

Elementary Data Mining

Christian Borgelt

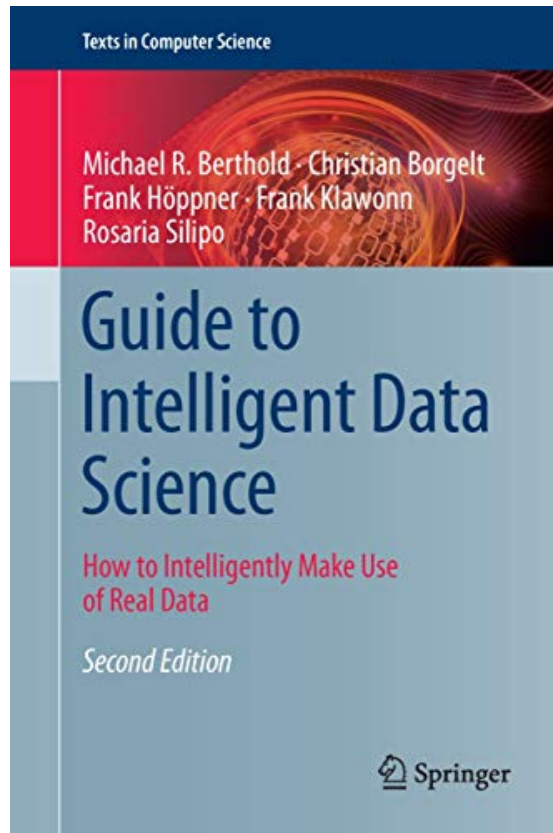
Dept. of Artificial Intelligence & Human Interfaces
Paris Lodron University of Salzburg
Jakob-Haringer-Straße 2, 5020 Salzburg, Austria

`christian.borgelt@plus.ac.at`

`christian@borgelt.net`

`https://www.borgelt.net/`

Bibliography



picture not available
in online version

picture not available
in online version

Textbook, 2nd ed.
Springer-Verlag
Heidelberg, DE 2020
(in English)

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

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

- **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).

Data

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

Knowledge

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: $T^2 \sim a^3$.

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

- Data Science / 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, sometimes even more)
 - 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** ⇒ other lectures, e.g. “Regression Methods & Computational Statistics”
(characteristic measures, parameter estimation, hypothesis testing, regression, model selection)
tasks: classification, prediction, trend analysis
- **k-nearest Neighbor / Case-based Reasoning** blue: in this lecture
(lazy learning, similarity measures, neighbor weighting, data structures for fast search)
tasks: classification, prediction
- **Bayes Classifiers** blue: in this lecture
(probabilistic classification, naive and Gaussian Bayes classifiers, Bayesian network classifiers)
tasks: classification, prediction
- **Decision and Regression Trees** blue: in this lecture
(top down induction, attribute selection measures, pruning, regression trees)
tasks: classification, prediction
- **Rule Learning** ⇒ Lecture “Advanced Data Mining 1”
(extraction from decision trees, A^q, CN2, version spaces, inductive logic programming)
tasks: classification, prediction

- **Support Vector Machines** ⇒ Lecture “Advanced Data Mining 1”
(linear and non-linear classification and regression, kernel trick, support vectors)
tasks: classification, prediction, clustering
- **Artificial Neural Networks** ⇒ Lecture “Artificial Neural Networks and Deep Learning”
(multilayer perceptrons, radial basis function networks, learning vector quantization)
tasks: classification, prediction, clustering
- **Ensemble Methods (especially Random Forests)** ⇒ Lecture “Advanced Data Mining 2”
(Bayesian voting, bagging, boosting (AdaBoost), injecting randomness, stacking)
tasks: classification, prediction
- **Cluster Analysis** green: ⇒ Lecture “Advanced Data Mining 2”
(*k*-means and fuzzy clustering, Gaussian mixtures, hierarchical agglomerative clustering)
tasks: segmentation, clustering
- **Association Rule Induction** ⇒ Lecture “Frequent Pattern Mining”
(frequent item set mining, rule generation)
tasks: association analysis

Principles of Modeling

Principles of Modeling

- The **Data Mining** step of the **KDD Process** consists mainly of **model building** for specific purposes (e.g. prediction, segmentation).
- What type of model is to be built depends on the task, e.g.,
 - if the task is numeric prediction, one may use a regression function,
 - if the task is classification, one may use a decision tree,
 - if the task is clustering, one may use a set of cluster prototypes,
 - etc.
- Most data analysis methods comprise the following four steps:
 - **Select the Model Class** (e.g. decision tree)
 - **Select the Objective Function** (e.g. misclassification rate)
 - **Apply an Optimization Algorithm** (e.g. top-down induction)
 - **Validate the Results** (e.g. cross-validation)

Model Classes

- In order to extract information from data, it is necessary to specify the general form the analysis result should have.
- We call this the **model class** or **architecture** of the analysis result.
- Attention: In Data Mining / Machine Learning the notion of a model class is considerably more general than, e.g., in statistics, where it reflects a structure inherent in the data or represents the process of data generation.
- Typical distinctions w.r.t. model classes:
 - **Type of Model** (e.g. linear function, rules, decision tree, clusters etc.)
 - **Global versus Local Model**
(e.g. regression models usually cover the whole data space while rules are applicable only in the region where their antecedent is satisfied)
 - **Interpretable versus Black Box**
(rules and decision trees are usually considered as interpretable, artificial neural networks as “black boxes”)

Supervised and Unsupervised Model Building

Fundamental Distinction for Model Building / Model Classes

- **Supervised Model Building / Supervised Learning**

- There is a target variable that the model is supposed to predict.
(nominal target: *classification*, metric target: *regression*)
- The data **D** consists of pairs (\vec{x}, y) , where \vec{x} is a tuple of descriptive attribute values and y is the target value.
- The objective is to find a model $m : \mathbf{X} \rightarrow Y$ that predicts the target y from the values \vec{x} of the descriptive attributes.

- **Unsupervised Model Building / Unsupervised Learning**

- There is *no* target variable that the model is supposed to predict.
Rather model outputs are to be chosen by the model.
- The data **D** consists of tuples \vec{x} of descriptive attributes.
- Typical objective: similar inputs should produce similar outputs.

Reinforcement Learning, Classification, Regression

A third type of model building (or learning) is (not covered in this lecture)

- **Reinforcement Learning**

- Interdisciplinary area of machine learning and optimal control.
- Concerned with what actions an intelligent agent should take in a dynamic environment in order to maximize a cumulative reward.
- Does not need labeled input/output pairs, but gets feedback to actions. Also does not need sub-optimal actions to be explicitly corrected, rather the obtained feedback might be incomplete or delayed.
- Tries to find a balance between *exploration* (of uncharted territory) and *exploitation* (of already collected knowledge).

- **Supervised Learning** is commonly further subdivided into

- **Classification** if the target attribute is nominal (or ordinal),
- **Regression** if the target attribute is metric (numeric).

Scale Types / Attribute Types

Scale Type	Meaningful Operations	Examples
nominal (categorical, qualitative)	test for equality	sex blood group
ordinal (rank scale, comparative)	test for equality greater/less than	exam grade wind strength
metric (interval scale, quantitative) (ratio scale)	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 (in °C) does not (it does in °K). time as duration does, time as calendar time does not.
- **Counts** may be considered a special type (e.g. number of children).

Model Evaluation

- After a model has been constructed, one would like to know how “good” it is.
⇒ **How can we measure the quality of a model?**
- Desired: The model should **generalize** well and thus yield, on *new* data, an error (to be made precise) that is as small as possible.
- However, due to possible **overfitting** to the induction / training data (i.e. adaptations to features that are not regular, but accidental), the error on the training data is usually not very indicative.
⇒ **How can we assess the (expected) performance on new data?**
- General idea: Evaluate on a hold-out data set (**validation data / test data**), that is, on data **not** used for building / training the predictor.
 - It is (highly) unlikely that the validation data exhibits the same accidental features as the training data.
 - Hence an evaluation on the validation data can provide a good indication of the performance on new data.

Fitting Criteria and Score / Loss Functions

- In order to find the best or at least a good model for the given data, a fitting criterion is needed, usually in the form of an **objective function**

$$f : \mathcal{M} \rightarrow \mathbb{R},$$

where \mathcal{M} is the set of considered models.

- The objective function f may also be referred to as
 - **Score Function** (usually to be maximized),
 - **Loss Function** (usually to be minimized).
- Typical examples of objective functions are ($m \in \mathcal{M}$ is a model, \mathbf{D} the data)

- Mean squared error (MSE):
$$f(m, \mathbf{D}) = \frac{1}{|\mathbf{D}|} \sum_{(\vec{x}, y) \in D} (m(\vec{x}) - y)^2$$

- Mean absolute error (MAE):
$$f(m, \mathbf{D}) = \frac{1}{|\mathbf{D}|} \sum_{(\vec{x}, y) \in D} |m(\vec{x}) - y|$$

- Accuracy:
$$f(m, \mathbf{D}) = \frac{1}{|\mathbf{D}|} \sum_{(\vec{x}, y) \in D} \delta_{m(\vec{x}), y}$$

Regression Evaluation (metric target)

mean squared error (MSE)	$f(m, \mathbf{D}) = \frac{1}{ \mathbf{D} } \sum_{(\vec{x}, y) \in D} (m(\vec{x}) - y)^2$
most commonly used loss function, emphasizes large distances / errors	
root mean squared error (RMSE)	$f(m, \mathbf{D}) = \sqrt{\frac{1}{ \mathbf{D} } \sum_{(\vec{x}, y) \in D} (m(\vec{x}) - y)^2}$
emphasizes large distances / errors, same unit as the target values	
R^2 (R -squared)	coefficient of determination
	$f(m, \mathbf{D}) = 1 - \frac{\sum_{(\vec{x}, y) \in D} (m(\vec{x}) - y)^2}{\sum_{(\vec{x}, y) \in D} (y - \bar{y})^2}$
universal range $[0, 1]$, closer to 1 is better; \bar{y} is the mean of the target values: $y = \frac{1}{ \mathbf{D} } \sum_{(\vec{x}, y) \in D} y$	
mean absolute error (MAE)	$f(m, \mathbf{D}) = \frac{1}{ \mathbf{D} } \sum_{(\vec{x}, y) \in D} m(\vec{x}) - y $
equal weights for all distances / errors, same unit as the target values	
mean absolute percentage error (MAPE)	$f(m, \mathbf{D}) = \frac{100\%}{ \mathbf{D} } \sum_{(\vec{x}, y) \in D} \left \frac{m(\vec{x}) - y}{y} \right $
requires non-zero target values	
mean signed difference	$f(m, \mathbf{D}) = \frac{1}{ \mathbf{D} } \sum_{(\vec{x}, y) \in D} (m(\vec{x}) - y)$
only useful for detecting a bias of the error	

Classification Evaluation (nominal target)

- The most common loss function for classification is the **misclassification rate**

$$E(m, \mathbf{D}) = \frac{1}{|\mathbf{D}|} \sum_{(\vec{x}, y) \in \mathbf{D}} (1 - \delta_{m(\vec{x}), y}),$$

and (alternatively) its dual, the score function **accuracy**

$$A(m, \mathbf{D}) = \frac{1}{|\mathbf{D}|} \sum_{(\vec{x}, y) \in \mathbf{D}} \delta_{m(\vec{x}), y} = 1 - E(m, \mathbf{D}).$$

- A **confusion matrix** displays the misclassifications in more detail. It is a table in which the rows represent the true classes and the columns the predicted classes.
- Entries specify how many objects from the true class of the corresponding row are classified as belonging to the class of the corresponding column.
- The **accuracy** is the sum of the diagonal entries divided by the sum of all entries.
The **misclassification rate** (or simply **error rate**) is the sum of the off-diagonal entries divided by the sum of all entries.
- An ideal classifier has non-zero entries only on the diagonal.

What about Ordinal Target Attributes?

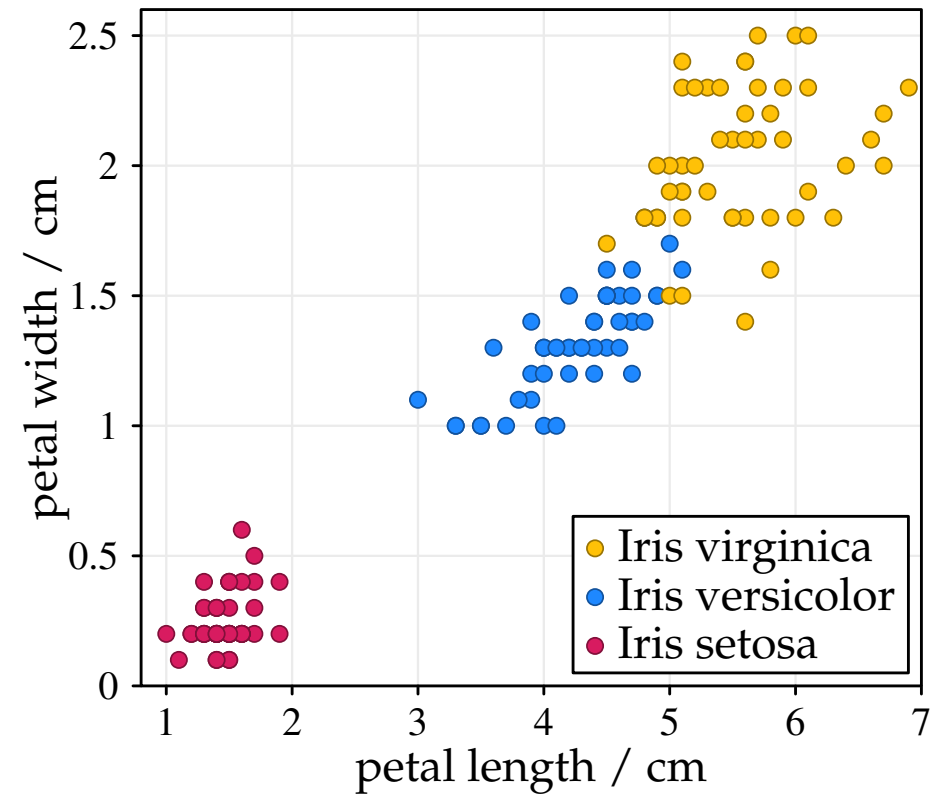
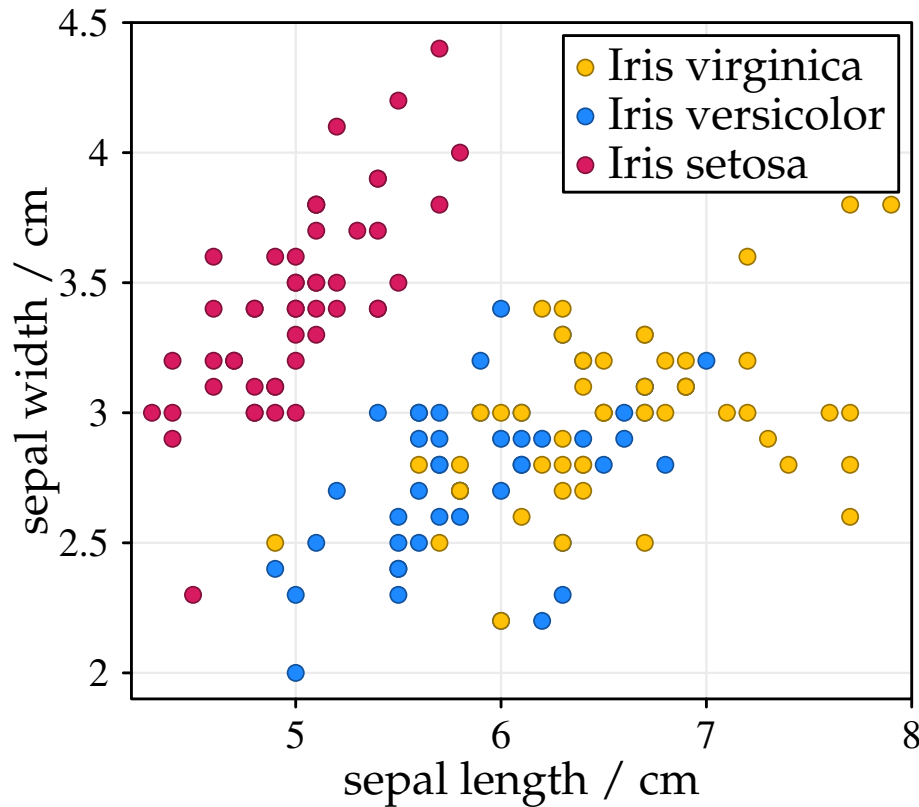
- There are only few approaches that can explicitly treat ordinal targets. (These approaches are not covered in this lecture.)
- Hence ordinal attributes often pose an unpleasant problem.
- Of course, one may treat ordinal targets like nominal targets, and this is actually what is most commonly done.
- However, this discards the ordinal structure of the domain of such a target. As a consequence, a lot of information is lost.
- On the other hand, if we represent the values of an ordinal target by numbers that reflect the ordering, it will be treated as if it were a metric target, implicitly assuming that at least differences can be computed meaningfully.
- Unfortunately there seems to be no perfect solution for this problem.
- One has to choose for the application at hand which drawback is easier to tolerate.

Reminder: The Iris Data

pictures not available in online version

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

Reminder: 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).

Classification Evaluation: Confusion Matrix

- The table below shows a possible confusion matrix for the Iris data set.

true class	predicted class		
	Iris setosa	Iris versicolor	Iris virginica
Iris setosa	50	0	0
Iris versicolor	0	47	3
Iris virginica	0	2	48

- From this matrix, we can see that all cases of the class Iris setosa are classified correctly and no case of another class is wrongly classified as Iris setosa.
- A few cases of the other classes are wrongly classified:
three cases of Iris versicolor are (wrongly) classified as Iris virginica,
two cases of Iris virginica are (wrongly) classified as Iris versicolor.
- The **misclassification rate** is $E = \frac{2+3}{50+47+3+2+48} = \frac{5}{150} \approx 3.33\%$.
- The **accuracy** is $A = \frac{50+47+48}{50+47+3+2+48} = \frac{145}{150} \approx 96.67\%$.

Classification Evaluation: Two Classes

- For many classification problems there are only two classes that the classifier is supposed to distinguish.
- Let us call the two classes *plus* (or *positive*) and *minus* (or *negative*).
- Then the classifier can make two different kinds of mistakes:
 - Cases of the class *minus* may be wrongly assigned to the class *plus*. These cases are called **false positives (fp)**.
 - Vice versa, cases of the class *plus* may be wrongly classified as *minus*. Such cases are called **false negatives (fn)**.
- The cases that are classified correctly are called **true positives (tp)** and **true negatives (tn)**, respectively.

• **error rate:**
$$E = \frac{fp + fn}{tp + fn + fp + tn}$$

• **accuracy:**
$$A = \frac{tp + tn}{tp + fn + fp + tn}$$

Confusion matrix:

true class	predicted class		
	plus	minus	
plus	tp	fn	p
minus	fp	tn	n

Classification Evaluation: Precision and Recall

- Sometimes one wants to capture not merely the overall classification accuracy, but how well the individual classes are recognized.
- Especially if the class distribution is skewed, that is, if there are large differences in the class frequencies, overall measures may give a wrong impression.
- For example, if in a two class problem
 - one class occurs in 98% of all cases,
 - while the other covers only the remaining 2%,a classifier that always predicts the first class reaches an impressive accuracy of 98% — without distinguishing between the classes at all.
- Such unpleasant situations are fairly common in practice, for example:
 - illnesses are (fortunately) rare and
 - replies to mailings are (unfortunately?) scarce.Hence: predict that everyone is healthy or a non-replier.
- However, such a classifier is useless to a physician or a product manager.

Classification Evaluation: Precision and Recall

- In such cases (skewed class distribution) higher error rates are usually accepted in exchange for a better coverage of the minority class.
- In order to allow for this, the following two measures may be used:

- **Precision:** $\pi = \frac{tp}{tp + fp}$

- **Recall:** $\rho = \frac{tp}{tp + fn}$

[Perry, Kent & Berry 1955]

- In other words:
precision is the ratio of true positives to all data points classified as positive;
recall is the ratio of true positives to all actually positive data points.
- In yet other words:
precision is the fraction of cases for which a positive classification is correct;
recall is the fraction of positive cases that is identified by the classifier.
- Precision and recall are usually complementary quantities:
higher precision may be obtained at the price of lower recall and *vice versa*.

Classification Evaluation: Precision and Recall

- Attention: Precision and recall **evaluate** the performance of a classifier **w.r.t. one class**, namely the *plus* or *positive* class!
- If the two classes are exchanged (*plus* becomes *minus* and *vice versa*), the values of precision and recall (usually) change, maybe even considerably.
- Example: 980 cases in one class and 20 cases in another class (very skewed!).
Consider a classifier that always predicts the majority class.

Majority is class *plus* (or *positive*):

true ↓	plus	minus	← pred.
plus	980	0	980
minus	20	0	20

$$\pi = \frac{980}{1000} = 98\% \quad \text{very good!}$$
$$\rho = \frac{980}{980} = 100\%$$

Minority is class *plus* (or *positive*):

true ↓	plus	minus	← pred.
plus	0	20	20
minus	0	980	980

$$\pi = \frac{0}{0} [= 0\%] \quad \text{horribly bad!}$$
$$\rho = \frac{0}{20} = 0\%$$

Classification Evaluation: Other Quantities I

- recall, **sensitivity**, hit rate, or true positive rate (TPR)

$$\text{TPR} = \frac{\text{tp}}{p} = \frac{\text{tp}}{\text{tp} + \text{fn}} = 1 - \text{FNR}$$

- **specificity**, selectivity or true negative rate (TNR)

$$\text{TNR} = \frac{\text{tn}}{n} = \frac{\text{tn}}{\text{tn} + \text{fp}} = 1 - \text{FPR}$$

- precision or positive predictive value (PPV)

$$\text{PPV} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

- negative predictive value (NPV)

$$\text{NPV} = \frac{\text{tn}}{\text{tn} + \text{fn}}$$

- miss rate or false negative rate (FNR)

$$\text{FNR} = \frac{\text{fn}}{p} = \frac{\text{fn}}{\text{fn} + \text{tp}} = 1 - \text{TPR}$$

Classification Evaluation: Other Quantities II

- fall-out or false positive rate (FPR)

$$\text{FPR} = \frac{\text{fp}}{n} = \frac{\text{fp}}{\text{fp} + \text{tn}} = 1 - \text{TNR}$$

- false discovery rate (FDR)

$$\text{FDR} = \frac{\text{fp}}{\text{fp} + \text{tp}} = 1 - \text{PPV}$$

- false omission rate (FOR)

$$\text{FOR} = \frac{\text{fn}}{\text{fn} + \text{tn}} = 1 - \text{NPV}$$

- accuracy (ACC)

$$\text{ACC} = \frac{\text{tp} + \text{tn}}{p + n} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{tn} + \text{fp} + \text{fn}}$$

- misclassification rate or error rate (ERR)

$$\text{ERR} = \frac{\text{fp} + \text{fn}}{p + n} = \frac{\text{fp} + \text{fn}}{\text{tp} + \text{tn} + \text{fp} + \text{fn}}$$

Classification Evaluation: F-Measure

- With precision and recall two numbers describe the quality of a classifier.
- A common way to combine them into one number is to compute the **F₁-measure** [Rijsbergen 1979], which is the harmonic mean of precision and recall:

$$F_1 = \frac{2}{\frac{1}{\pi} + \frac{1}{\rho}} = \frac{2\pi\rho}{\pi + \rho} = \frac{2tp}{2tp + fn + fp}.$$

In this formula precision and recall have the same weight.

- The **generalized F-measure** [Rijsbergen 1979] introduces a mixing parameter. It can be found in several different, but equivalent versions, for example:

$$F_\alpha = \frac{1}{\frac{\alpha}{\pi} + \frac{1-\alpha}{\rho}} = \frac{\pi\rho}{\alpha\rho + (1-\alpha)\pi}, \quad \alpha \in [0, 1],$$

or

$$F_\beta = \frac{1 + \beta^2}{\frac{1}{\pi} + \frac{\beta^2}{\rho}} = \frac{\pi\rho(1 + \beta^2)}{\rho + \beta^2\pi}, \quad \beta \in [0, \infty).$$

Obviously, the standard F_1 -measure results for $\alpha = \frac{1}{2}$ or $\beta = 1$, respectively.

