based Clustering**

- Alternating adaptation of data point assignment and cluster parameters.
- Online or batch adaptation of the cluster center.
- Crisp/hard or fuzzy/probabilistic assignment of a datum to a cluster.
- Local minima can pose a problem.
- Fuzzy/probabilistic approaches are usually more robust.

- **Hierarchical Agglomerative Clustering**

- Start with singletons (one element clusters).
- Always merge those clusters that are closest.
- Different ways to measure the distance of clusters.
- Cluster hierarchy can be depicted as a dendrogram.

Software

Software for

- Multipolynomial and Logistic Regression,
- Bayes Classifier Induction (naive and full),
- Decision and Regression Tree Induction,
- Artificial Neural Networks (MLPs, RBFNs),
- Learning Vector Quantization,
- Fuzzy and Probabilistic Clustering,
- Association Rule Induction and Frequent Item Set Mining
- Frequent Subgraph Mining / Molecular Fragment Mining

can be found at

<http://www.borgelt.net/software.html>