Classification Evaluation: F-Measure

- The generalized **F-measure** is [Rijsbergen 1979]:

$$F_{\alpha} = \frac{\pi\rho}{\alpha\rho + (1 - \alpha)\pi}, \quad \alpha \in [0, 1], \quad F_{\beta} = \frac{\pi\rho(1 + \beta^2)}{\rho + \beta^2\pi}, \quad \beta \in [0, \infty).$$

- By choosing the mixing parameters α or β it can be controlled whether the focus should be more on precision or more on recall:
 - For $\alpha > \frac{1}{2}$ or $\beta > 1$ the focus is more on precision;
for $\alpha = 1$ or $\beta = 0$ we have $F_{\alpha} = F_{\beta} = \pi$.
 - For $\alpha < \frac{1}{2}$ or $\beta < 1$ the focus is more on recall;
for $\alpha = 0$ or $\beta \rightarrow \infty$ we have $F_{\alpha} = F_{\beta} = \rho$.
- However, this possibility is rarely used, presumably, because precision and recall are usually considered to be equally important.
- Note that precision and recall and thus the generalized F -measure as well as its special case, the F_1 -measure, focus on one class (namely the positive class). **Exchanging the two classes usually changes all of these measures.**

Classification Evaluation: More Than Two Classes

- The **misclassification rate** (or **error rate**) and the **accuracy** can be used regardless of the number of classes (whether only two or more).
- In contrast, **precision**, **recall** and **F-measure** are defined only for two classes.
- However, they can be generalized to more than two classes by computing them for each class separately and averaging the results.
- In this approach, each class in turn is seen as the positive class (*plus*) while all other classes together form the negative class (*minus*).
- **Macro-averaging** (1st possibility of averaging) [Sebastini 2002]

○ **precision:**
$$\pi_{\text{macro}} = \frac{1}{c} \sum_{k=1}^c \pi_k = \frac{1}{c} \sum_{k=1}^c \frac{\text{tp}^{(k)}}{\text{tp}^{(k)} + \text{fp}^{(k)}}$$

○ **recall:**
$$\rho_{\text{macro}} = \frac{1}{c} \sum_{k=1}^c \rho_k = \frac{1}{c} \sum_{k=1}^c \frac{\text{tp}^{(k)}}{\text{tp}^{(k)} + \text{fn}^{(k)}}$$

Here c is the number of classes. All classes have equal weight.

Classification Evaluation: More Than Two Classes

- **Class-weighted averaging** (2nd possibility of averaging)

- **precision:**
$$\pi_{\text{wgt}} = \sum_{k=1}^c \frac{n_k}{n} \pi_k = \frac{1}{n} \sum_{k=1}^c \frac{\text{tp}^{(k)} + \text{fn}^{(k)}}{\text{tp}^{(k)} + \text{fp}^{(k)}} \cdot \text{tp}^{(k)}$$

- **recall:**
$$\rho_{\text{wgt}} = \sum_{k=1}^c \frac{n_k}{n} \rho_k = \frac{1}{n} \sum_{k=1}^c \text{tp}^{(k)}$$

Here c is again the number of classes,
 n_k is the number of cases belonging to class k , $k = 1, \dots, c$, and
 n is the total number of cases, $n = \sum_{k=1}^c n_k$.

- While macro-averaging treats each class as having the same weight (thus ignoring the (possibly skewed) class frequencies) class-weighted averaging takes the class frequencies into account.
- Note that class-weighted average recall is equivalent to **accuracy**, since $\sum_{k=1}^c \text{tp}^{(k)}$ is simply the sum of the diagonal elements of the confusion matrix and n , the total number of cases, is the sum over all entries.

Classification Evaluation: More Than Two Classes

- **Micro-averaging** (3rd possibility of averaging) [Sebastini 2002]

- **precision:**
$$\pi_{\text{micro}} = \frac{\sum_{k=1}^c \text{tp}^{(k)}}{\sum_{k=1}^c (\text{tp}^{(k)} + \text{fp}^{(k)})} = \frac{1}{n} \sum_{k=1}^c \text{tp}^{(k)}$$

- **recall:**
$$\rho_{\text{micro}} = \frac{\sum_{k=1}^c \text{tp}^{(k)}}{\sum_{k=1}^c (\text{tp}^{(k)} + \text{fn}^{(k)})} = \frac{1}{n} \sum_{k=1}^c \text{tp}^{(k)}$$

Here c is again the number of classes and n is the total number of cases.
This averaging renders precision and recall identical and equal to **accuracy**.

- As a consequence, micro-averaging is not useful in this setting, but it may be useful, e.g., for averaging results over different data sets.
- For all different averaging approaches, the **F₁-measure** may be computed as the harmonic mean of (averaged) precision and recall.
- Alternatively, the F_1 -measure may be computed for each class separately and then averaged in analogy to the above methods.

Classification Evaluation: Misclassification Costs

- Misclassifications may also be handled via **misclassification costs**.
- Misclassification costs are specified in a matrix analogous to a confusion matrix, that is, as a table, the rows of which refer to the true class and the columns of which refer to the predicted class.
 - The diagonal of a misclassification cost matrix is zero (correct class).
 - The off-diagonal elements specify the costs of a specific misclassification: entry z_{ij} specifies the costs of misclassifying class i as class j .
- If a cost matrix $\mathbf{Z} = (z_{ij})_{1 \leq i, j \leq c}$ is given (like the one on the right), the **expected loss** is used as the objective function:

$$L(m, \mathbf{D}) = \sum_{i=1}^c \sum_{j=1}^c p_{ij} \cdot z_{ij}$$

Here p_{ij} is the (relative) frequency with which class i is misclassified as class j .

true class	predicted class			
	1	2	...	c
1	0	z_{12}	...	z_{1c}
2	z_{21}	0	...	z_{2c}
\vdots	\vdots	\vdots	\ddots	\vdots
c	z_{c1}	z_{c2}	...	0

(Note: generally $z_{ij} \neq z_{ji}$!)

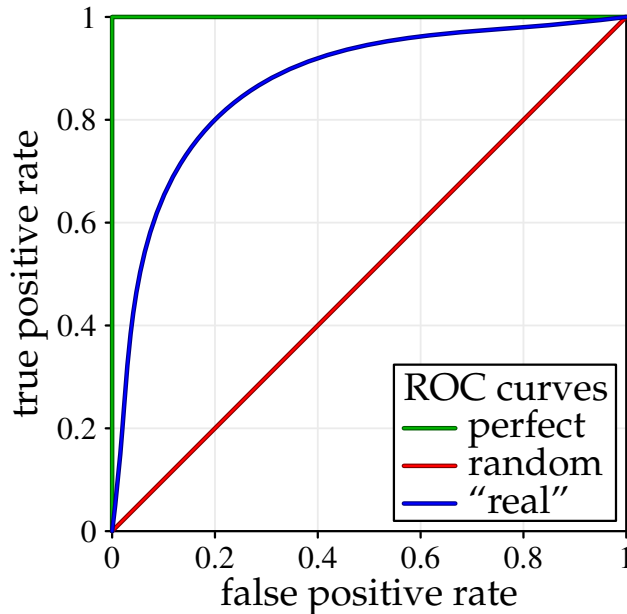
Classification Evaluation: Misclassification Costs

- Misclassification costs generalize the **misclassification rate**, which results as the special case of **equal misclassification costs**.
- With misclassification costs one can avoid the problems caused by **skewed class distributions**, because it can take into account that certain misclassifications can have more severe consequences or higher costs than others.
 - Misclassifying a sick patient as healthy has high costs (as this leaves the disease untreated).
 - Misclassifying a healthy patient as sick has low costs (although the patient may have to endure additional tests, it will finally be revealed that he/she is healthy).
 - Not sending an ad to a prospective buyer has high costs (because the seller loses the revenue from the sale).
 - Sending an ad to a non-buyer has low costs (only the cost of the mailing is lost, which may be very low).
- However, specifying proper costs can be tedious and time consuming.

Classification Evaluation: ROC Curves

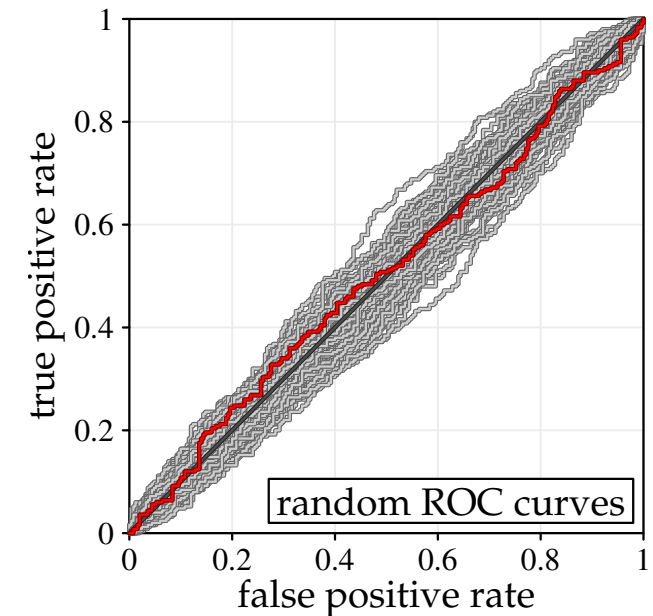
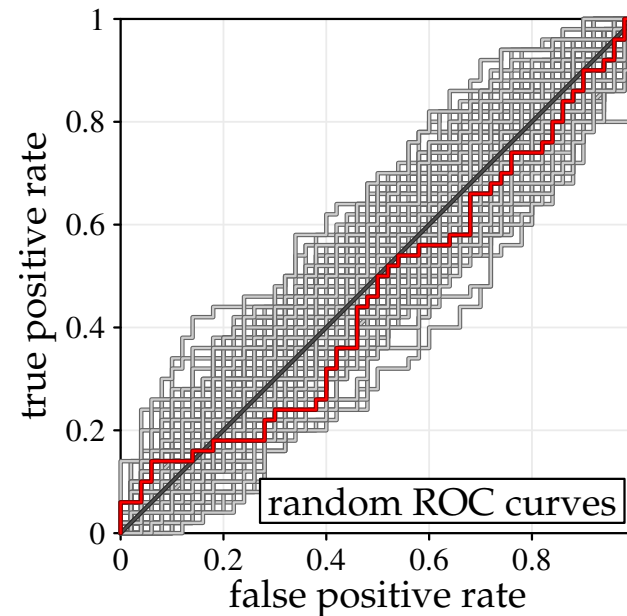
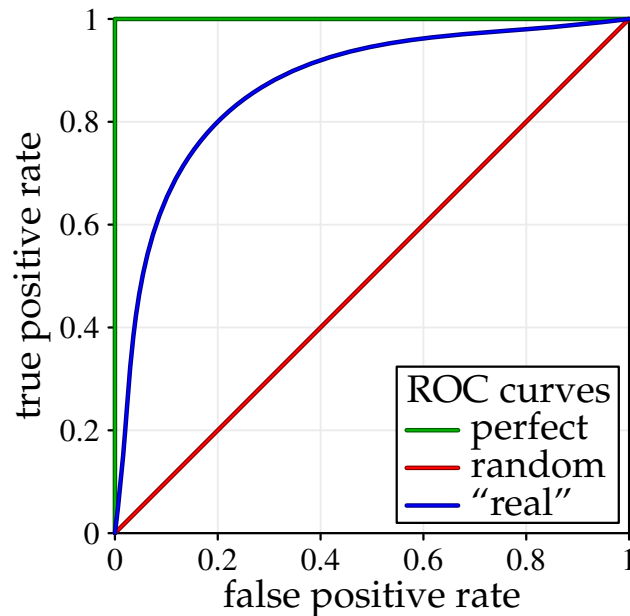
- Some classifiers (can) yield for every case to be classified a (formal) **probability** or **confidence** for each class.
- In such a case it is common to assign a case to the class for which the highest confidence / probability is produced (more later).
- In the case of two classes, *plus* and *minus*, one may assign a case to class *plus* if the probability for this class exceeds 0.5 and to class *minus* otherwise.
- However, one may also be more careful and assign a case to class *plus* only if the probability exceeds, e.g., $\tau = 0.8$, leading to fewer false positives.
- On the other hand, choosing a threshold $\tau < 0.5$ leads to more true positives.
- The trade-off between true positives and false positives is illustrated by the **receiver operating characteristic curve (ROC curve)** that shows the true positive rate versus the false positive rate.
- The **area under the (ROC) curve (AUC)** may be used as an indicator of how well a classifier solves a given problem.

Classification Evaluation: ROC Curves



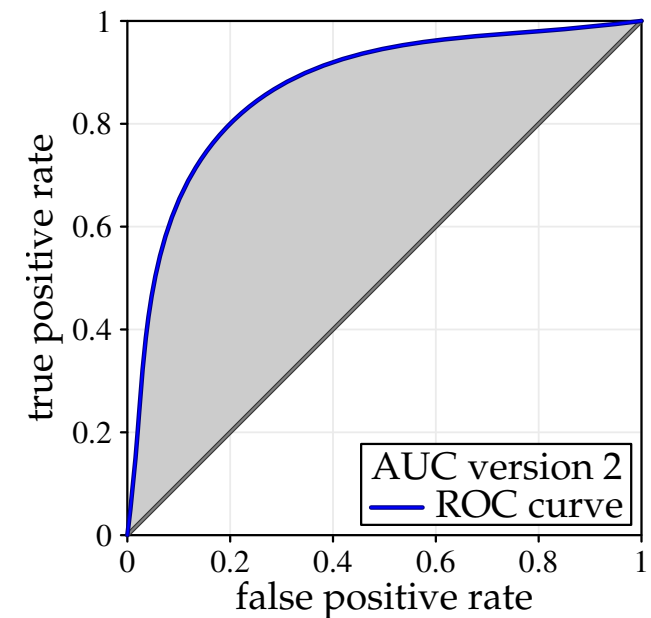
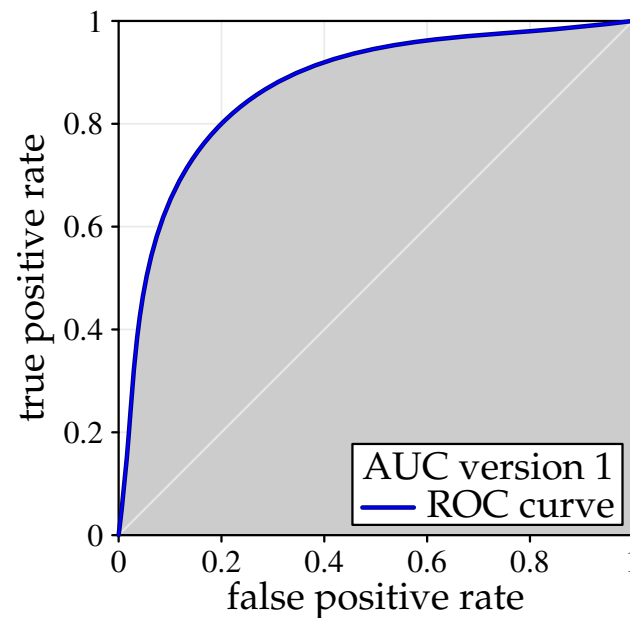
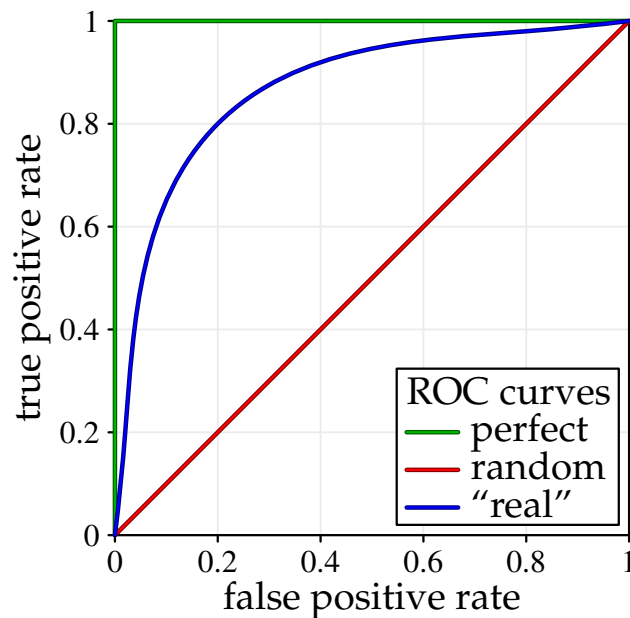
- **ROC curve:** For all choices of the threshold τ a new point is drawn at the respective coordinates of false positive rate and true positive rate.
 - These dots are connected to form a curve.
 - The curves on the left are idealized; actual ROC curves are (usually) step functions.
 - Diagonal segments may be used if the same confidence is assigned to cases of different classes.
-
- An ideal ROC curve (green) jumps to 100% true positives without producing any false positives, then adds the remaining cases as false positives.
 - A random classifier, which assigns random confidence values to the cases, is represented by the (idealized) red ROC curve ("expected" ROC curve).
 - An actual classifier may produce an ROC curve like the blue one.

Classification Evaluation: ROC Curves



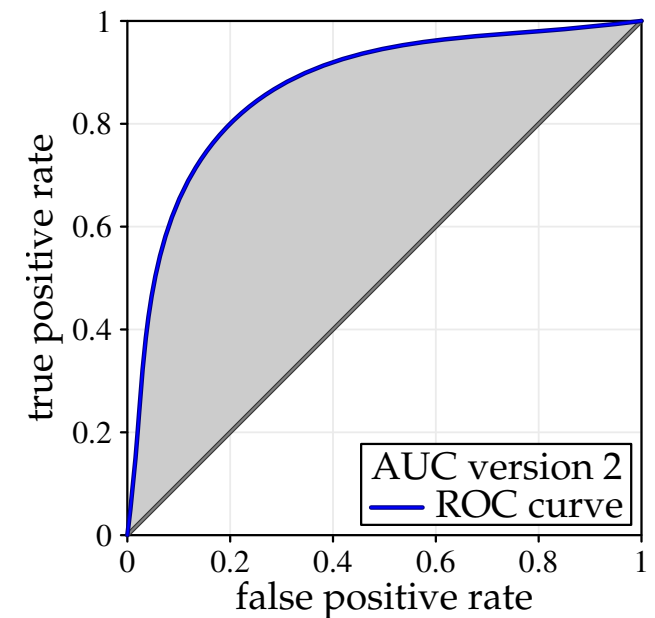
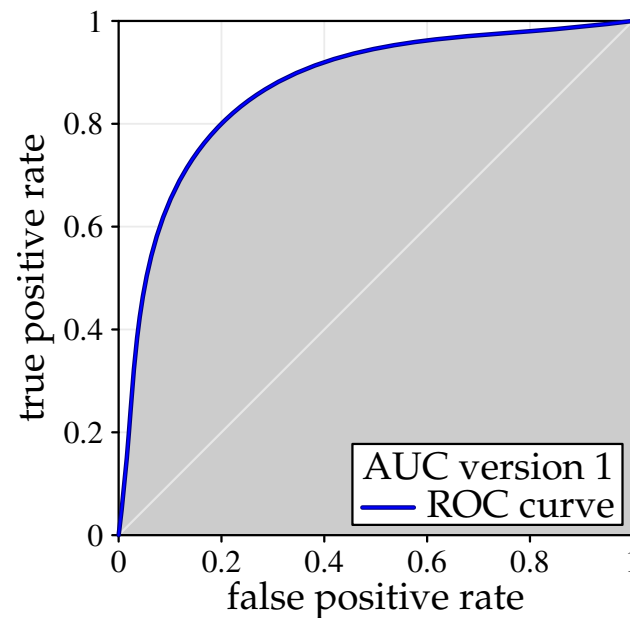
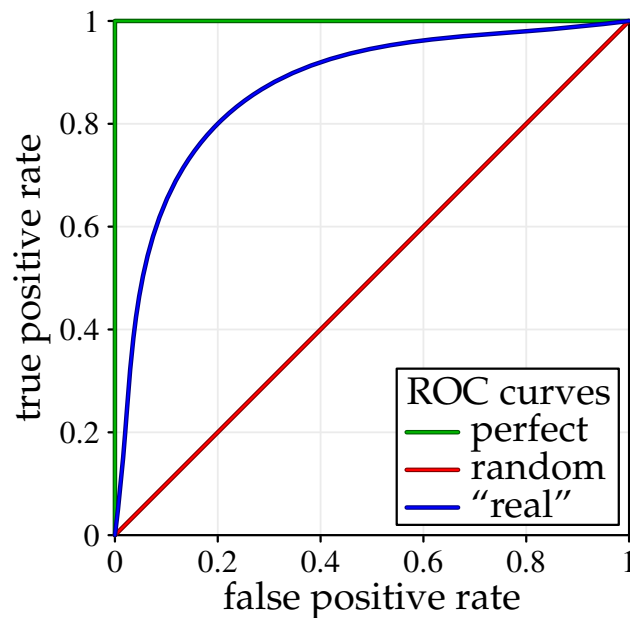
- A random classifier, which assigns random confidence values to the cases, is represented by the (idealized) red ROC curve ("expected" ROC curve).
- This line is idealized, because different random classifiers produce different ROC curves that scatter around this diagonal.
- middle: 50 positive, 50 negative cases; right: 250 positive, 250 negative cases; the diagrams show 100 random ROC curves each, with one highlighted in red.

Classification Evaluation: Area Under the (ROC) Curve



- The **Area Under the (ROC) Curve (AUC)** may be defined in two ways:
 - the area extends down to the horizontal axis (more common),
 - the area extends down to the diagonal and is doubled (more intuitive).
- It is $AUC_2 = 2(AUC_1 - \frac{1}{2})$ and $AUC_1 = \frac{1}{2}AUC_2 + \frac{1}{2}$.
- For a random ROC curve it is $AUC_1 \approx 0.5$, but $AUC_2 \approx 0$.
Note: AUC_2 may become negative, AUC_1 is always non-negative!

Classification Evaluation: Area Under the (ROC) Curve



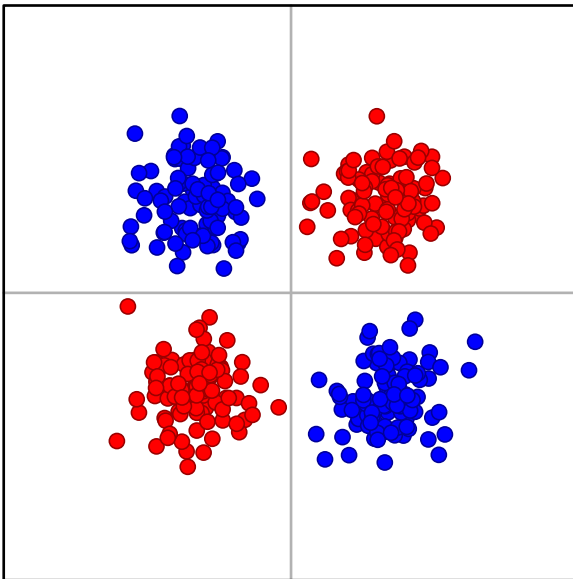
- The **Area Under the (ROC) Curve (AUC)** may be defined in two ways:
 - the area extends down to the horizontal axis (more common),
 - the area extends down to the diagonal and is doubled (more intuitive).
- The first version may also be fairly intuitively interpreted as the probability that a randomly chosen positive example is rated higher than a randomly chosen negative example.

Causes of Errors

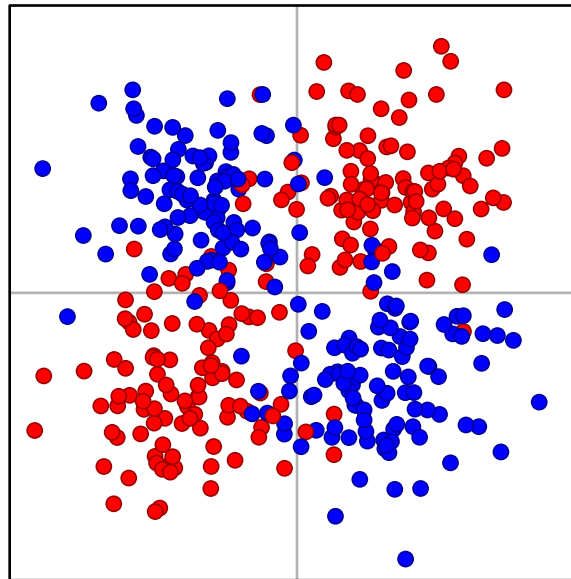
- One may distinguish between **four types of errors**:
 - the pure / intrinsic / experimental / Bayes error,
 - the sample error, variance error or scatter,
 - the lack of fit error, model error or bias error,
 - the algorithmic error.
- **Pure / Intrinsic / Experimental / Bayes Error**
 - Inherent in the data, impossible to overcome by any model.
 - Due to noise, random variations, imprecise measurements, or the influence of hidden (unobservable / unobserved) variables.
 - For any data points usually several classes (if classification) or numeric values (if numeric prediction) have a non-vanishing probability.
However, usually a predictor has to produce a single class / specific value; hence it cannot yield correct predictions all of the time.

Causes of Errors: Bayes Error

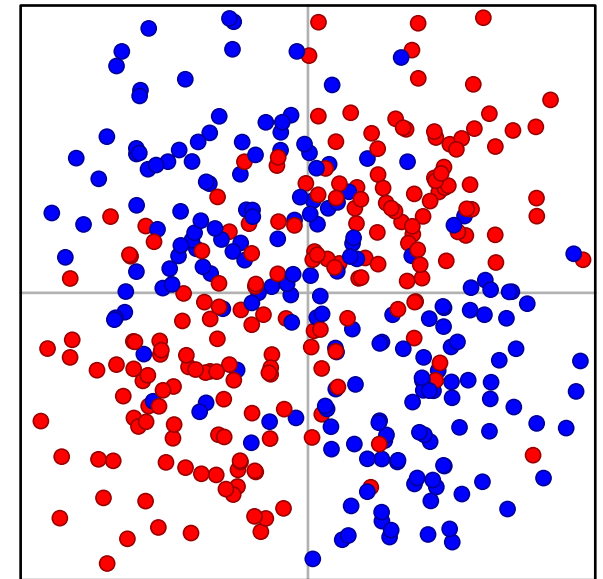
- The term “**Bayes error**” is usually used in the context of classification.
- It describes a situation, in which for any data point more than one class is possible (multiple classes have a non-vanishing probability).



perfect class separation



more difficult problem

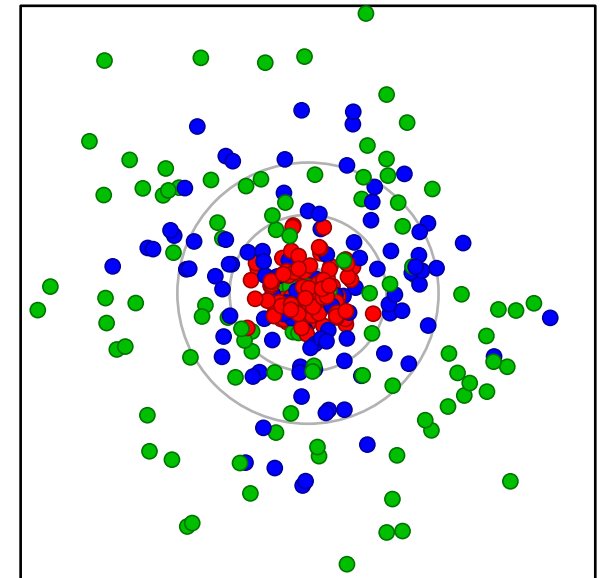


strongly overlapping
classes

- If classes overlap in the data space, no model can perfectly separate them.

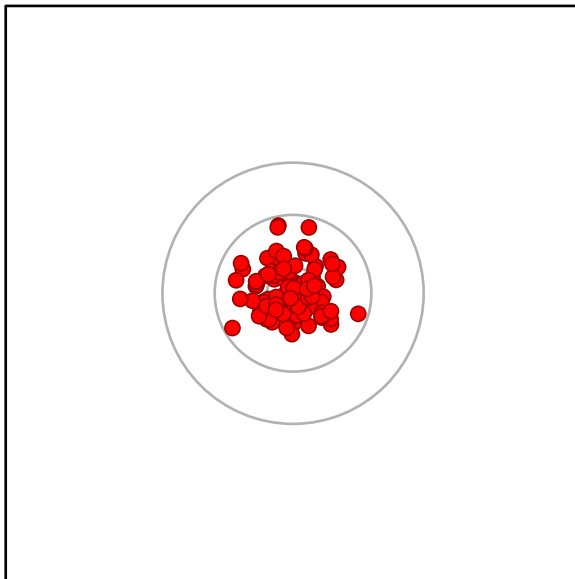
Causes of Errors: Bayes Error

- In the (artificial) example on the previous slide, the samples of each class are drawn from two bivariate normal distributions (four in total).
- In all three cases, the means of the distributions are the same, only the variances differ, leading to a greater overlap of the classes the greater the variances.
- However, this is not necessarily the only (relevant) situation. Classes may have similar means and rather differ in their variances.
- Example: Three darts players try to hit the center of the dartboard (the so-called *bull's eye*).
- Assume one of the players is a professional, one is a hobby player and one is an absolute beginner.
- Objective: Predict who has thrown the dart, given the point where the dart hit the dartboard.

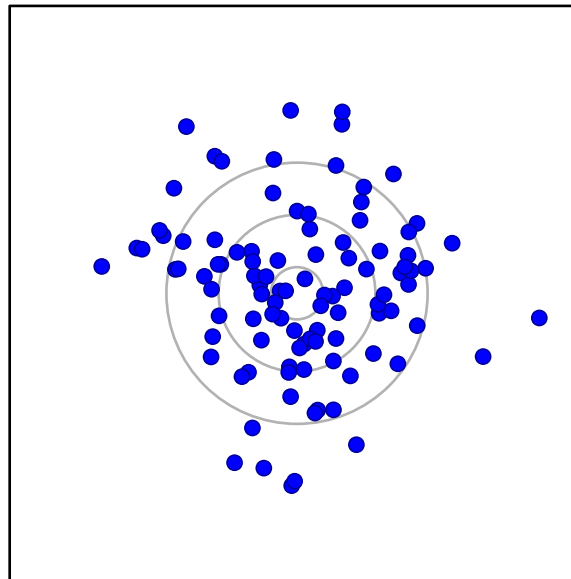


Causes of Errors: Bayes Error

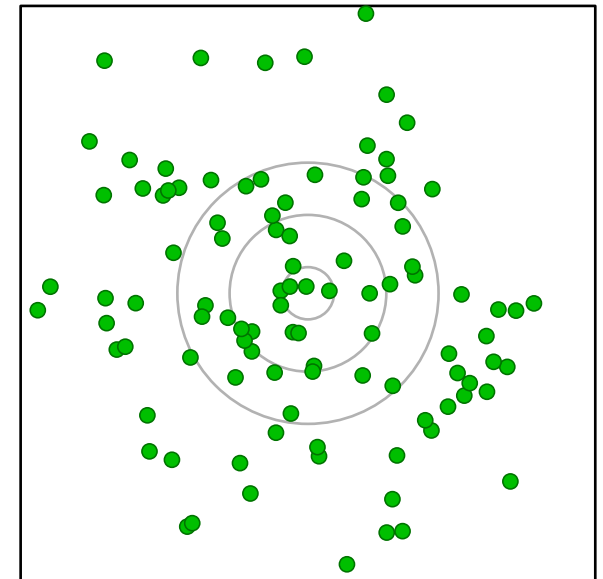
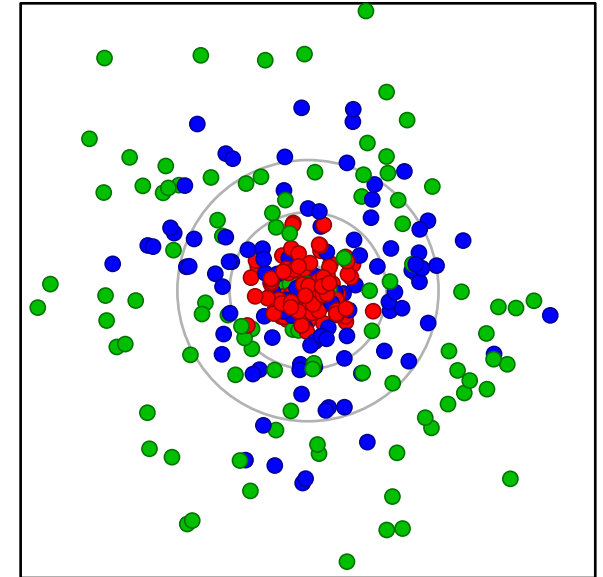
- Example: Three darts players try to hit the center of the dartboard (the so-called *bull's eye*).
- Assume one of the players is a professional, one is a hobby player and one is an absolute beginner.
- Objective: Predict who has thrown the dart, given the point where the dart hit the dartboard.



professional



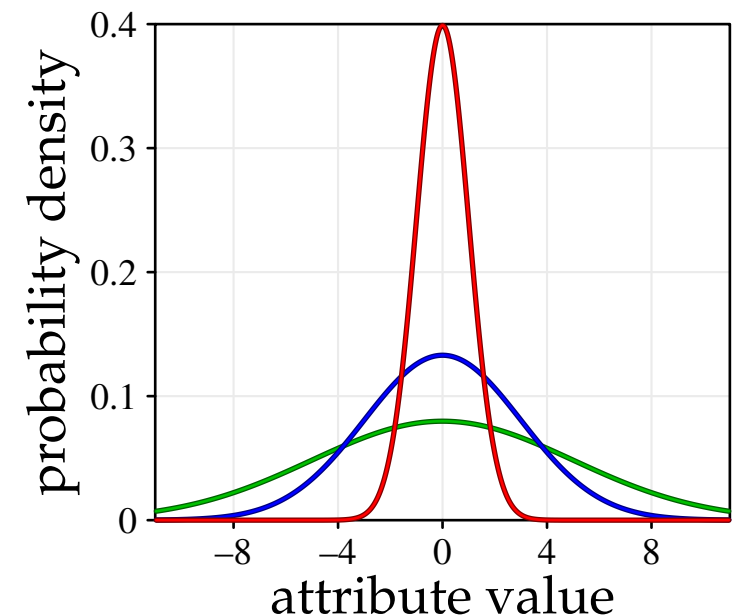
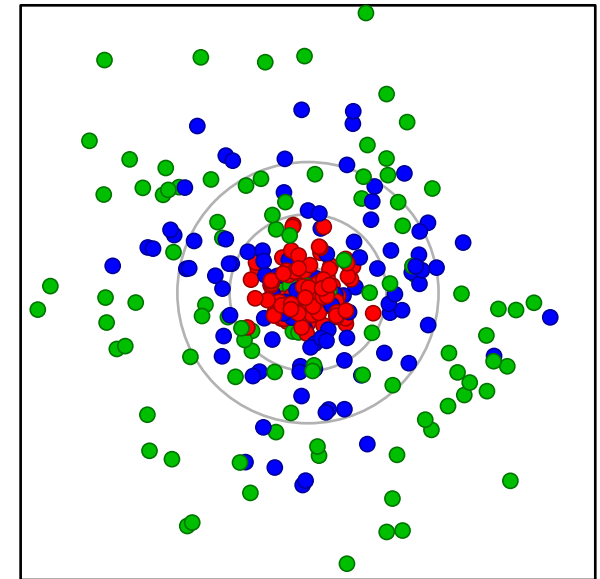
hobby player



beginner

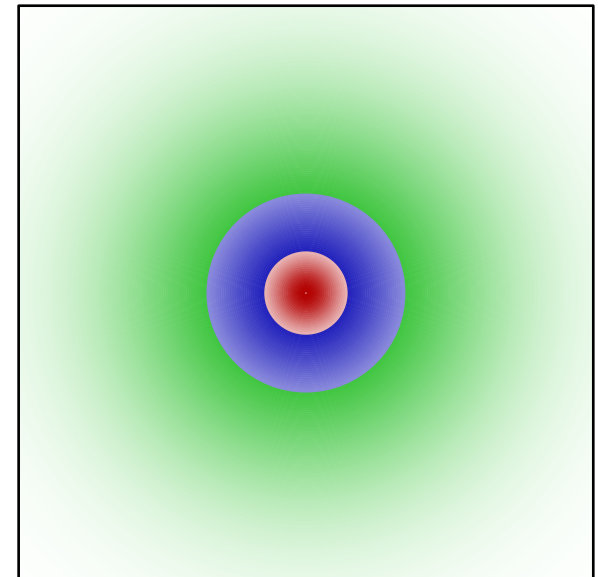
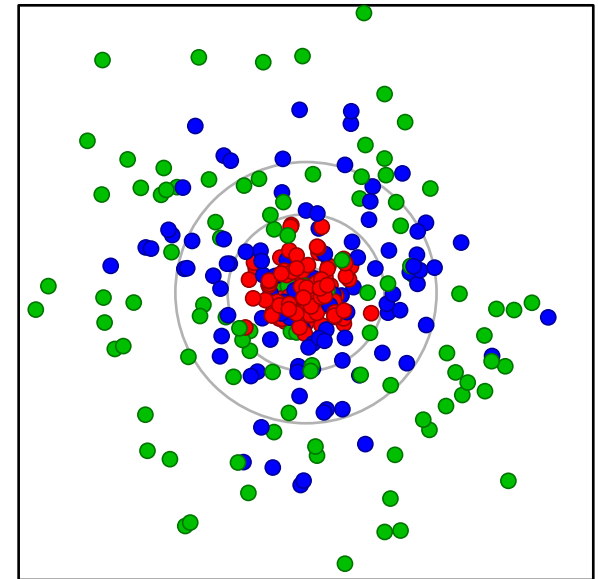
Causes of Errors: Bayes Error

- Example: Three darts players try to hit the center of the dartboard (the so-called *bull's eye*).
- Assume one of the players is a professional, one is a hobby player and one is an absolute beginner.
- Objective: Predict who has thrown the dart, given the point where the dart hit the dartboard.
- Simple classification rule: Assuming equal frequency of the three classes, assign the class with the highest likelihood. (one-dimensional normal distributions \Rightarrow)
- **Attention:** Do not confuse **classification boundaries** with **class boundaries** (which may not even exist).



Causes of Errors: Bayes Error

- Example: Three darts players try to hit the center of the dartboard (the so-called *bull's eye*).
- Assume one of the players is a professional, one is a hobby player and one is an absolute beginner.
- Objective: Predict who has thrown the dart, given the point where the dart hit the dartboard.
- Simple classification rule: Assuming equal frequency of the three classes, assign the class with the highest likelihood. (two-dimensional normal distributions \Rightarrow)
- **Attention:** Do not confuse **classification boundaries** with **class boundaries** (which may not even exist).

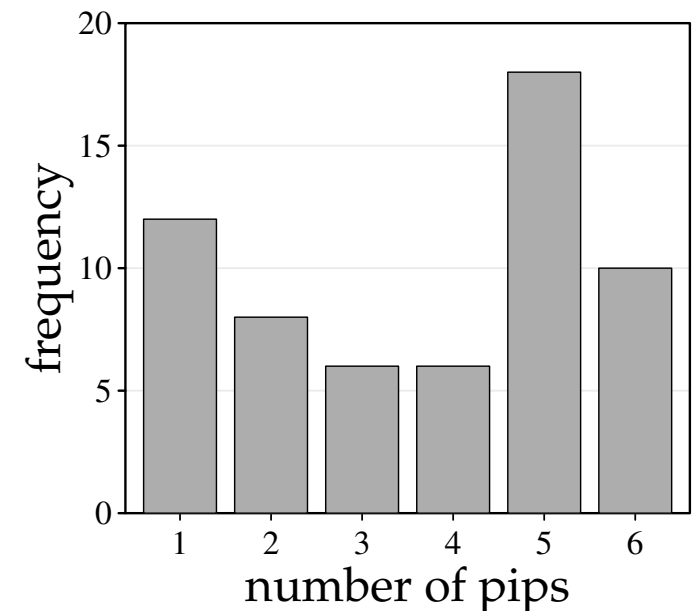


Causes of Errors: Sample Error

- **Sample Error, Variance Error or Scatter**

- The sample error is caused by the fact that the given data is only an imperfect representation of the underlying distribution.
- According to the laws of large numbers, the sample distribution converges in probability to the true distribution if the sample size approaches infinity.
- However, a finite sample can deviate significantly from the true distribution although the probability for such a deviation might be small.

- The bar chart on the right shows the result for throwing a fair die 60 times.
- In the ideal case, one would expect each of the numbers $1, \dots, 6$ to occur 10 times.
- But for this sample, the distribution does not look uniform.

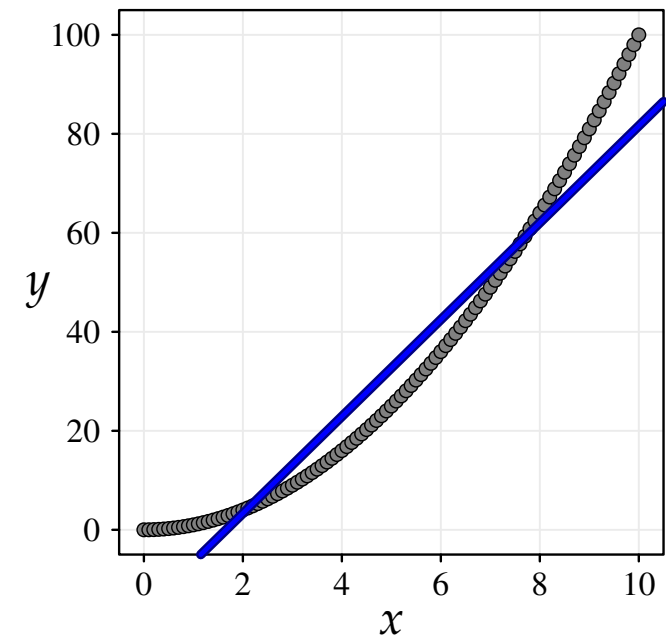


Causes of Errors: Sample Error

- Another source for sample errors are measurements with limited precision and round-off errors in features derived by computations.
(Remark: Precise and imprecise numbers, Harry Kemelman: “The Nine Mile Walk”, 1967: “Remember, it’s a *nine* mile walk and nine is one of the exact numbers.”)
- Sometimes the sample is also (systematically) **biased** or **censored**.
 - Consider a bank that supplies loans to customers.
 - Based on historical data available on customers who have obtained loans, the bank wants to estimate a new customer’s credit-worthiness (i.e., the probability that the customer will repay a loan).
 - The collected data will be biased towards better customers, because customers with a more problematic financial status have not been granted loans.
 - Therefore, no information is available for such customers whether they might have paid back the loan nevertheless.
 - In statistical terms: the sample is not representative, but biased.

Causes of Errors: Model Error

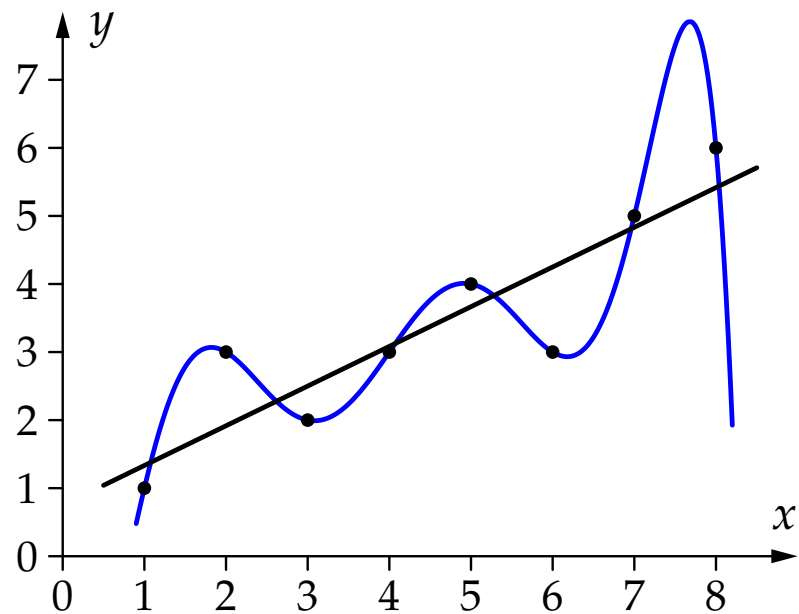
- A large error might be caused by a high pure error, but it might also be due to a **lack of fit**.
- If the set of considered models is too simple for the structure in the data, no model (from this set) will yield a small error.
- Such an error is also called **model error** or **bias error**.
(Because an improper choice of the model class introduces a bias into the fit.)
- The chart on the right shows a regression line fitted to data with no pure and no sample error.
- However, the data points originate from a quadratic and not from a linear relationship.
- As a consequence, there is a considerable lack of fit.



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.

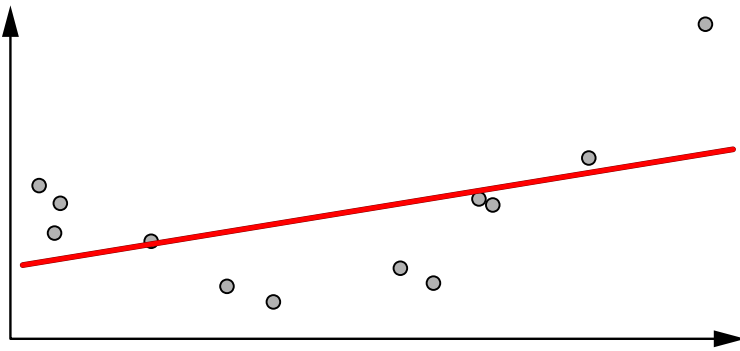


black line:
regression line
(2 free parameters)

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

- The blue curve fits the data points perfectly, *but it is not a good model.*
- On the other hand, too simple a model can lead to a lack of fit.

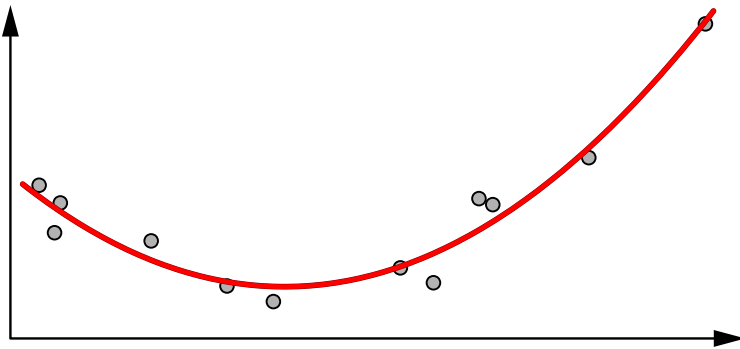
Model Capacity: Under- and Overfitting



Underfitting

[here: linear]

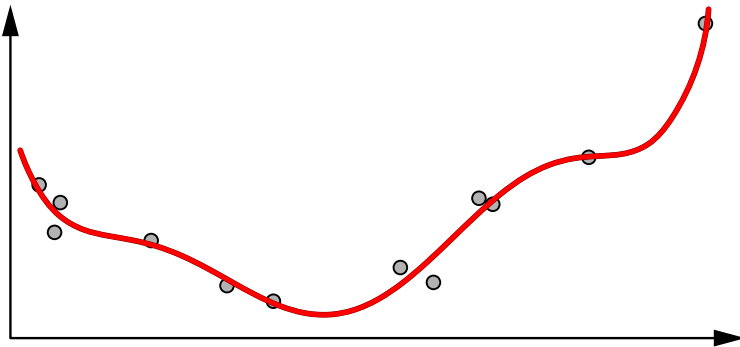
- Caused by model error / lack of fit.
- Model has not enough capacity to fit the regularities in the data.



(Proper) Fitting

[here: quadratic]

- Model has the proper capacity to fit regularities, but not enough to fit accidental properties.

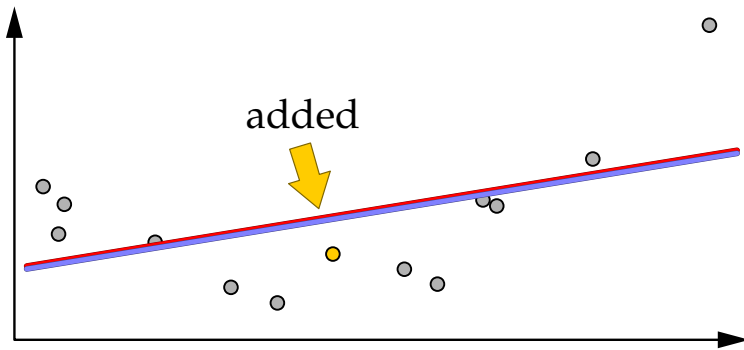


Overfitting

[here: degree 6]

- Caused by pure & sample error and ...
- ... model has too much capacity and thus fits not only regularities, but also accidental properties.

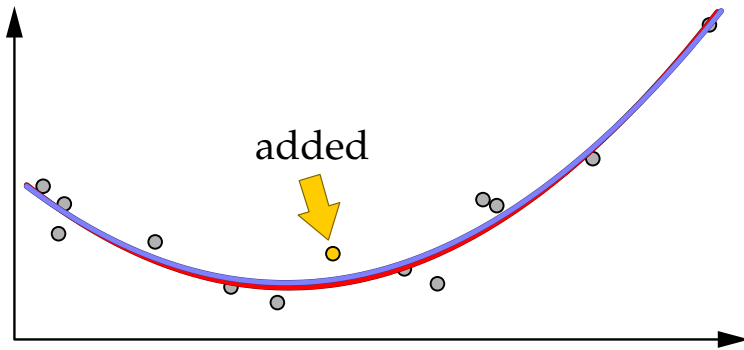
Model Capacity: Under- and Overfitting



Underfitting

[here: linear]

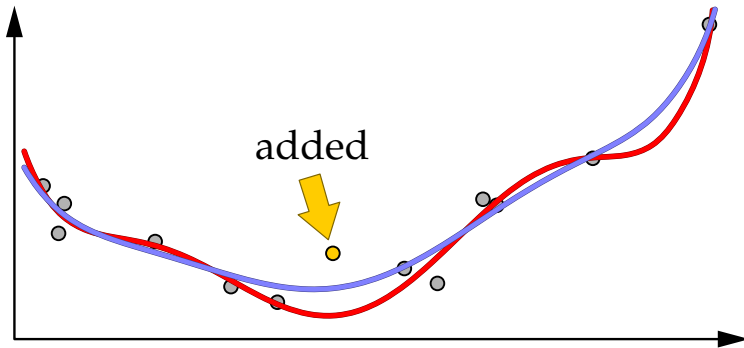
- Caused by model error / lack of fit.
- Model has not enough capacity to fit the regularities in the data.



(Proper) Fitting

[here: quadratic]

- Model has the proper capacity to fit regularities, but not enough to fit accidental properties.



Overfitting

[here: degree 6]

- Caused by pure & sample error and ...
- ... model has too much capacity and thus fits not only regularities, but also accidental properties.

Causes of Errors: Algorithmic Error

- The **algorithmic error** is caused by the method that is used to fit the model or the model parameters.
- In the ideal case, if an analytical solution for the optimum of the objective function exists, the algorithmic error is zero or only caused by numerical problems.
- However, in many cases an analytical solution cannot be provided and heuristic strategies are needed to fit the model to the data.
- Even if a model exists with a very good fit—the global optimum of the objective function—the heuristic optimization strategy might only be able to find a local optimum with a much larger error.
- This error is neither caused by the pure error nor by the sample error nor by the error due to the lack of fit (model error).
- Most of the time, the algorithmic error will not be considered and it is assumed that the heuristic optimization strategy is chosen well enough to find an optimum that is at least close to the global optimum.

Machine Learning Bias and Variance

- The four types of errors can be grouped into two categories.
- The *algorithmic* and the *model error* can be controlled to a certain extent, since we are free to choose a suitable model and algorithm.

These errors are also called **machine learning bias**.

- On the other hand, we have no influence on the *pure / intrinsic error* or the *sample error* (at least if the data to be analyzed have already been collected).

These errors are also called **machine learning variance**.

- Note that this decomposition *differs* from the one commonly known in statistics, where, for example, the mean squared error of an estimator $\hat{\theta}$ for a parameter θ can be decomposed in terms of the variance of the estimator and its bias:

$$\text{MSE} = \text{Var}(\hat{\theta}) + (\text{Bias}(\hat{\theta}))^2.$$

Here the variance depends on the intrinsic error, i.e. on the variance of the random variable from which the sample is generated, but also on the choice of the estimator $\hat{\theta}$ which is considered part of the model bias in machine learning.

Learning Without Bias? No Free Lunch Theorem

- The different types of errors or biases have an interesting additional impact on the ability to find a suitable model for a given data set:

If we have no model bias, we will not be able to generalize.

- The model bias is actually essential to put some sort of a-priori knowledge into the model learning process.
- Essentially this means that we need to constrain
 - either the types of models that are available
 - or the way we are searching for a suitable model (or both).
- The technical reason for this need is the **No Free Lunch Theorem**.
[Wolpert and MacReady 1997]
- Intuitively, this theorem states that if an algorithm (e.g. a machine learning or optimization algorithm) performs well on a certain class of problems, then it must pay for that with degraded performance on the set of remaining problems.

Algorithms for Model Fitting

- An objective function (score / loss function) does not tell us directly how to find the best or at least a good model.
- To find a good model, an **optimization method** is needed (i.e., a method that optimizes the objective function).
- Typical examples of optimization methods are
 - **Analytical / Closed Form Solutions**
Sometimes a solution can be obtained in closed form (e.g. linear regression).
 - **Combinatorial Optimization**
If the model space is (very) small, *exhaustive search* may be feasible.
 - **Gradient Methods**
If the objective function is differentiable, a gradient method (gradient ascent or descent) may be applied to find a (possibly only local) optimum.
 - **Random Search, Greedy Strategies and Other Heuristics**
For example, *hill climbing, greedy search, alternating optimization, widening, evolutionary* and *swarm-based algorithms* etc.

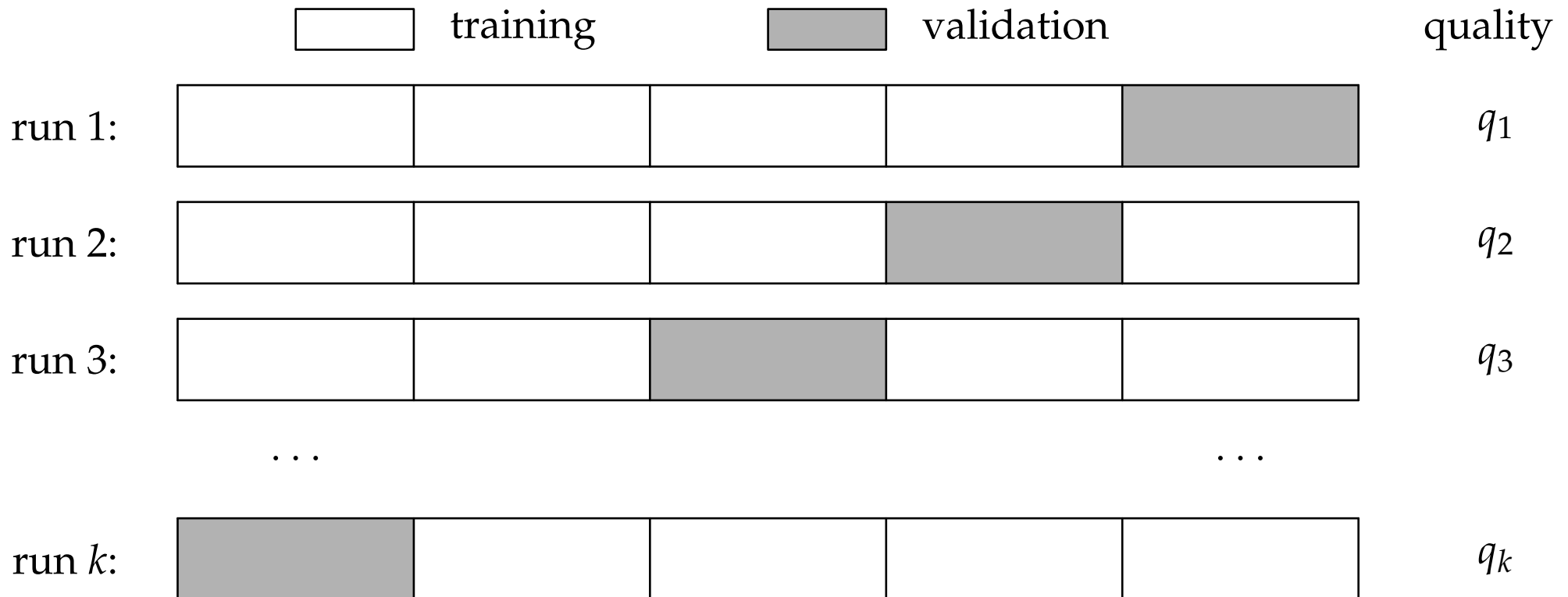
Model Validation

- Due to possible **overfitting** to the induction / training data (i.e. adaptations to features that are not regular, but accidental), the error on the training data is not very indicative of the error on new data.
- General idea of model validation:
Evaluate on a **hold-out data set (validation data)**, that is, on data **not** used for building / training the model.
 - Split the data into two parts: **training data** and **validation data** (often recommended: training data 80%, validation data 20%).
 - Train a model on the training data and evaluate it on the validation data.
- It is (highly) unlikely that the validation data exhibits the same accidental features as the training data.
- However, we might be lucky (unlucky) that the validation data contains easy (difficult) examples leading to an over-optimistic (-pessimistic) evaluation.
- Solution approach: repeat the split, the training and the evaluation.

Model Validation: Cross-Validation

- General method to assess / to predict the performance of models created by a given model building procedure (*not* of a specific model!).
- Serves the purpose to **estimate the error (rate) on new example cases**.
- Procedure of cross-validation:
 - Split the given data set into k so-called *folds* of equal size (**k-fold cross-validation**). Often recommended: $k = 10$.
 - Combine $k - 1$ folds into a training data set, build a classifier, and test it on the k -th fold (the *hold-out fold*).
 - Do this for all k possible selections of $k - 1$ folds and average the error (rates) (or some other quality measure).
- Special case: **leave-1-out cross-validation** (also known as **jackknife method**). (use as many folds as there are example cases)
- The final classifier is generated from the full data set (in order to exploit all available information).

Model Validation: Cross-Validation



Build and evaluate a model in each run, then average their quality.

$$\hat{q} = \frac{1}{k} \sum_{i=1}^k q_i$$

- **Cross-validation does not assess a single model** (because k models are built). It assesses the expected performance of models built with a certain procedure.

Cross-Validation: Stratified and Grouped Splits

- **Stratified Split / Stratification** (from the Latin *stratum*: layer, level, tier)
 - If the target is nominal, the split should maintain in the created folds the relative frequency of the target values in the whole data set.
 - Shuffle the examples of the data set, then sort them w.r.t. the target attribute.
 - 1st fold: $[d_0, d_k, d_{2k}, \dots]$
 - 2nd fold: $[d_1, d_{k+1}, d_{2k+1}, \dots]$
 - ...
 - kth fold: $[d_{k-1}, d_{2k-1}, d_{3k-1}, \dots]$

In this way the frequencies of the target values differ by at most 1 between folds.
- **Grouped Split** (sample cases are not independent)
 - In some situations certain groups of sample cases should not be separated (e.g. different episodes of illness of the same patient).
 - In this case the split has to be done on groups, assigning them to folds, rather than on individual sample cases. (possible problem: fold sizes)
- **Stratified Grouped Splits** can be very tricky. (NP-hard in general)

Model Validation: Cross-Validation

- Cross-validation is also a method that may be used to determine good so-called **hyper-parameters** of a model building method.
- Distinction between *parameters* and *hyper-parameters*:
 - **parameter** refers to parameters of a model as it is produced by an algorithm, for example, regression coefficients;
 - **hyper-parameter** refers to the parameters of a model-building method, for example, the maximum height of a decision tree, the number of trees in a random forest, the number of neurons in a neural network etc.
- Hyper-parameters are commonly chosen by running a cross-validation for various choices of the hyper-parameter(s) and finally choosing the one that produced the best models (in terms of their evaluation on the validation data sets).
- A final model is built using the found values for the hyper-parameters on the whole data set (to maximize the exploitation of information).

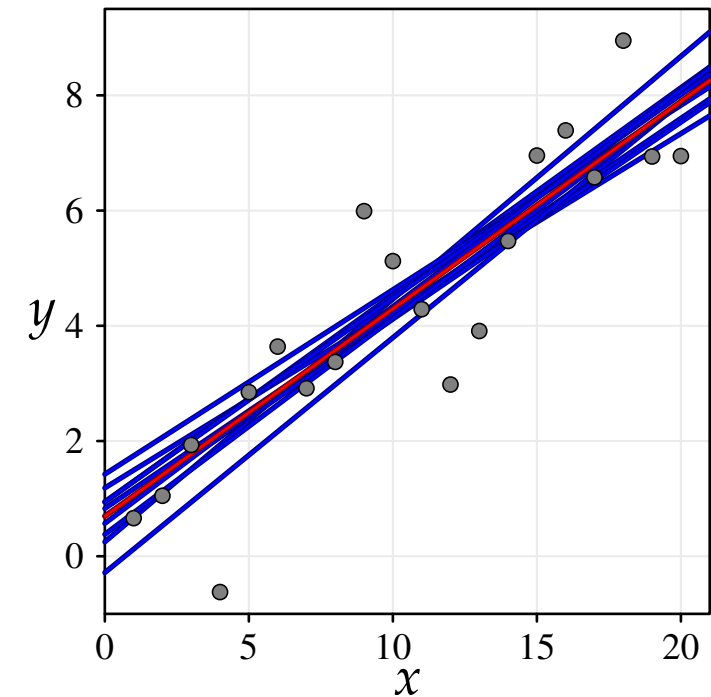
Model Validation: Bootstrapping

- **Bootstrapping** is a resampling technique from statistics that does not directly evaluate the model error, but aims at estimating the variance of the estimated model parameters.
- Therefore, bootstrapping is suitable for models with real-valued parameters.
- Like in cross-validation the model is computed multiple times.
- For this purpose, k bootstrap samples, each of size n , are drawn randomly *with replacement* from an original data set with n records.
- A model is fitted to each of these bootstrap samples, so that we obtain k estimates for the model parameters.
- Based on these k estimates the empirical standard deviation can be computed for each parameter to provide an assessment of how reliable its estimation is.
- It is also possible to compute confidence intervals for the parameters based on bootstrapping.

Model Validation: Bootstrapping

- The figure on the right shows a data set with $n = 20$ data points from which $k = 10$ bootstrap samples were drawn.
- For each of the bootstrap samples the corresponding regression line is shown.

sample	intercept	slope
1	0.3801791	0.3749113
2	0.5705601	0.3763055
3	-0.2840765	0.4078726
4	0.9466432	0.3532497
5	1.4240513	0.3201722
6	0.9386061	0.3596913
7	0.6992394	0.3417433
8	0.8300100	0.3385122
9	1.1859194	0.3075218
10	0.2496341	0.4213876
mean	0.6940766	0.3601367
std. dev.	0.4927206	0.0361004



- The resulting parameter estimates for the intercept and the slope of the regression line are listed in the table on the left.
- The standard deviation for the slope is much lower than for the intercept, so that the estimation for the slope is more reliable.

k-Nearest Neighbors

k-Nearest Neighbors

- **Basic Principle and Simple Examples**
- **Ingredients of k-Nearest Neighbor Methods**
 - Distance Metric
 - Number of Neighbors
 - Weighting Function for the Neighbors
 - Prediction Function
- **Weighting with Kernel Functions**
- **Locally Weighted Polynomial Regression**
- **Implementation Aspects**
- **Feature / Attribute Weights**
- **Data Set Reduction and Prototype Building**
- **Summary**

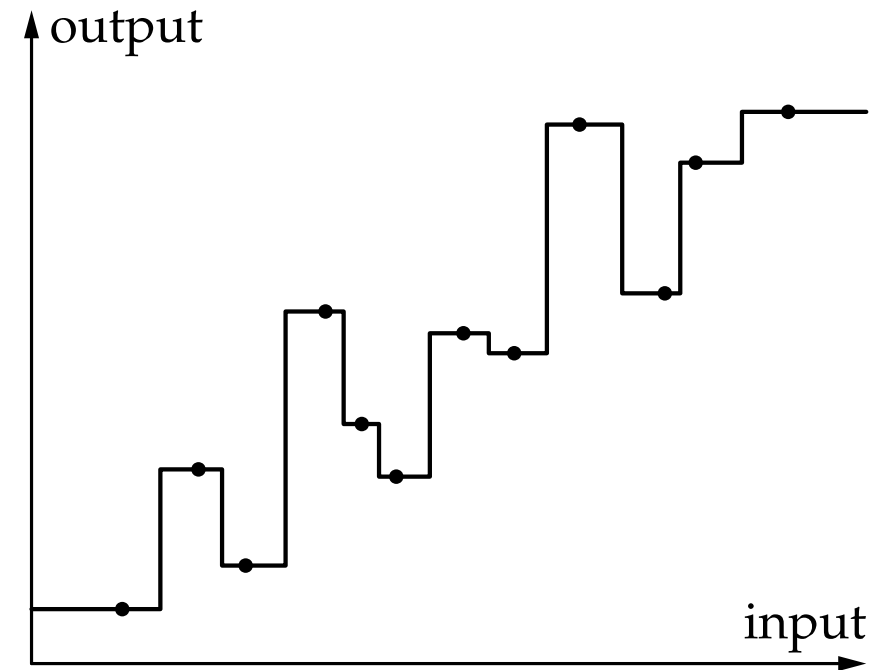
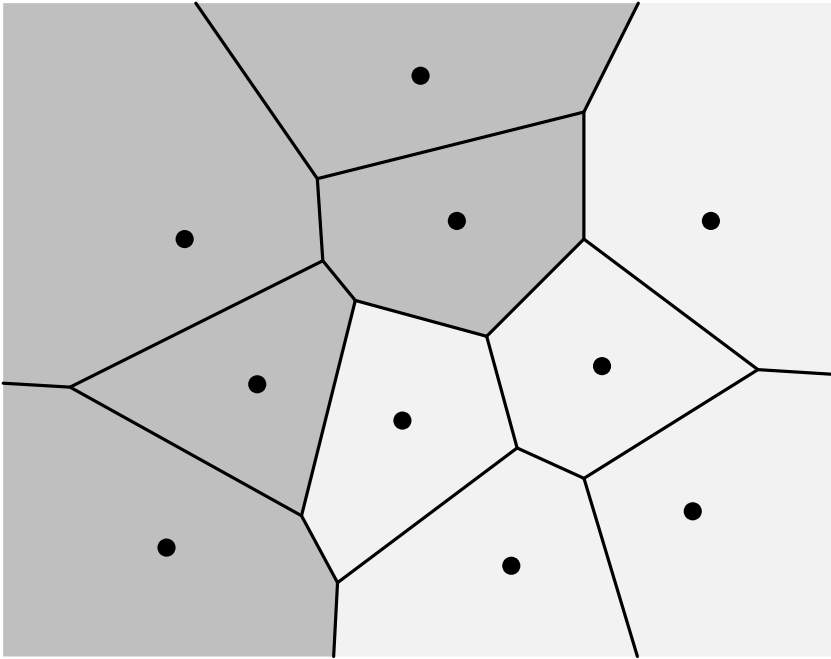
k-Nearest Neighbors: Principle

- The nearest neighbor algorithm [Cover and Hart 1967] is one of the simplest and most natural classification and numeric prediction methods.
- It derives the class labels or the (numeric) target values of new input objects from the **most similar training examples**, where similarity is measured by distance in the feature space.
- The prediction is computed by a **majority vote** of the nearest neighbors or by **averaging** their (numeric) target values.
- The number k of neighbors to be taken into account is a parameter of the algorithm, the best choice of which depends on the data and the prediction task.
- In a basic nearest neighbor approach only one neighbor object, namely the closest one, is considered, and its class or target value is directly transferred to the query object.

k-Nearest Neighbors: Principle

- Constructing nearest neighbor classifiers and numeric predictors is a special case of **case- or instance-based learning** [Aha *et al.* 1991].
- As such, it is a **lazy learning** method in the sense that it is **not** tried to construct a model that generalizes beyond the training data (as **eager learning** methods do).
- Rather, the training examples are merely stored.
- Predictions for new cases are derived directly from these stored examples and their (known) classes or target values, usually without any intermediate model construction.
- (Partial) Exception: **lazy decision trees** construct from the stored cases the single path in the decision tree along which the query object is passed down.
- This can improve on standard decision trees in the presence of missing values.
- However, this comes at the price of **higher classification costs**.

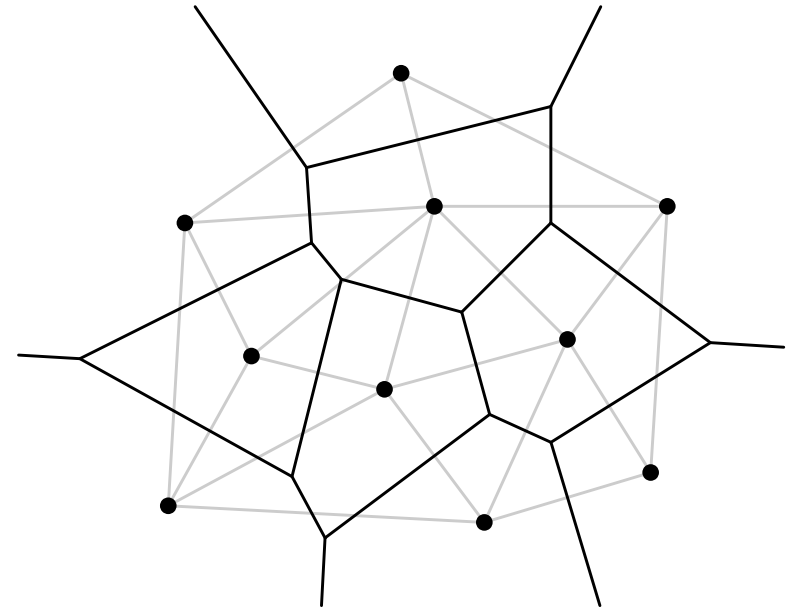
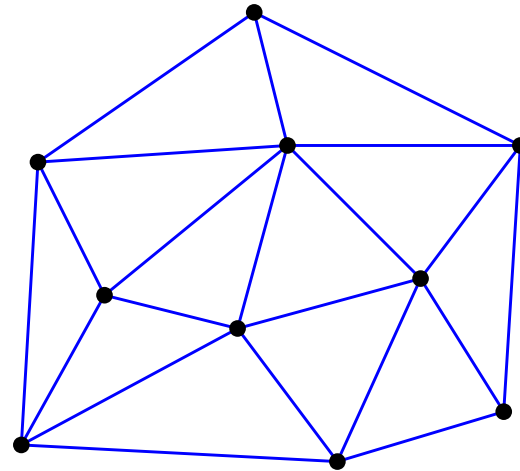
k-Nearest Neighbors: Simple Examples



- In both example cases it is $k = 1$.
- Classification works with a Voronoi tessellation of the data space.
- Numeric prediction leads to a piecewise constant function.
- Using more than one neighbor changes the classification/prediction.

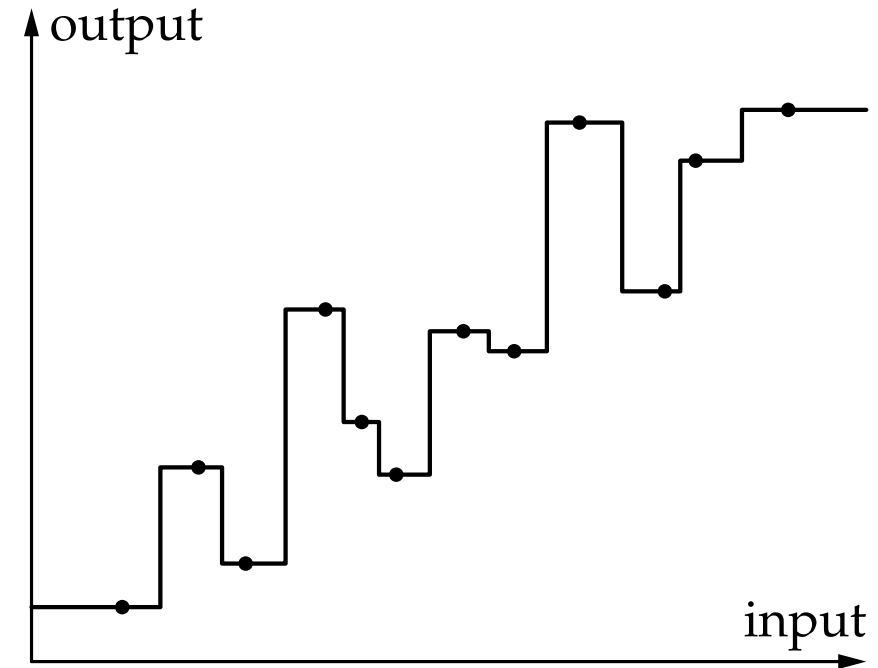
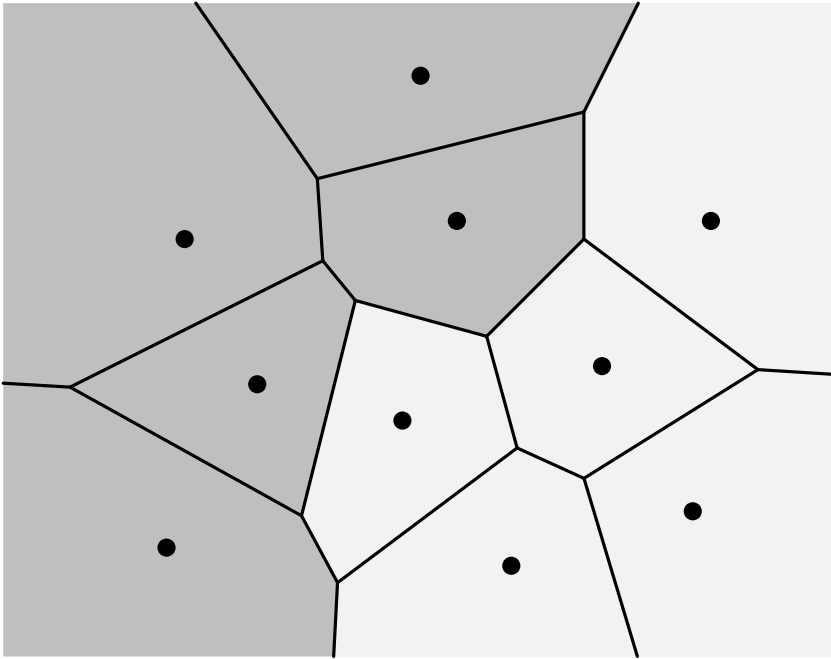
Delaunay Triangulations and Voronoi Diagrams

Dots represent
data points



- **Triangulation:** Maximum selection of connecting edges without intersections.
- **Left: Delaunay Triangulation**
The circle through the corners of a triangle does not contain another point.
- **Right: Voronoi Diagram / Tessellation**
Midperpendiculars of the Delaunay triangulation: boundaries of the regions of points that are closest to the enclosed data points (Voronoi cells).

k-nearest Neighbors: Simple Examples



- Note: neither the Voronoi tessellation nor the piecewise constant function are actually computed in the learning process; **no model is built at training time.**
- The prediction is determined only in response to a query for the class or target value of a new input object, namely by finding the closest neighbor of the query object and then transferring its class or target value.

Using More Than One Neighbor

- A straightforward generalization of the nearest neighbor approach is to use not just the one closest, but the **k nearest neighbors** (usually abbreviated as **k-NN**).
- If the task is classification, the prediction is then determined by a **majority vote** among these k neighbors (breaking ties arbitrarily).
- If the task is numeric prediction, the **average** of the target values of these k neighbors is computed.
- Not surprisingly, using more than one neighbor improves the robustness of the algorithm, since it is not so easily fooled by individual training instances that are labeled incorrectly or are outliers for a class (i.e. data points that have an unusual location for the class assigned to them).
- Outliers for the complete data set, on the other hand, do not affect nearest neighbor predictors much, because they can only change the prediction for data points that should not occur or should occur only very rarely (provided the rest of the data is representative).

Using More Than One Neighbor

- However, using too many neighbors can reduce the capability of the algorithm as it may smooth the classification boundaries or the interpolation too much to yield good results.
(Extreme example: Use all available training data points as neighbors.)
- As a consequence, apart from the core choice of the distance function that determines which training examples are the nearest, the choice of the number of neighbors to consider is crucial.
- Once multiple neighbors are considered, further extensions become possible.
- For example, the (relative) influence of a neighbor on the prediction may be made dependent on its distance from the query point (distance-weighted k -nearest neighbors).
- Or the prediction may be computed from a local model that is constructed on the fly for a given query point (i.e. from its nearest neighbors) rather than by a simple majority or averaging rule.

k-Nearest Neighbors: Basic Ingredients

- **Distance Metric**

The distance metric, together with a possible task-specific scaling or weighting of the attributes, determines which training example(s) are nearest to a query point and thus selects the training example(s) used to compute a prediction.

- **Number of Neighbors**

The number of neighbors of the query point that are considered can range from only one (the basic nearest neighbor approach) through a few (like k -nearest neighbor approaches) to, in principle, all data points as an extreme case.

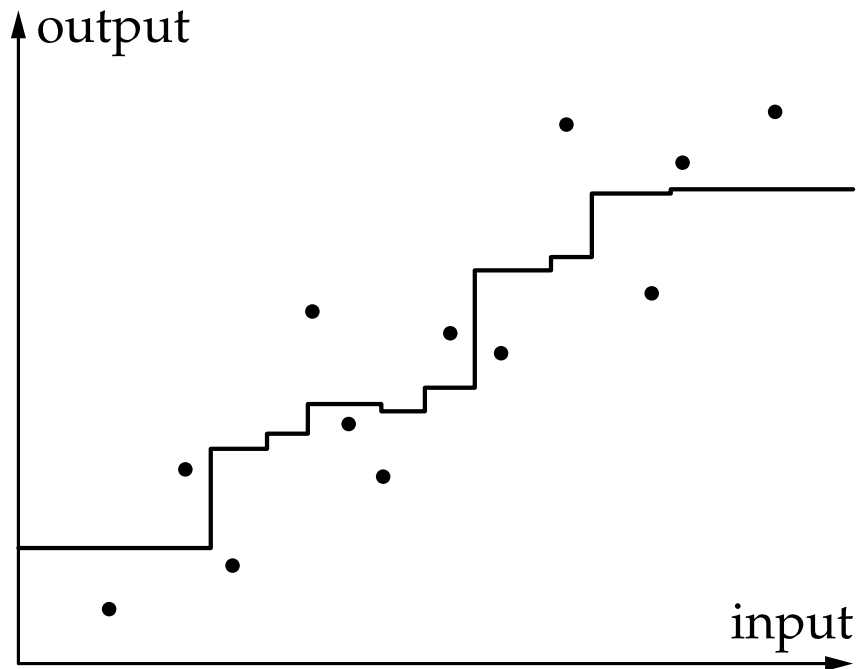
- **Weighting Function for the Neighbors**

If multiple neighbors are considered, it is plausible that closer (more similar) neighbors should have a stronger influence on the prediction result. This can be expressed by a weighting function yielding higher values for smaller distances.

- **Prediction Function**

If multiple neighbors are considered, one needs a procedure to compute the prediction from the classes or target values of these neighbors, since they may differ and thus may not yield a unique prediction directly.

k-Nearest Neighbors: More Than One Neighbor



3-nearest neighbor predictor,
using a simple averaging
of the target values
of the nearest neighbors.

Note that the prediction is still
a piecewise constant function.

- The main effect of the number k of considered neighbors is how much the class boundaries or the numeric prediction is smoothed.
- If only one neighbor is considered, the prediction is constant in the Voronoi cells of the training data set and meeting the data points.

k-Nearest Neighbors: Number of Neighbors

- If only one neighbor is considered, the prediction is constant in the Voronoi cells of the training data set.
- This makes the prediction highly susceptible to the deteriorating effects of incorrectly labeled instances or outliers w.r.t. their class, because a single data point with the wrong class spoils the prediction in its whole Voronoi cell.
- Considering several neighbors ($k > 1$) mitigates this problem, since neighbors having the correct class can override the influence of an outlier.
- However, choosing a very large k is also not generally advisable, because it can prevent the classifier from being able to properly approximate narrow class regions or narrow peaks or valleys in a numeric target function.
- The example of 3-nearest neighbor prediction on the preceding slide (using a simple averaging of the target values of these nearest neighbors) shows the smoothing effect (especially at the borders of the input range).
- The interpolation deviates considerably from the data points.

k-Nearest Neighbors: Number of Neighbors

- A common method to automatically determine an appropriate value for the number k of neighbors is **cross validation** (brief reminder...).
- The training data set is divided into r cross validation folds of (approximately) equal size.
- The fold sizes may differ by one data point, to account for the fact that the total number of training examples may not be divisible by r , the number of folds.
- Then r classification or prediction experiments are performed: each combination of $r - 1$ folds is once chosen as the training set, with which the remaining fold is classified or the (numeric) target value is predicted, using all numbers k of neighbors from a user-specified range.
- The classification accuracy or the prediction error is aggregated, for the same value k , over these experiments.
- Finally the number k of neighbors with the lowest aggregated error is chosen.

k-Nearest Neighbors: Weighting

- Approaches that weight the considered neighbors differently based on their distance to the query point are known as **distance-weighted k-nearest neighbor** or (for numeric targets) **locally weighted regression** or **locally weighted/estimated scatterplot smoothing (LOWESS or LOESS)**.
- Such weighting is mandatory in the extreme case in which all n training examples are used as neighbors, because otherwise only the majority class or the global average of the target values can be predicted.
- However, it is also recommended for $k < n$, since it can, at least to some degree, counteract the smoothing effect of a large k , because the excess neighbors are likely to be farther away and thus will influence the prediction less.
- It should be noted though, that distance-weighted k -NN is *not* a way of avoiding the need to find a good value for the number k of neighbors.

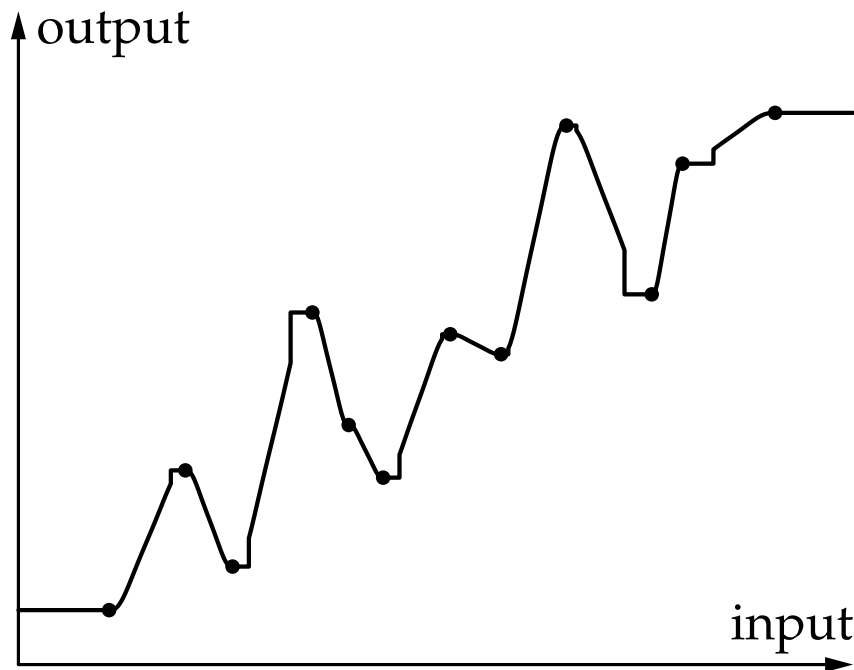
k-Nearest Neighbors: Weighting

- A typical example of a weighting function for the nearest neighbors is the so-called **tricubic weighting function**, which is defined as

$$w(s_i, q, k) = \left(1 - \left(\frac{d(s_i, q)}{d_{\max}(q, k)} \right)^3 \right)^3.$$

- q is the query point,
 k is the number of considered neighbors,
 s_i is (the input vector of) the i -th nearest neighbor of q in the training data set,
 d is the employed distance function, and
 $d_{\max}(q, k)$ is the maximum distance between
any two points from the set $\{q, s_1, \dots, s_k\}$,
that is, $d_{\max}(q, k) = \max_{a, b \in \{q, s_1, \dots, s_k\}} d(a, b)$.
- The function w yields the weight with which the target value of the i -th nearest neighbor s_i of q enters the prediction computation.

k-Nearest Neighbors: Weighting



2-nearest neighbor predictor,
using a distance-weighted averaging
of the nearest neighbors.

This ensures that the prediction
meets the data points.

- Note that the interpolation is mainly linear (because *two* nearest neighbors are used), except for some small plateaus close to the data points.
- These result from the weighting, and certain jumps at points where the two closest neighbors are on the same side of the query point.

k-Nearest Neighbors: Weighting

- An alternative approach to distance-weighted k -NN consists in abandoning the requirement of a predetermined number of nearest neighbors.
- Rather a data point is weighted with a **kernel function** K that is defined on its distance d to the query point and that satisfies the following properties:
(1) $K(d) \geq 0$, (2) $K(0) = 1$ (or at least that K has its mode at 0), and
(3) $K(d)$ decreases monotonically for $d \rightarrow \infty$.
- In this case all training examples for which the kernel function yields a non-vanishing value w.r.t. a given query point are used for the prediction.
- Since the density of training examples may, of course, differ for different regions of the feature space, this may lead to a different number of neighbors being considered, depending on the query point.
- If the kernel function has an infinite support (that is, does not vanish for any finite argument value), *all* data points are considered for any query point.

k-Nearest Neighbors: Weighting

- By using a kernel function, we try to mitigate the problem of choosing a good value for the number k of neighbors, which is now taken care of by the fact that instances that are farther away have a smaller influence on the prediction.
- On the other hand, we now face the problem of having to decide how quickly the influence of a data point should decline with increasing distance, which is analogous to choosing the right number of neighbors and can be equally difficult to solve.
- Examples of kernel functions with a finite support, given as a radius σ around the query point within which training examples are considered, are

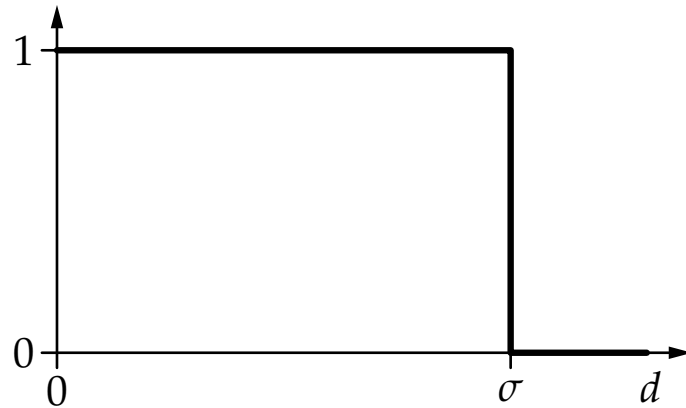
$$\begin{aligned}K_{\text{rect}}(d, \sigma) &= \tau(d \leq \sigma), \\K_{\text{triangle}}(d, \sigma) &= \tau(d \leq \sigma) \cdot \left(1 - \frac{d}{\sigma}\right), \\K_{\text{bisquare}}(d, \sigma) &= \tau(d \leq \sigma) \cdot \left(1 - \frac{d^2}{\sigma^2}\right)^2, \\K_{\text{tricubic}}(d, \sigma) &= \tau(d \leq \sigma) \cdot \left(1 - \frac{d^3}{\sigma^3}\right)^3,\end{aligned}$$

where $\tau(\phi)$ is 1 if ϕ is true and 0 otherwise.

k-Nearest Neighbors: Weighting

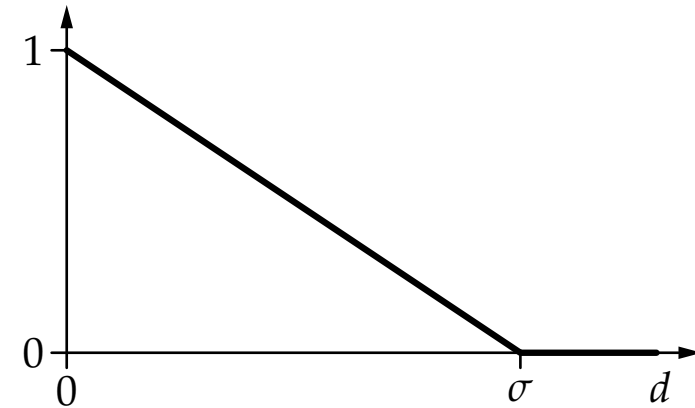
rectangle function:

$$K(d, \sigma) = \tau(d \leq \sigma)$$



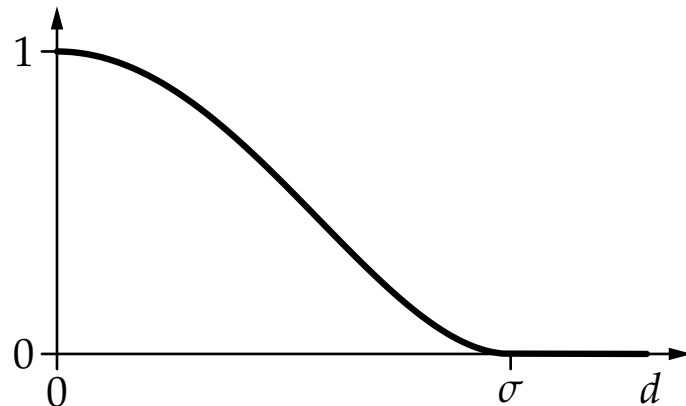
triangle function:

$$K(d, \sigma) = \tau(d \leq \sigma) \left(1 - \frac{d}{\sigma}\right)$$



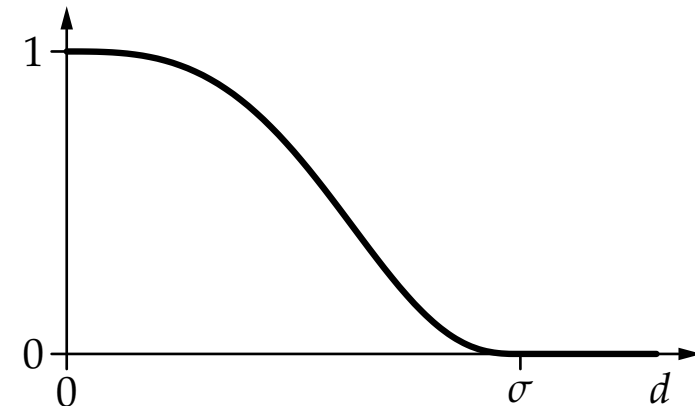
bisquare function:

$$K(d, \sigma) = \tau(d \leq \sigma) \left(1 - \frac{d^2}{\sigma^2}\right)$$



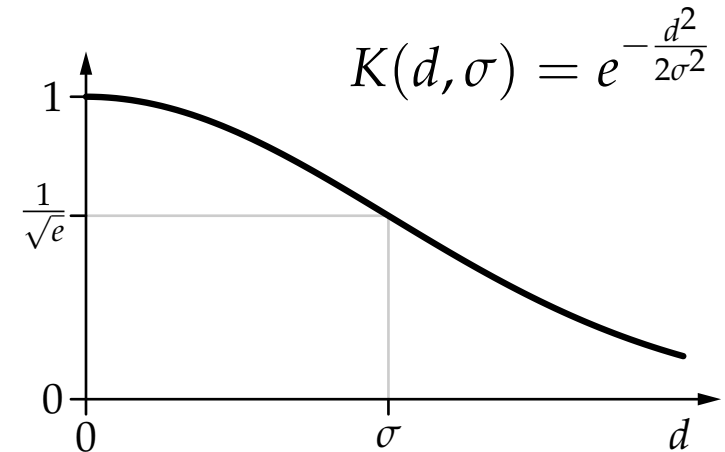
tricubic function:

$$K(d, \sigma) = \tau(d \leq \sigma) \left(1 - \frac{d^3}{\sigma^3}\right)$$

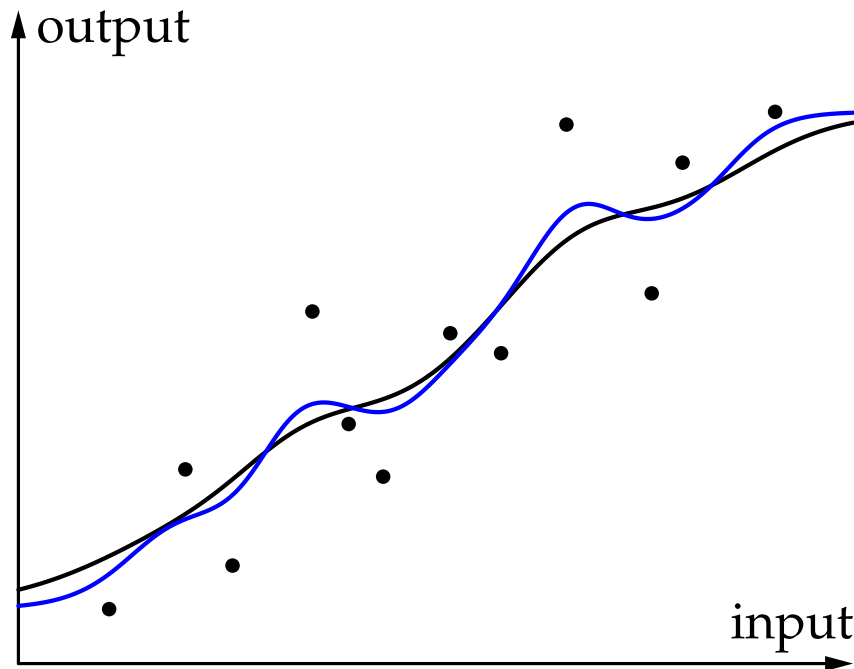


k-Nearest Neighbors: Weighting

- A typical kernel function with infinite support is the **Gaussian function**, where d is the distance of the training example to the query point and σ^2 is a parameter that determines the spread of the Gaussian function.
- An advantage of kernels with infinite support is that the prediction function is smooth (has no jumps) if the kernel is smooth, because then a training case does not suddenly enter the prediction if a query point is moved by an infinitesimal amount, but its influence rises smoothly in line with the kernel function.
- One also does not have to choose a number of neighbors.
- However, the disadvantage is, as already pointed out, that one has to choose an appropriate radius σ for the kernel function, which can be more difficult to choose than an appropriate number of neighbors.



k-Nearest Neighbors: Weighting



Kernel weighted regression, using a Gaussian kernel function, with two different values of σ ($\sigma_{\text{blue}} < \sigma_{\text{black}}$).

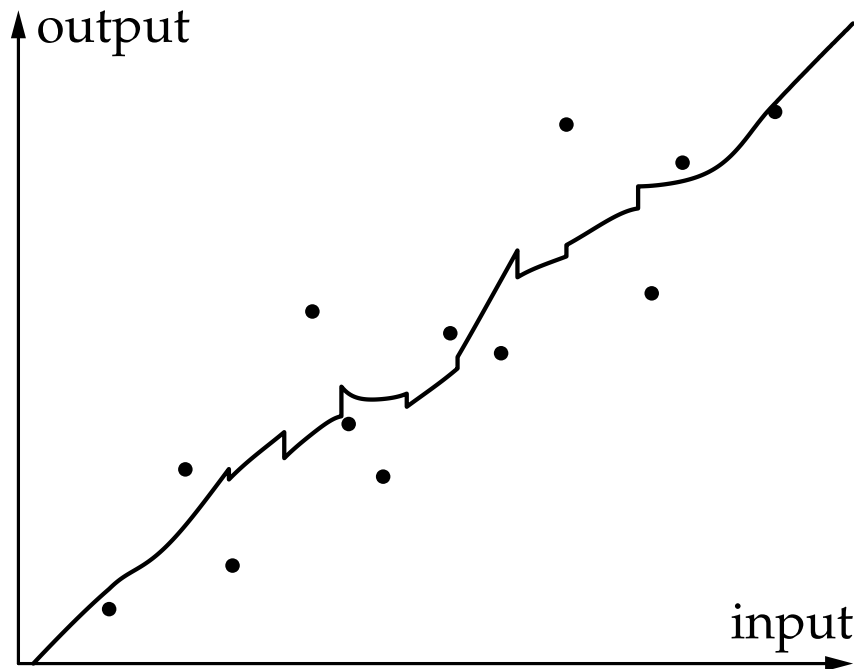
This ensures that the prediction is smooth, though possibly not very close to the training data points.

- Note that the regression function is smooth, because the kernel function is smooth and always refers to all data points as neighbors, so that no jumps occur due to a change in the set of nearest neighbors.
- The price is an increased computational cost, since the kernel function has to be evaluated for all data points, not only for the nearest neighbors.

k-Nearest Neighbors: Prediction Function

- The most straightforward choices for the prediction function are a simple (weighted) majority vote for classification or a simple (weighted) average for numeric prediction.
- However, especially for numeric prediction, one may also consider more complex prediction functions, like building a local regression model from the neighbors (usually with a linear function or a low-degree polynomial), thus arriving at **locally weighted polynomial regression**.
- The prediction is then computed from this local model.
- Not surprisingly, distance weighting may also be used in such a setting.
- Such an approach should be employed with a larger number of neighbors, so that a change of the set of nearest neighbors leads to less severe changes of the local regression line.
(Although there will still be jumps of the predicted value in this case, they are just less high.)

k-Nearest Neighbors: Locally Weighted Regression



4-nearest neighbor distance-weighted
locally linear regression
(using a tricubic weighting function).

Although linear regression is used,
the nearest neighbors do not enter
with unit weight!

- Note how the distance weighting leads to deviations from straight lines between the data points.
- Note also the somewhat erratic behavior of the resulting regression function (jumps at points where the set of nearest neighbors changes). This will be less severe, the larger the number of neighbors.

k-Nearest Neighbors: Locally Weighted Regression

- Locally weighted regression is usually applied with simple polynomials:
 - most of the time linear,
 - rarely quadratic,
 - basically never any higher order.
- The reason is that the local character of the regression is supposed to take care of the global shape of the function, so that the regression function is not needed to model it.
- The advantage of locally weighted polynomial regression is that no global regression function, derived from a data generation model, needs to be found.
- This makes the method applicable to a broad range of prediction problems.
- Its disadvantages are that its prediction can be less reliable in sparsely sampled regions of the feature space, where the locally employed regression function is stretched to a larger area and thus may fit the actual target function badly.

Feature / Distance Weights

- It is crucial for the success of a nearest neighbor approach that a proper distance function is chosen.
- A very simple and natural way of adapting a distance function to the needs of the prediction problem is to use distance weights, thus giving certain features a greater influence than others.
- If prior information is available about which features are most informative w.r.t. the target, this information can be incorporated into the distance function.
- One may also try to determine appropriate feature weights automatically.
- The simplest approach is to start with equal feature weights and to modify them iteratively in a hill climbing fashion:
 - apply a (small) random modification to the feature weights,
 - check with cross validation whether this improves the prediction quality;
 - if it does, accept the new weights, otherwise keep the old.
 - Repeat until some termination criterion is met.

k-Nearest Neighbors: Implementation

- A core issue of implementing nearest neighbor prediction is the data structure used to store the training examples.
- In a naïve implementation they are simply stored as a list, which requires merely $O(n)$ time, where n is the number of training examples.
- However, though fast at training time, this approach has the serious drawback of being very slow at execution time, because a linear traversal of all training examples is needed to find the nearest neighbor(s), requiring $O(nm)$ time, where m is the dimensionality of the data.
- As a consequence, this approach becomes quickly infeasible with a growing number of training examples or for high-dimensional data.
- Better approaches rely on data structures like a *kd*-tree (short for *k*-dimensional tree, where the *k* refers to the number of dimensions, *not* the number of neighbors), an *R*- or *R*^{*}-tree, a *UB*-tree etc.

k-Nearest Neighbors: Implementation

- With such data structures the query time can be reduced to $O(\log n)$ per query data point.
- The time to store the training examples (that is, the time to construct an efficient access structure for them) is, of course, worse than for storing them in a simple list.
- However, with a good data structure and algorithm it is usually acceptably longer.
- For example, a *kd*-tree is constructed by iterative bisections in different dimensions that split the set of data points (roughly) equally.
- As a consequence, constructing it from n training examples takes $O(n \log n)$ time if a linear time algorithm for finding the median in a dimension is employed.
- Whether such an approach pays off, depends also on the expected number of query points compared to the number of training data points.

Data Set Reduction and Prototype Building

- A core problem of nearest neighbor approaches is to quickly find the nearest neighbors of a given query point.
- This becomes an important practical problem if the training data set is large and predictions must be computed (very) quickly.
- In such a case one may try to reduce the set of training examples in a preprocessing step, so that a set of relevant or prototypical data points is found, which yields basically the same prediction quality.
- Note that this set may or may not be a subset of the training examples, depending on whether the algorithm used to construct this set merely samples from the training examples or constructs new data points if necessary.
- Note also that there are usually no or only few actually redundant data points, which can be removed without affecting the prediction at all.
- This is obvious for the numerical case and a 1-nearest neighbor classifier, but also holds for a k -nearest neighbor classifier with $k > 1$, because any removal of data points may change the vote at some point and potentially the classification.

Data Set Reduction and Prototype Building

- A straightforward approach is based on a simple iterative merge scheme:
 - At the beginning each training example is considered as a prototype.
 - Then successively two nearest prototypes are merged as long as the prediction quality on some hold-out test data set is not reduced.
 - Prototypes can be merged, for example, by simply computing a weighted sum, with the relative weights determined by how many original training examples a prototype represents.
 - This is similar to hierarchical agglomerative clustering (discussed later).
- More sophisticated approaches may employ, for example, genetic algorithms or any other method for solving a combinatorial optimization problem.
- This is possible, since the task of finding prototypes can be viewed as the task to find a subset of the training examples that yields the best prediction quality (on a given test data set, *not* the training data set): finding best subsets is a standard combinatorial optimization problem.

Summary k-Nearest Neighbors

- **Predict with Target Values of k Nearest Neighbors**
 - classification: majority vote
 - numeric prediction: average value
- **Special Case of Case- or Instance-based Learning**
 - method is easy to understand
 - intuitive and plausible prediction principle
 - lazy learning: no model is constructed
- **Ingredients of k-Nearest Neighbors**
 - Distance Metric / Feature Weights
 - Number of Neighbors
 - Weighting Function for the Neighbors
 - Prediction Function

Bayes Classifiers

Bayes Classifiers

- **Probabilistic Classification and Bayes' Rule**
- **Naive Bayes Classifiers** (conditional independence)
 - 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 or Gaussian Bayes Classifiers** (linear conditional dependence)
 - 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 to some case. (It is $q_i \in \{0, 1\}$ for a deterministic classifier.)
 - The probability of a correct class assignment is

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

- Therefore the best choice for the q_i is

$$q_i = \begin{cases} 1, & \text{if } i = \operatorname{argmax}_{k=1}^{n_C} p_k, \\ 0, & \text{otherwise.} \end{cases}$$

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

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

(Ties are broken arbitrarily.)

- **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.**

Reminder: Probability and Notation

- A **probability** is an additive, non-negative measure on a sample space Ω , which is a set of **elementary events**.
- A probability assigns a number in $[0, 1]$ to a set of elementary events.
For example, $P(E) = p$ means that the set $E \subseteq \Omega$ has the probability $p \in [0, 1]$.
- A **random variable** is a mapping from a sample space Ω to some domain D , most often the real numbers (i.e. $D = \mathbb{R}$).
- For a random variable $X : \Omega \rightarrow D$ one often writes $P(X = x) = p$.
This is formally incorrect (since “ $X = x$ ” is a logical expression and not a set).
However, it is meant as a shorthand for $P(\{\omega \in \Omega \mid X(\omega) = x\})$.
- $P(A_1 = a_1, \dots, A_m = a_m)$ is a shorthand for $P(A_1 = a_1 \wedge \dots \wedge A_m = a_m)$,
which in turn is a shorthand for $P(\{\omega \in \Omega \mid A_1(\omega) = a_1 \wedge \dots \wedge A_m(\omega) = a_m\})$.
- This may be further shortened to $P(A_1, \dots, A_m)$, leaving a_1, \dots, a_m implicit.
(May also occur in implicitly all-quantified statements, that is, $\forall a_1 : \dots \forall a_m :$)

Bayes' Rule and Bayes' Classifiers

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

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

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

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

- **Bayes' classifiers**: Compute the class probabilities as

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

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

Reminder: Chain Rule of Probability

- The **chain rule** of probability is based on the **product rule** of probability:

$$P(X \cap Y) = P(X \mid Y) \cdot P(Y)$$

(Simply multiply the definition of conditional probability with $P(Y)$.)

- **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 also works 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)

$$P(X \cap Y) = P(X) \cdot P(Y)$$

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

- Comparison to the **product rule**

$$P(X \cap Y) = P(X \mid Y) \cdot P(Y)$$

shows that this is equivalent to

$$P(X \mid Y) = P(X)$$

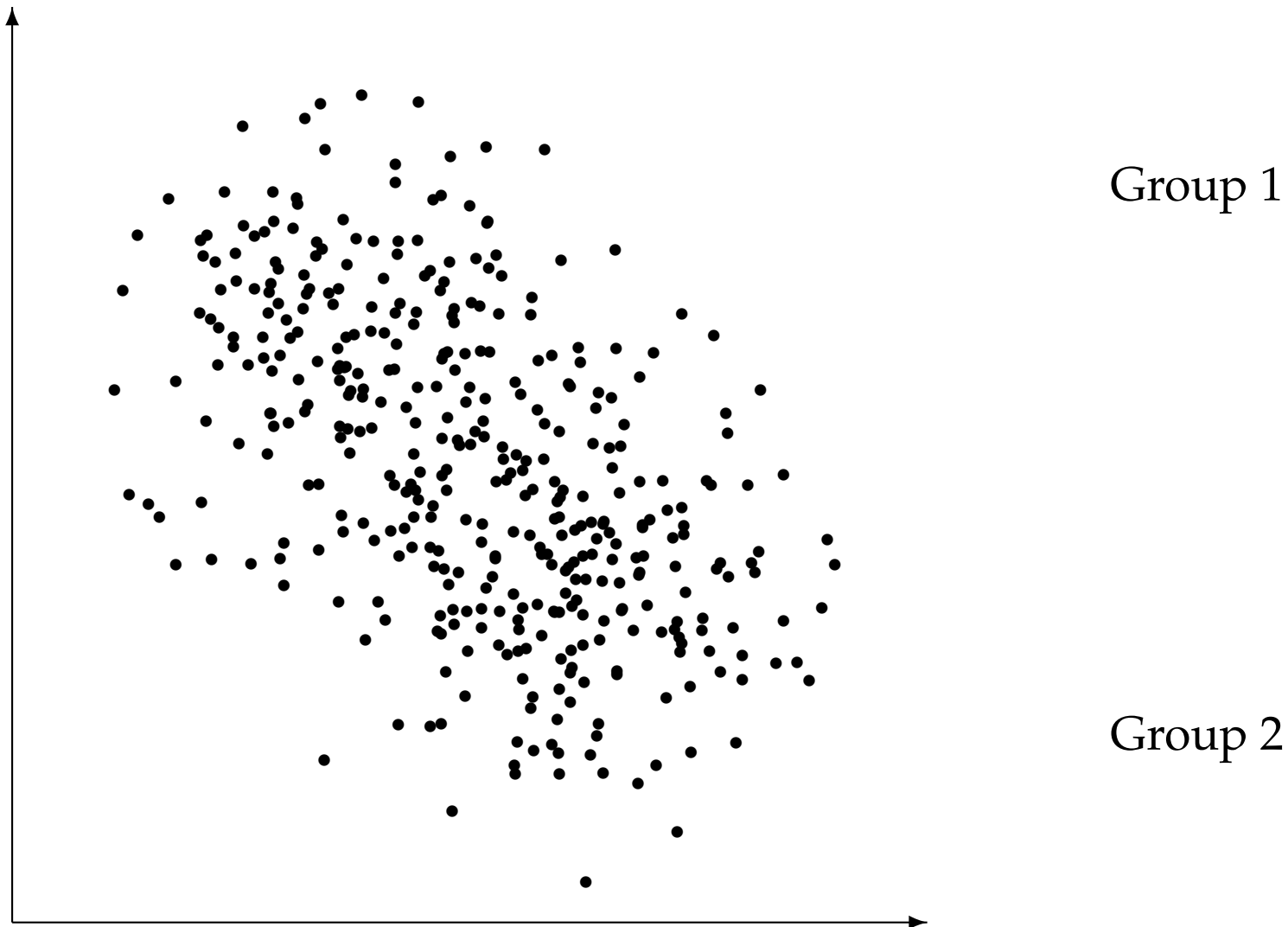
- The same formulae hold conditionally, i.e.

$$P(X \cap Y \mid Z) = P(X \mid Z) \cdot P(Y \mid Z) \quad \text{and}$$

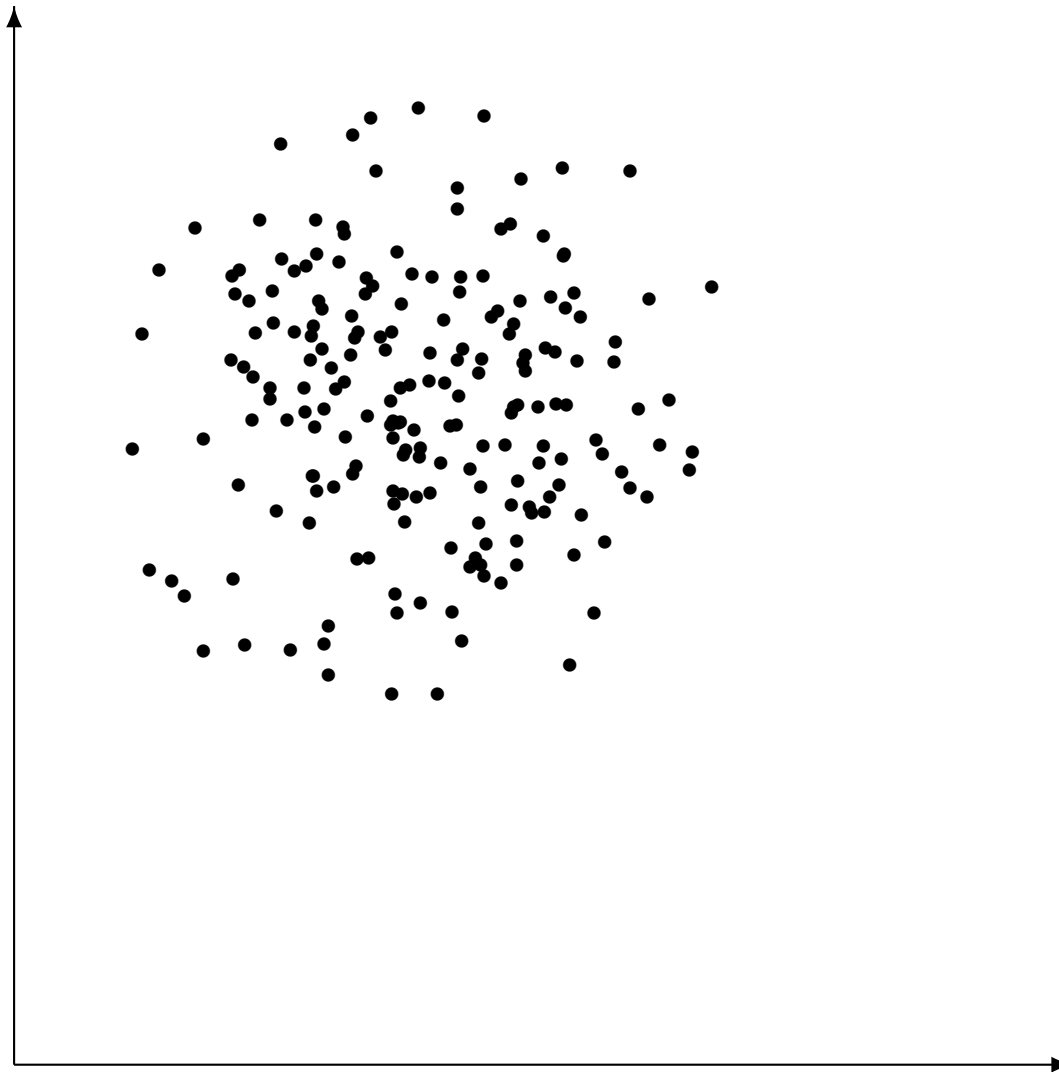
$$P(X \mid Y \cap Z) = P(X \mid Z).$$

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

Conditional Independence: An Example

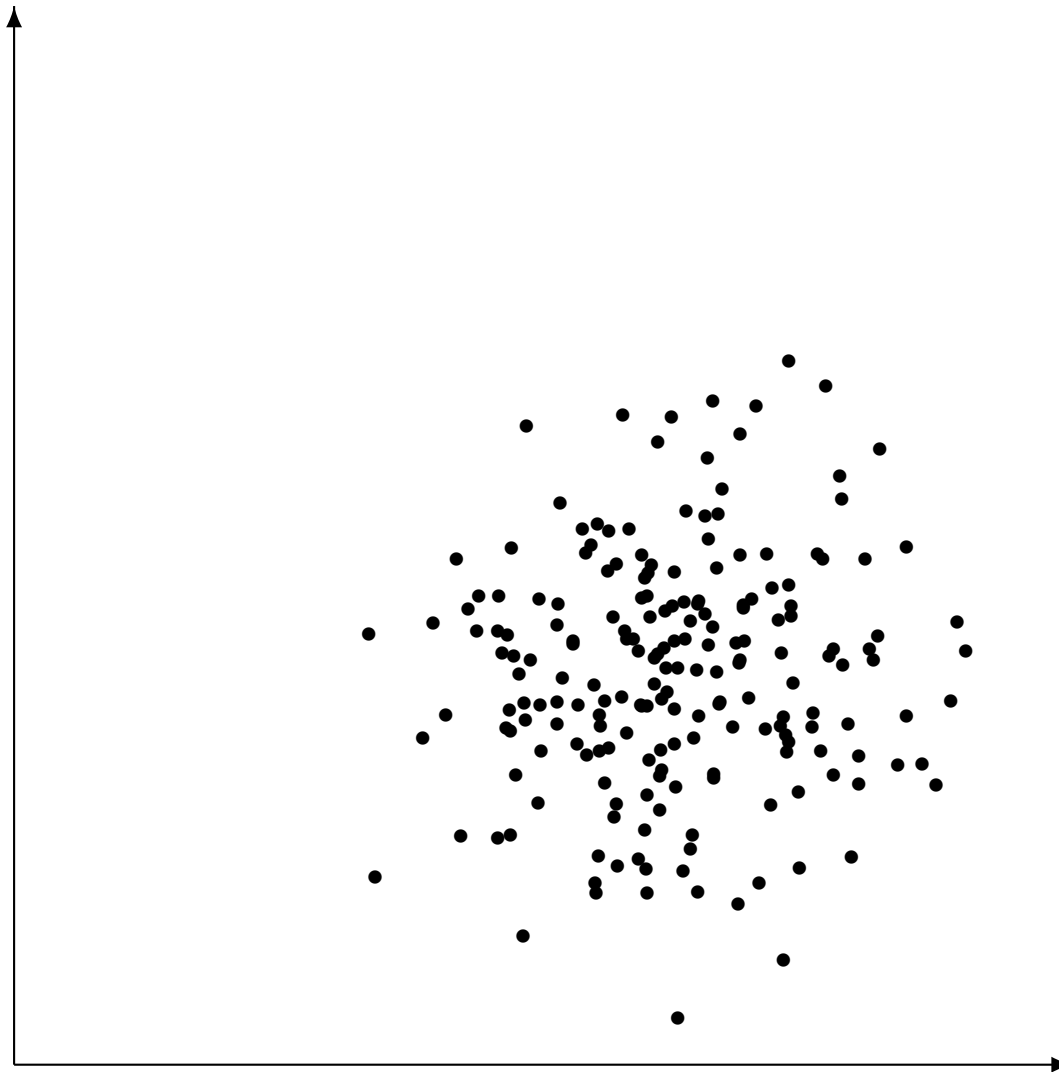


Conditional Independence: An Example



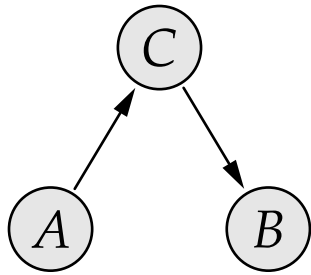
Group 1

Conditional Independence: An Example



Group 2

Conditional and Marginal Independence



causal chain

Example:

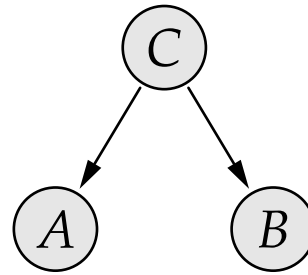
A – accelerator pedal

B – engine speed

C – fuel supply

$$A \not\perp B \mid \emptyset$$

$$A \perp B \mid C$$



common cause

Example:

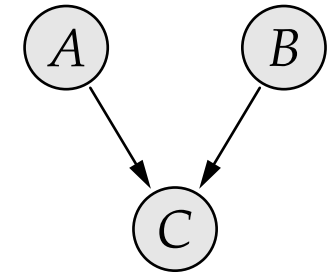
A – ice cream sales

B – bathing accidents

C – temperature

$$A \not\perp B \mid \emptyset$$

$$A \perp B \mid C$$



common effect

Example:

A – influenza

B – measles

C – fever

$$A \perp B \mid \emptyset$$

$$A \not\perp B \mid C$$

$A \perp B \mid C$ means A is conditionally independent of B given C .

$A \perp B \mid \emptyset$ means A is marginally independent of B (“independent given nothing”).

Naive Bayes Classifiers: Cookbook Recipe

Naive (some even say: “idiotic”) Assumption:

The descriptive attributes are conditionally independent given the class.

Bayes’ Rule:

(ω is shorthand for $A_1 = a_1, \dots, A_m = a_m$)

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

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:

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

Naive Bayes Classifiers

- Consequence: Manageable amount of data to store.
Store distributions $P(C = c_i)$ and $\forall 1 \leq k \leq m : P(A_k = a_k \mid C = c_i)$.
- 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 / Categorical Attributes**

$$(n_{A_k} = |\text{dom}(A_k)|)$$

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

$\#(\varphi)$ refers to the number of sample cases satisfying its argument φ .

γ is called **Laplace correction** (to prevent vanishing probabilities).

$\gamma = 0$: maximum likelihood estimation (probabilities may vanish).

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

Naive Bayes Classifiers

Estimation of Probabilities:

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

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

- 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

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

$\zeta = \#(C = c_i)$: maximum likelihood estimation

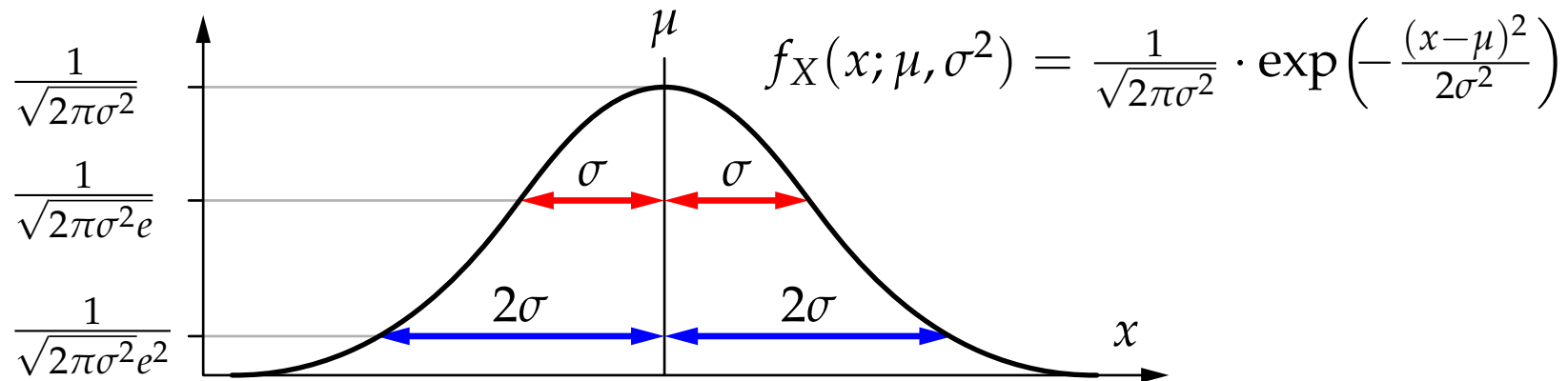
$\zeta = \#(C = c_i) - 1$: unbiased estimation (Bessel's correction)

(Bessel's correction is named after Friedrich Wilhelm Bessel, 1784–1846.)

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 probability density function.



- 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.

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.

The conditional probability tables on the right form the classifier.

Naive Bayes Classifiers: Simple Example 1

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

$$= c_1 \cdot P(\text{Drug A}) \cdot P(\text{male} \mid \text{Drug A}) \cdot f(61 \mid \text{Drug A}) \cdot P(\text{normal} \mid \text{Drug A})$$

$$\approx c_1 \cdot 0.5 \cdot 0.5 \cdot 0.004787 \cdot 0.5 = c_1 \cdot 5.984 \cdot 10^{-4} \approx \mathbf{0.219}$$

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

$$= c_1 \cdot P(\text{Drug B}) \cdot P(\text{male} \mid \text{Drug B}) \cdot f(61 \mid \text{Drug B}) \cdot P(\text{normal} \mid \text{Drug B})$$

$$\approx c_1 \cdot 0.5 \cdot 0.5 \cdot 0.017120 \cdot 0.5 = c_1 \cdot 2.140 \cdot 10^{-3} \approx \mathbf{0.781}$$

$$c_1 \cdot 5.984 \cdot 10^{-4} + c_1 \cdot 2.140 \cdot 10^{-3} \approx 1 \quad \Rightarrow \quad c_1 \approx 365.18$$

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

$$= c_2 \cdot P(\text{Drug A}) \cdot P(\text{female} \mid \text{Drug A}) \cdot f(30 \mid \text{Drug A}) \cdot P(\text{normal} \mid \text{Drug A})$$

$$\approx c_2 \cdot 0.5 \cdot 0.5 \cdot 0.027703 \cdot 0.5 = c_2 \cdot 3.471 \cdot 10^{-3} \approx \mathbf{0.671}$$

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

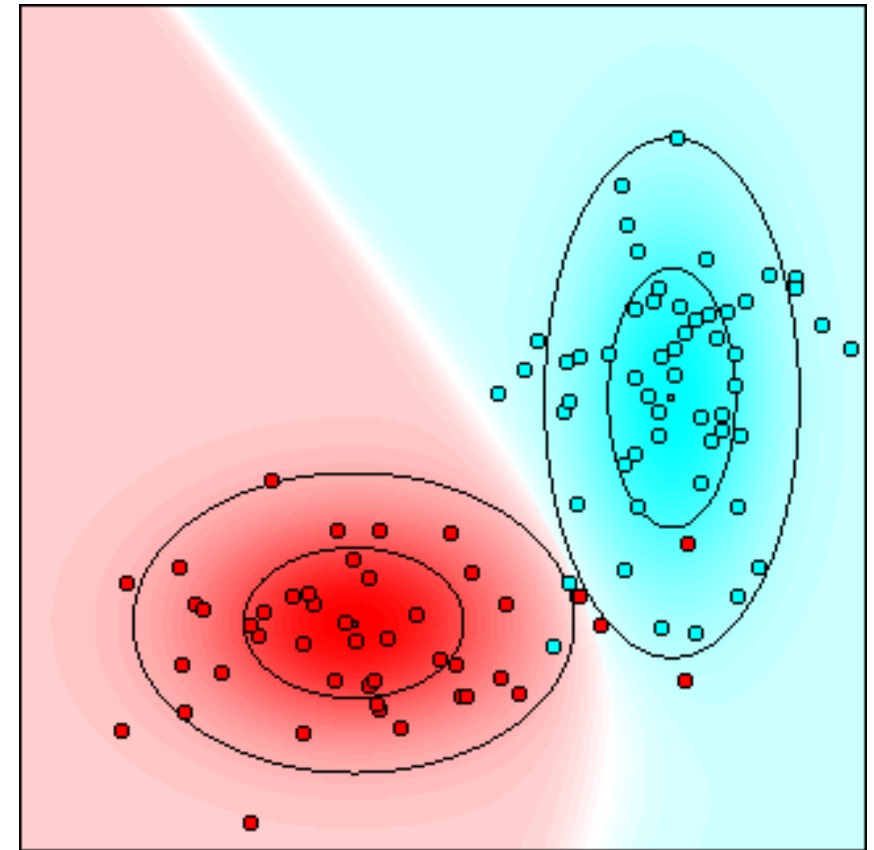
$$= c_2 \cdot P(\text{Drug B}) \cdot P(\text{female} \mid \text{Drug B}) \cdot f(30 \mid \text{Drug B}) \cdot P(\text{normal} \mid \text{Drug B})$$

$$\approx c_2 \cdot 0.5 \cdot 0.5 \cdot 0.013567 \cdot 0.5 = c_2 \cdot 1.696 \cdot 10^{-3} \approx \mathbf{0.329}$$

$$c_2 \cdot 3.471 \cdot 10^{-3} + c_2 \cdot 1.696 \cdot 10^{-3} \approx 1 \quad \Rightarrow \quad c_2 \approx 193.54$$

Naive Bayes Classifiers: Simple Example 2

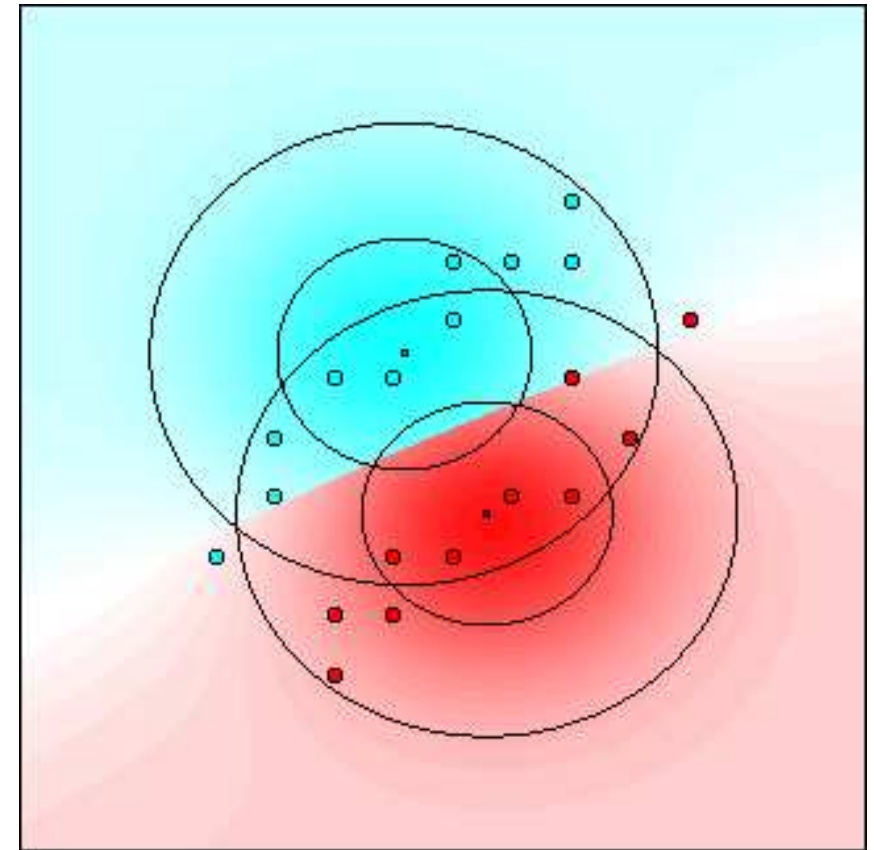
- 100 data points, 2 classes
- Small squares: mean values
- Inner ellipses: one standard deviation
- Outer ellipses: two standard deviations
- Classes overlap: classification is not perfect
- Color saturation indicates probability density



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
- Color saturation indicates probability density



Naive Bayes Classifier

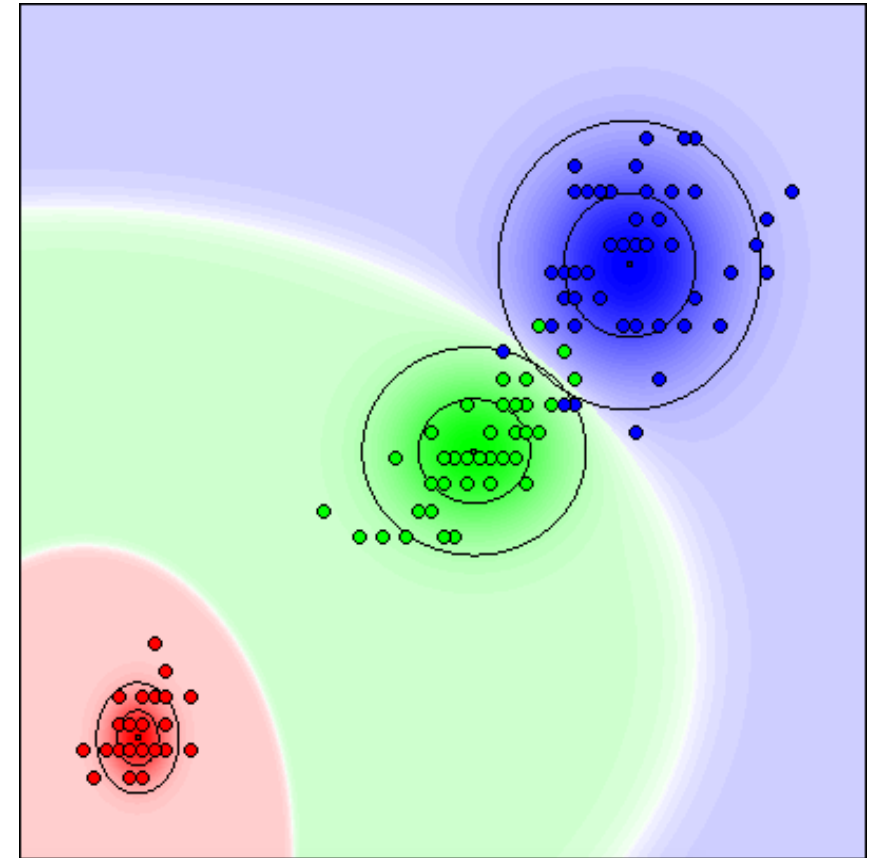
Reminder: The Iris Data

pictures not available in online version

- Collected by Edgar Anderson on the Gaspé Peninsula (Canada).
- First data-analyzed by Ronald Aylmer Fisher (famous statistician).
- 150 cases in total, 50 cases per Iris flower type (specie).
- 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 or Gaussian Bayes Classifiers

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

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

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

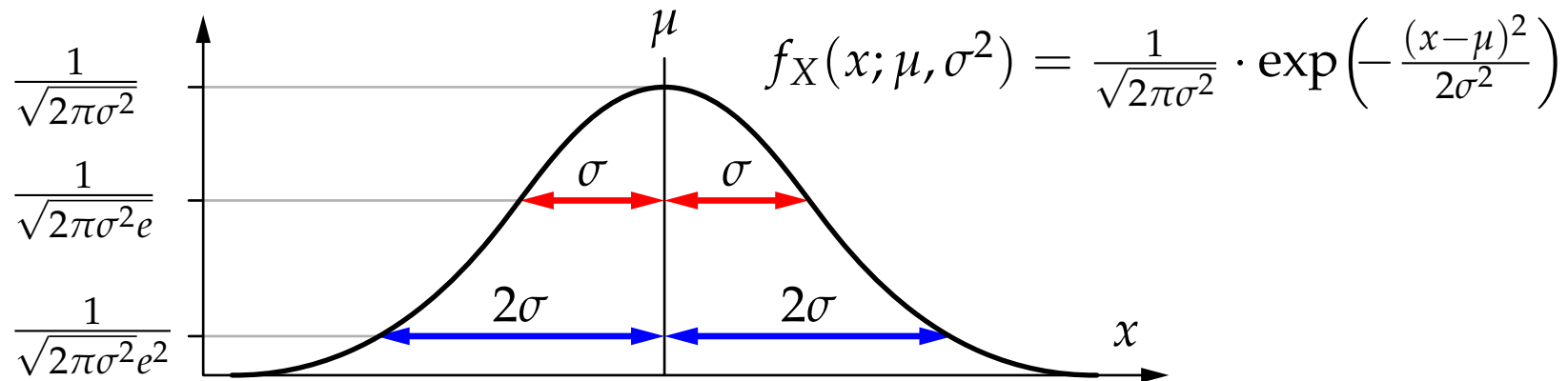
$\mathbf{\Sigma}_i$: covariance matrix for class c_i

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

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 probability density function.



- 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.

Reminder: Vector Products

- A covariance matrix is a multivariate generalization of the variance.

Inner Product Scalar Product

$$\vec{v}^\top \vec{v} \quad \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix}$$

$$(v_1, v_2, \dots, v_m) \quad \sum_{i=1}^m v_i^2$$

Outer Product Matrix Product

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

- In principle, both vector products may be used for a generalization of the variance to multiple dimensions.
- 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 $\vec{v}\vec{v}^\top$):

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

where s_x^2 and s_y^2 are variances and s_{xy} is the covariance of x and y :

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

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

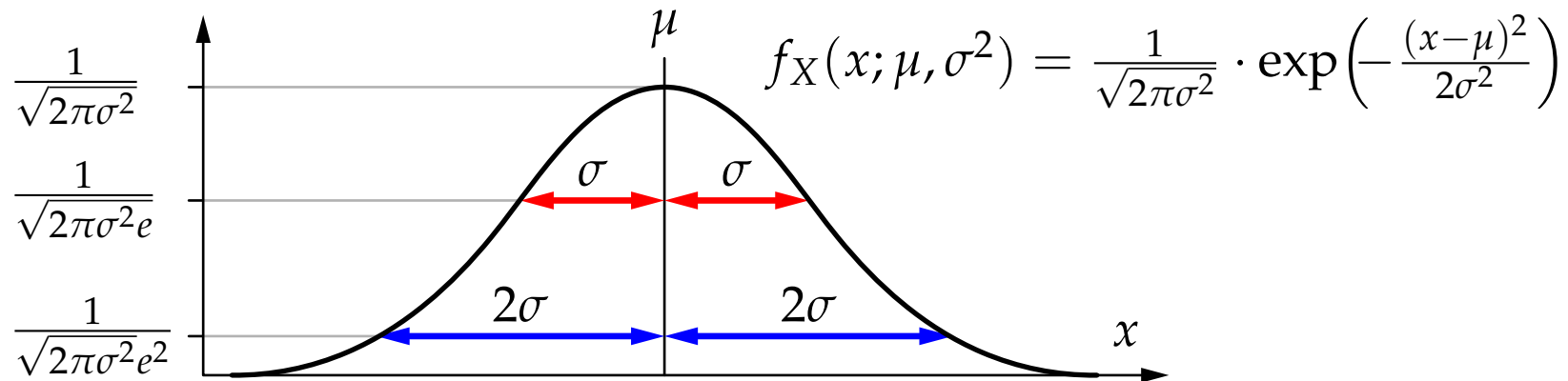
$$s_{xy} = s_{yx} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) = \frac{1}{n-1} \left(\sum_{i=1}^n x_i y_i - n\bar{x}\bar{y} \right)$$

(Using $n-1$ instead of n is called Bessel's correction, after Friedrich Wilhelm Bessel, 1784–1846.)

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 probability density function.



- 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 .

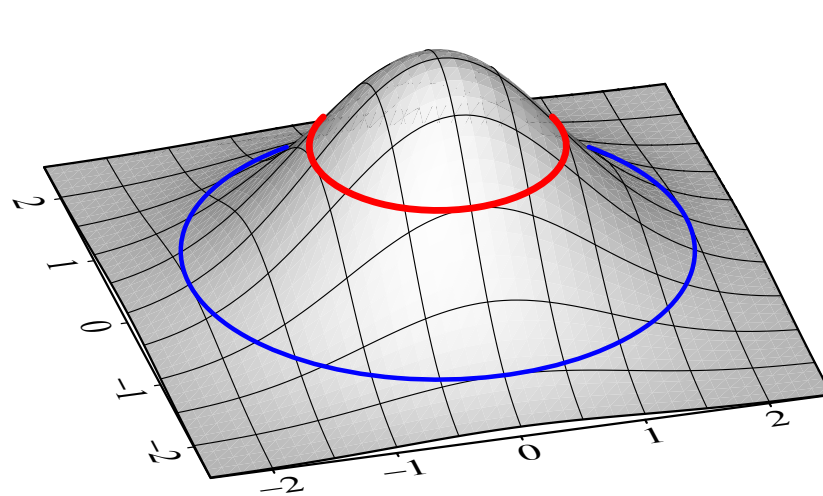
- A **multivariate normal distribution** has the density function

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

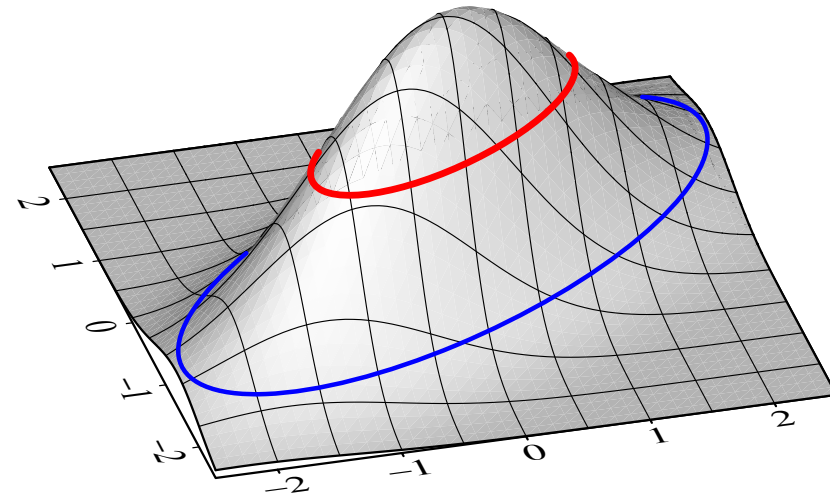
m : size of the vector \vec{x} (it is m -dimensional),
 $\vec{\mu}$: expected value vector, estimated by mean value vector $\bar{\vec{x}}$,
 $\mathbf{\Sigma}$: covariance matrix, estimated by (empirical) covariance matrix \mathbf{S} ,
 $|\mathbf{\Sigma}|$: determinant of the covariance matrix $\mathbf{\Sigma}$.

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



standard



general

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

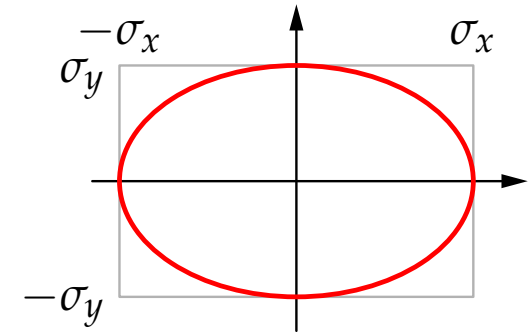
Interpretation of a Covariance Matrix

Question: Is there a multivariate analog of standard deviation?

First insight:

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

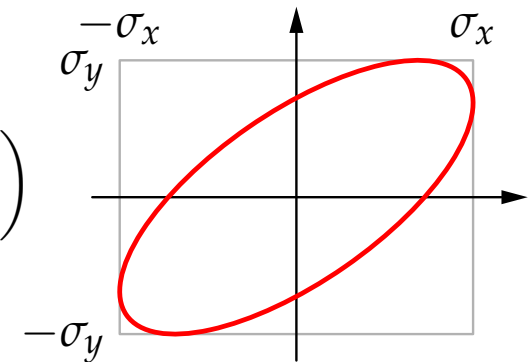
$$\Sigma = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix}$$



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.

Interpretation of a Covariance Matrix

A covariance matrix is always positive semi-definite.

- positive semi-definite (or non-negative definite): $\forall \vec{v} \in \mathbb{R}^m : \vec{v}^\top \mathbf{M} \vec{v} \geq 0$,
negative semi-definite (or non-positive definite): $\forall \vec{v} \in \mathbb{R}^m : \vec{v}^\top \mathbf{M} \vec{v} \leq 0$,

- For any $\vec{x} \in \mathbb{R}^m$ the outer product $\vec{x}\vec{x}^\top$ yields a positive semi-definite matrix:

$$\forall \vec{v} \in \mathbb{R}^m : \vec{v}^\top \vec{x}\vec{x}^\top \vec{v} = (\vec{v}^\top \vec{x})(\vec{x}^\top \vec{v}) = (\vec{v}^\top \vec{x})(\vec{v}^\top \vec{x})^\top = (\vec{v}^\top \vec{x})^2 \geq 0.$$

- If $\mathbf{M}_i, i = 1, \dots, k$, are positive (negative) semi-definite matrices, then $\mathbf{M} = \sum_{i=1}^k \mathbf{M}_i$ is a positive (negative) semi-definite matrix.

$$\forall \vec{v} \in \mathbb{R}^m : \vec{v}^\top \mathbf{M} \vec{v} = \vec{v}^\top \left(\sum_{i=1}^k \mathbf{M}_i \right) \vec{v} = \sum_{i=1}^k \underbrace{\vec{v}^\top \mathbf{M}_i \vec{v}}_{\geq 0} \geq 0.$$

- A(n empirical) covariance matrix is computed as

$$\mathbf{S} = \frac{1}{n-1} \sum_{i=1}^n (\vec{x}_i - \bar{\vec{x}})(\vec{x}_i - \bar{\vec{x}})^\top.$$

As the sum of positive semi-definite matrices, it is positive semi-definite itself.

Interpretation of a Covariance Matrix

A covariance matrix is generally positive definite, unless all data points lie in a lower-dimensional (linear) subspace.

- positive definite: $\forall \vec{v} \in \mathbb{R}^m - \{\vec{0}\} : \vec{v}^\top \mathbf{M} \vec{v} > 0,$
negative definite: $\forall \vec{v} \in \mathbb{R}^m - \{\vec{0}\} : \vec{v}^\top \mathbf{M} \vec{v} < 0,$
- A(n empirical) covariance matrix is computed as $\mathbf{S} = \frac{1}{n-1} \sum_{i=1}^n (\vec{x}_i - \bar{\vec{x}})(\vec{x}_i - \bar{\vec{x}})^\top.$
- Let $\vec{z}_i = (\vec{x}_i - \bar{\vec{x}}), i = 1, \dots, n,$ and suppose that
 $\exists \vec{v} \in \mathbb{R}^m - \{\vec{0}\} : \forall i; 1 \leq i \leq n: \vec{v}^\top \vec{z}_i = 0$ (implying $\vec{v}^\top \vec{z}_i \vec{z}_i^\top \vec{v} = (\vec{v}^\top \vec{z}_i)^2 = 0$).
Furthermore, suppose that the set $\{\vec{z}_1, \dots, \vec{z}_n\}$ of difference vectors spans \mathbb{R}^m .
Then there exist $\alpha_1, \dots, \alpha_n \in \mathbb{R}$ such that $\vec{v} = \alpha_1 \vec{z}_1 + \dots + \alpha_n \vec{z}_n$.
Hence $\vec{v}^\top \vec{v} = \underbrace{\vec{v}^\top \vec{z}_1}_{=0} \alpha_1 + \dots + \underbrace{\vec{v}^\top \vec{z}_n}_{=0} \alpha_n = 0$, implying $\vec{v} = \vec{0}$, contradicting $\vec{v} \neq \vec{0}$.
- Therefore, if the $\vec{z}_i, i = 1, \dots, n,$ span \mathbb{R}^m , then \mathbf{S} is positive definite.
Only if the $\vec{z}_i, i = 1, \dots, n,$ do not span \mathbb{R}^m , that is, if the data points lie in a lower-dimensional (linear) subspace, \mathbf{S} is only positive semi-definite.

Covariance Matrix Decomposition

- Intuitively: **Compute an analog of standard deviation.**
- In the univariate case, standard deviation is the square root of the variance.
- Question: **Can we compute a “square root” of a covariance matrix?**
(Not all matrices, not even all square matrices, allow for computing a “square root”, but positive definite matrices, and hence (most) covariance matrices, do.)
- General idea: Given a positive definite matrix \mathbf{S} , find a matrix \mathbf{M} such that $\mathbf{M}\mathbf{M}^\top = \mathbf{S}$ (intuitively: “square” of \mathbf{M} is \mathbf{S}).
- Problem: Generally, the equation $\mathbf{M}\mathbf{M}^\top = \mathbf{S}$ does not have a unique solution. Generally, there are multiple matrices \mathbf{M} that satisfy the equation.
- A unique solution can be enforced by additional constraints / requirements.
 - **Cholesky Decomposition:** \mathbf{M} is a lower/left triangular matrix.
 - **Eigenvalue Decomposition:** \mathbf{M} is composed of 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: $\forall 1 \leq i, j \leq m : \mathbf{S}_{ij} = \mathbf{S}_{ji}$ or $\mathbf{S}^\top = \mathbf{S}$.
(\mathbf{S}^\top is the transpose of the matrix \mathbf{S} .)
 - positive definite: $\forall \vec{v} \in \mathbb{R}^m - \{0\} : \vec{v}^\top \mathbf{S} \vec{v} > 0$
- 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} .)

$$\mathbf{L}_{ii} = \left(\mathbf{S}_{ii} - \sum_{k=1}^{i-1} \mathbf{L}_{ik}^2 \right)^{\frac{1}{2}}$$
$$\mathbf{L}_{ji} = \frac{1}{\mathbf{L}_{ii}} \left(\mathbf{S}_{ij} - \sum_{k=1}^{i-1} \mathbf{L}_{ik} \mathbf{L}_{jk} \right), \quad j = i + 1, i + 2, \dots, q.$$

Cholesky Decomposition

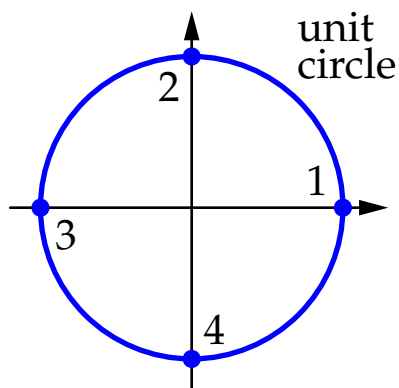
Special Case: Two Dimensions

- Covariance matrix

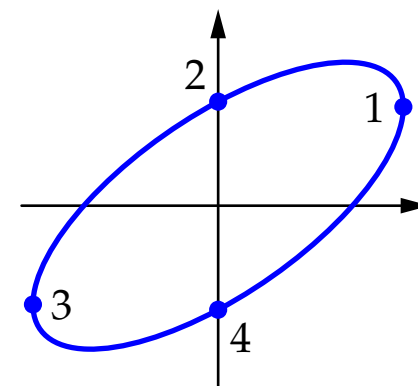
$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$$

- Cholesky decomposition

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



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



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

- The eigenvalues $\lambda_j, j = 1, \dots, m$, 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

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

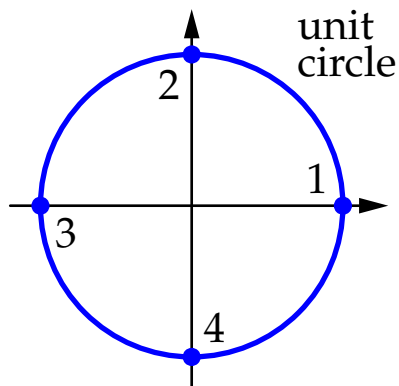
- Eigenvalue decomposition

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

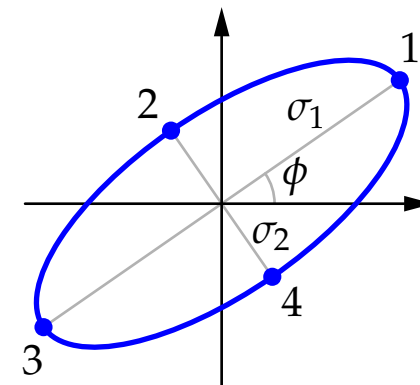
$$s = \sin \phi, c = \cos \phi, \phi = \frac{1}{2} \arctan \frac{2\sigma_{xy}}{\sigma_x^2 - \sigma_y^2},$$

$$\sigma_1 = \sqrt{c^2\sigma_x^2 + s^2\sigma_y^2 + 2sc\sigma_{xy}},$$

$$\sigma_2 = \sqrt{s^2\sigma_x^2 + c^2\sigma_y^2 - 2sc\sigma_{xy}}.$$



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



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 .

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

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

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

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, to which a (hyper-)sphere with radius r is mapped with the eigenvalue decomposition of a covariance matrix Σ , is

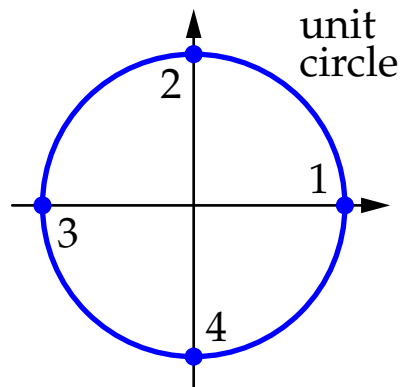
$$V_m(r) = \frac{\pi^{\frac{m}{2}} r^m}{\Gamma(\frac{m}{2} + 1)} \sqrt{|\Sigma|}, \quad \text{where} \quad \begin{aligned} \Gamma(x) &= \int_0^\infty e^{-t} t^{x-1} dt, \quad x > 0, \\ \Gamma(x+1) &= x \cdot \Gamma(x), \quad \Gamma(\tfrac{1}{2}) = \sqrt{\pi}, \quad \Gamma(1) = 1. \end{aligned}$$

Eigenvalue Decomposition

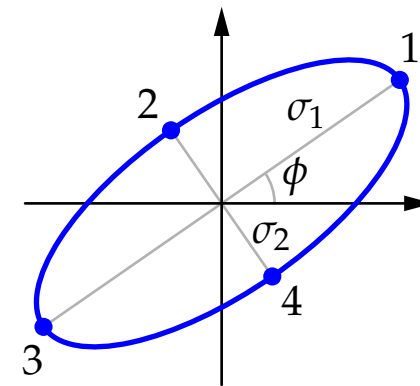
Special Case: Two Dimensions

- Covariance matrix and its eigenvalue decomposition:

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



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



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

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

Full or Gaussian 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

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

$\zeta = \#(C = c_i)$: maximum likelihood estimation

$\tilde{\zeta} = \#(C = c_i) - 1$: unbiased estimation (Bessel's correction)

\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.

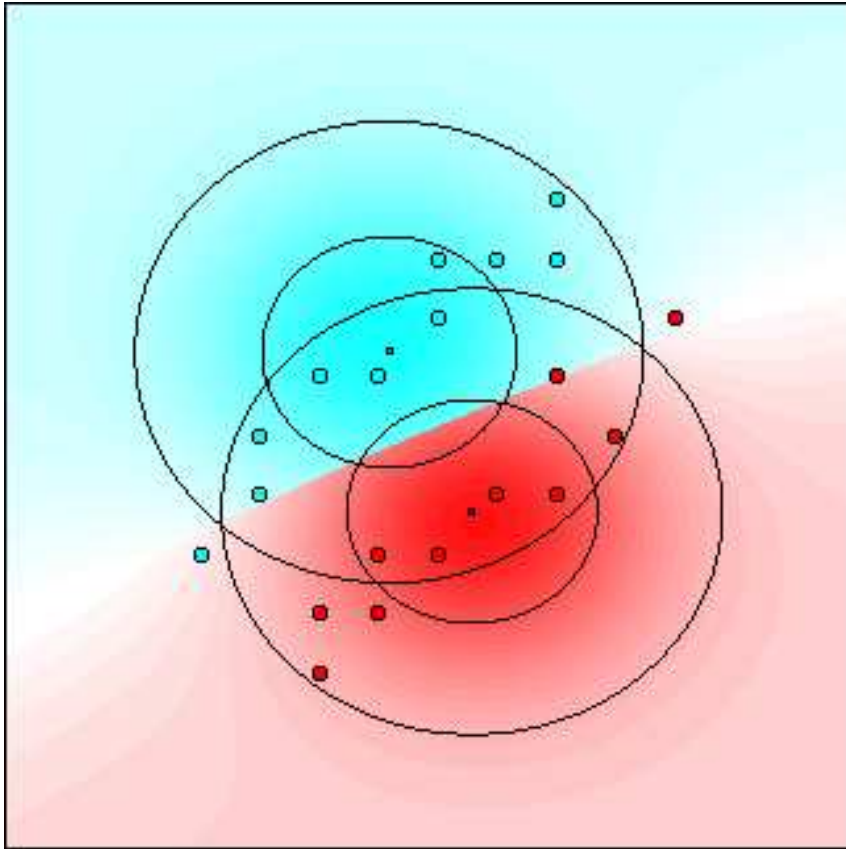
Comparison of Naive and Full / Gaussian Bayes Classifiers

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

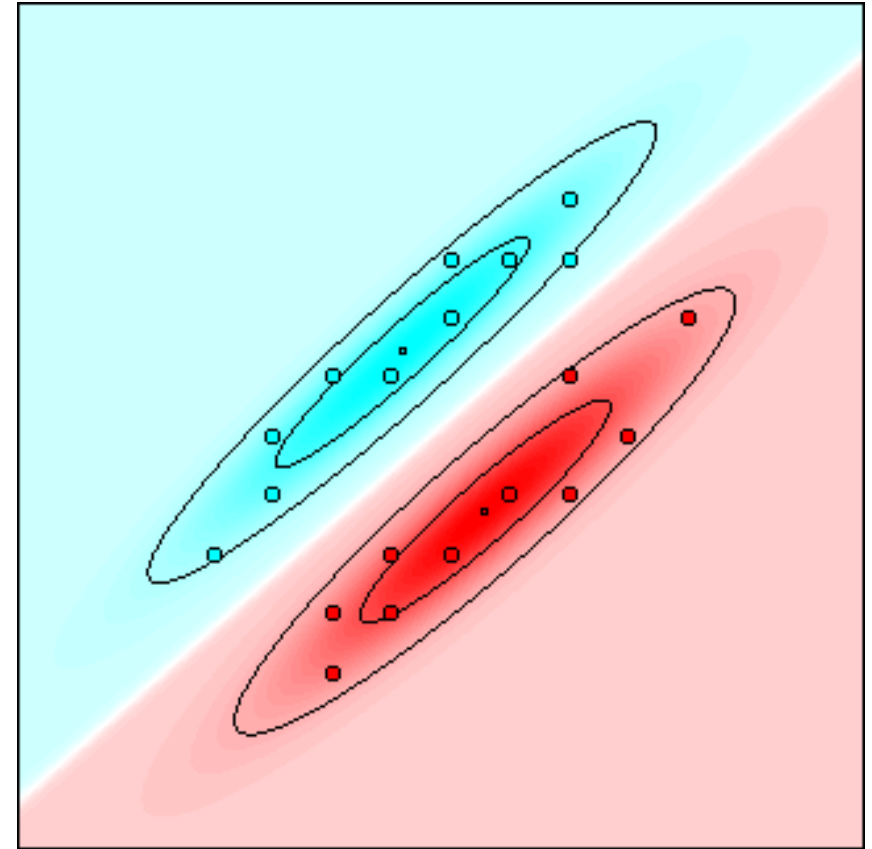
$$\begin{aligned} & f(A_1 = a_1, \dots, A_m = a_m \mid C = c_i) \\ &= \frac{1}{\sqrt{(2\pi)^m |\mathbf{\Sigma}_i|}} \cdot \exp\left(-\frac{1}{2}(\vec{a} - \vec{\mu}_i)^\top \mathbf{\Sigma}_i^{-1}(\vec{a} - \vec{\mu}_i)\right) \\ &= \frac{1}{\sqrt{(2\pi)^m \prod_{k=1}^m \sigma_{i,k}^2}} \cdot \exp\left(-\frac{1}{2}(\vec{a}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

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

Comparison of Naive and Full / Gaussian Bayes Classifiers



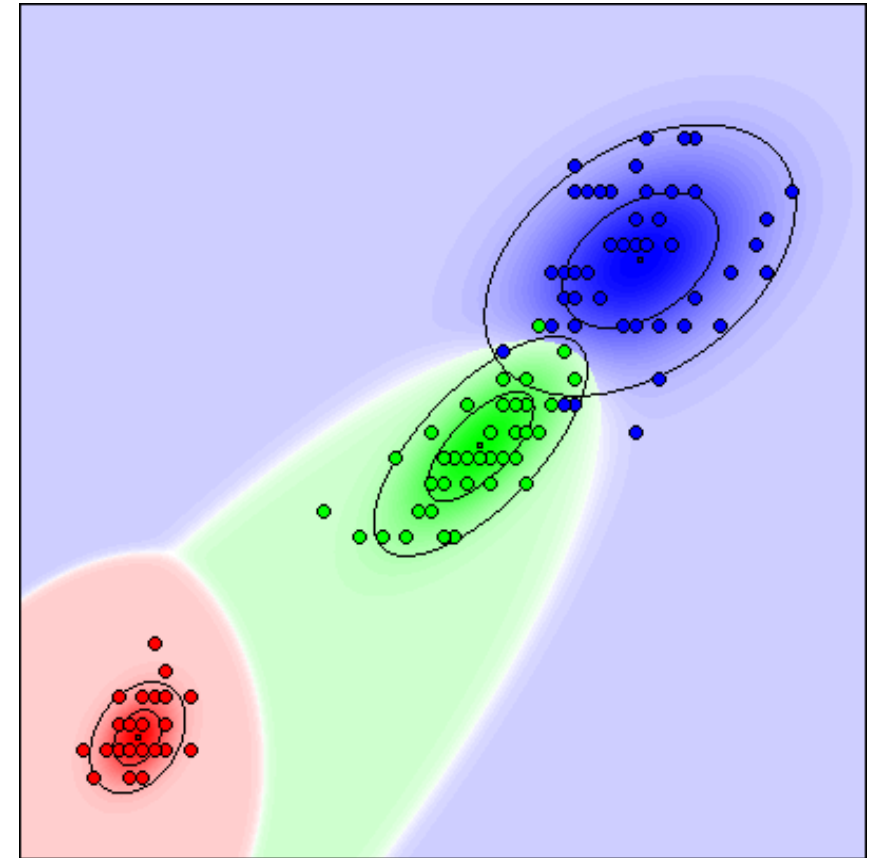
Naive Bayes Classifier



Full / Gaussian Bayes Classifier

Full / Gaussian 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 / Gaussian 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.

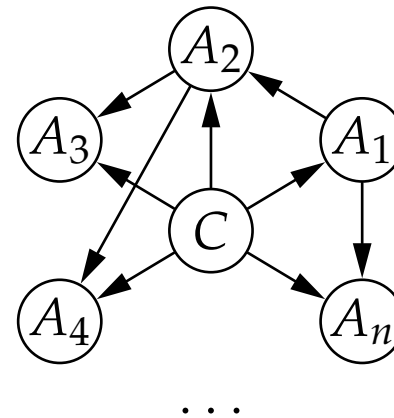
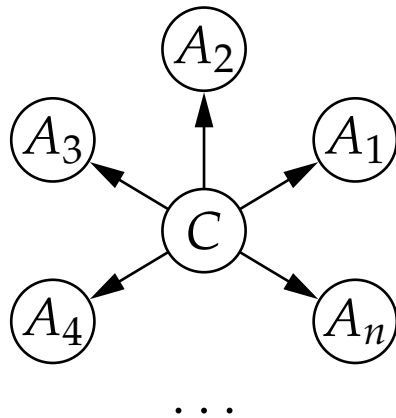
Definition (d -separation):

Let $\vec{G} = (V, \vec{E})$ 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 $X \perp\!\!\!\perp Y \mid Z$ 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 them 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 or Gaussian 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.

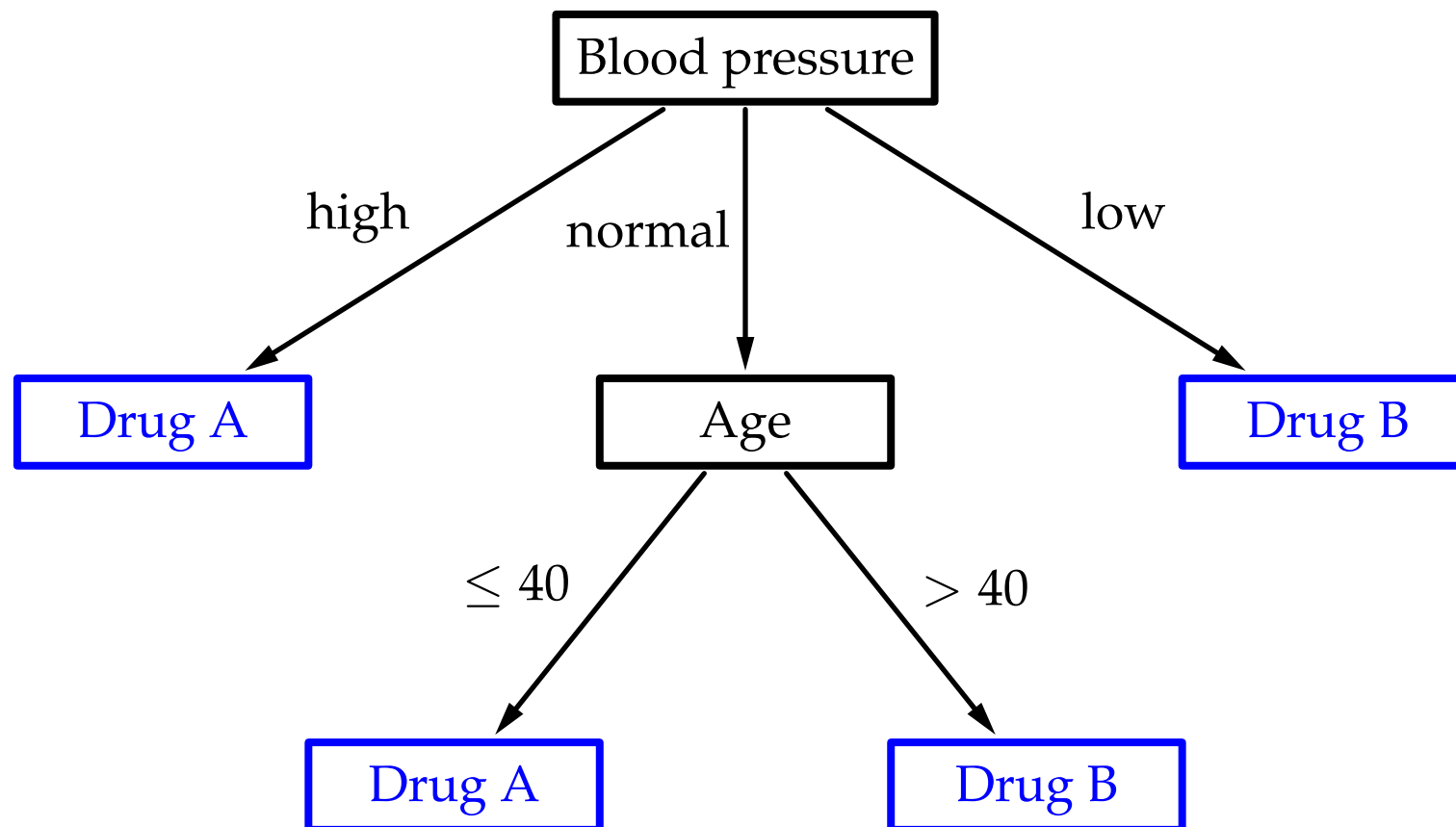
Decision and Regression Trees

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/edge labeled with the outcome of the test.
 - Apply the classification algorithm recursively.
(Make the node reached by following the branch the current node.)

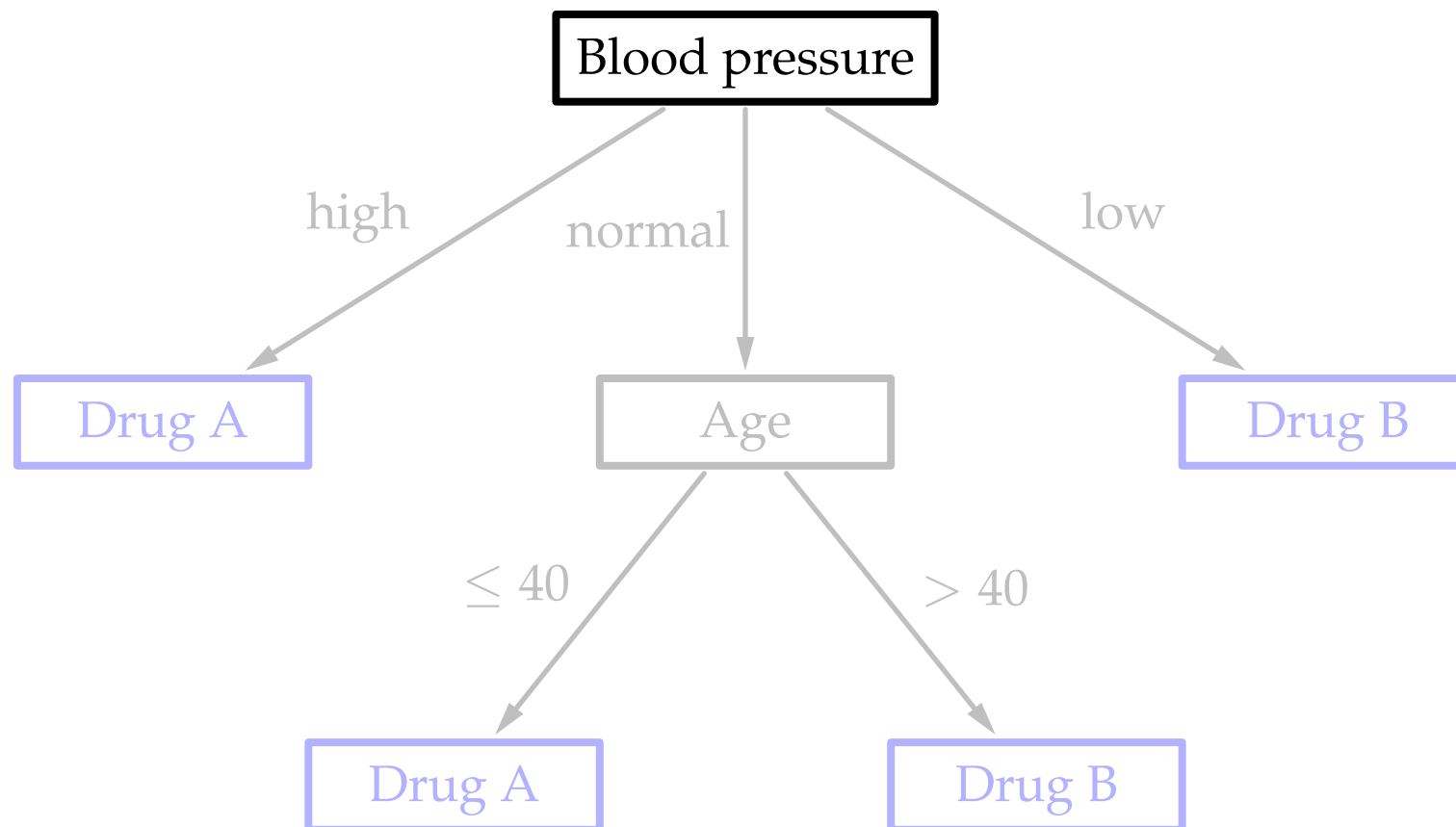
Intuitively: Follow the path corresponding to the case to be classified.

The followed path may be interpreted as a classification rule:

If the attributes tested in the inner nodes have the branch/edge labels as values, then the example/case belongs to the class of the leaf node reached by the path.

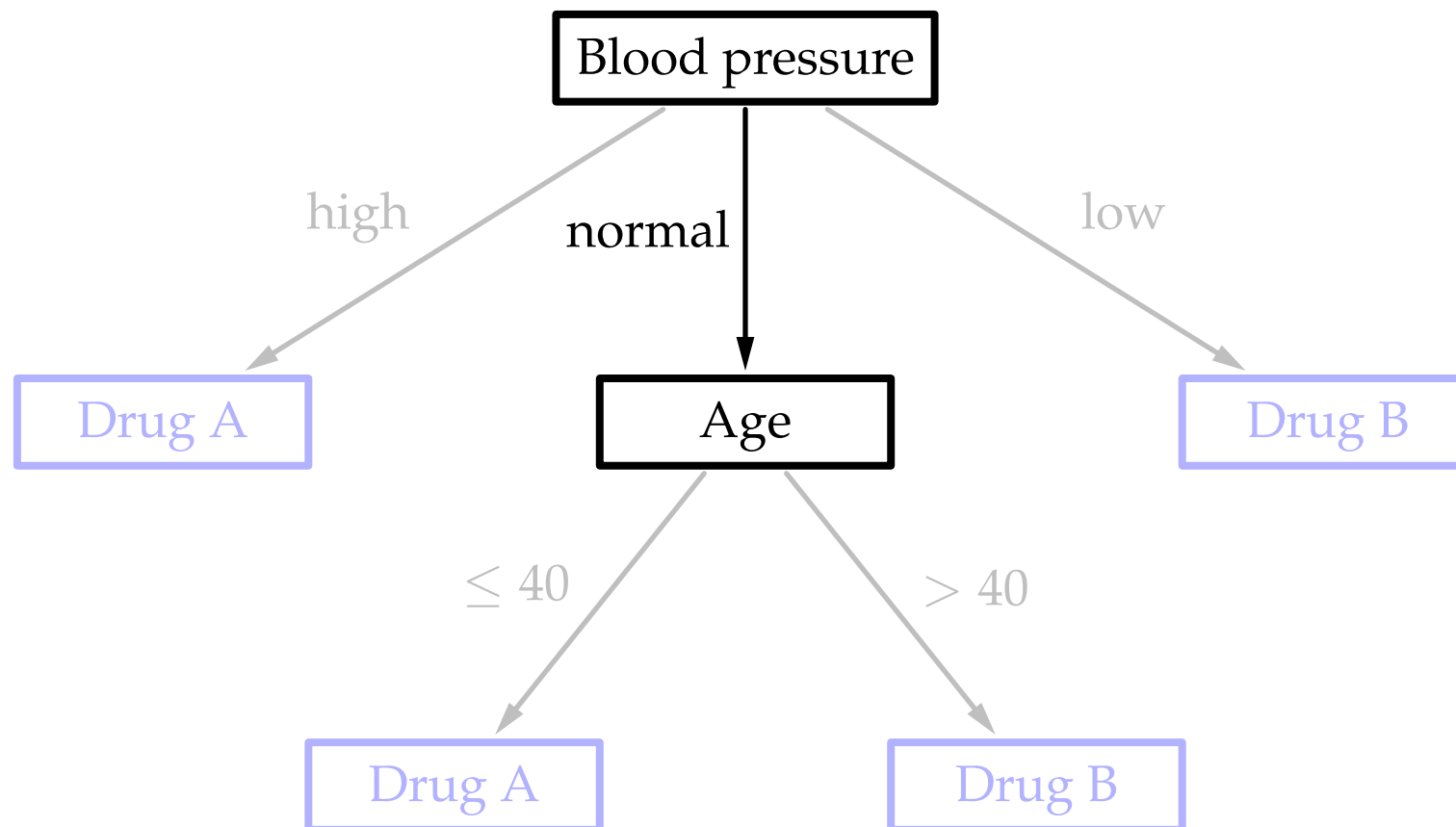
Classification in the Example

Assignment of a drug to a patient:



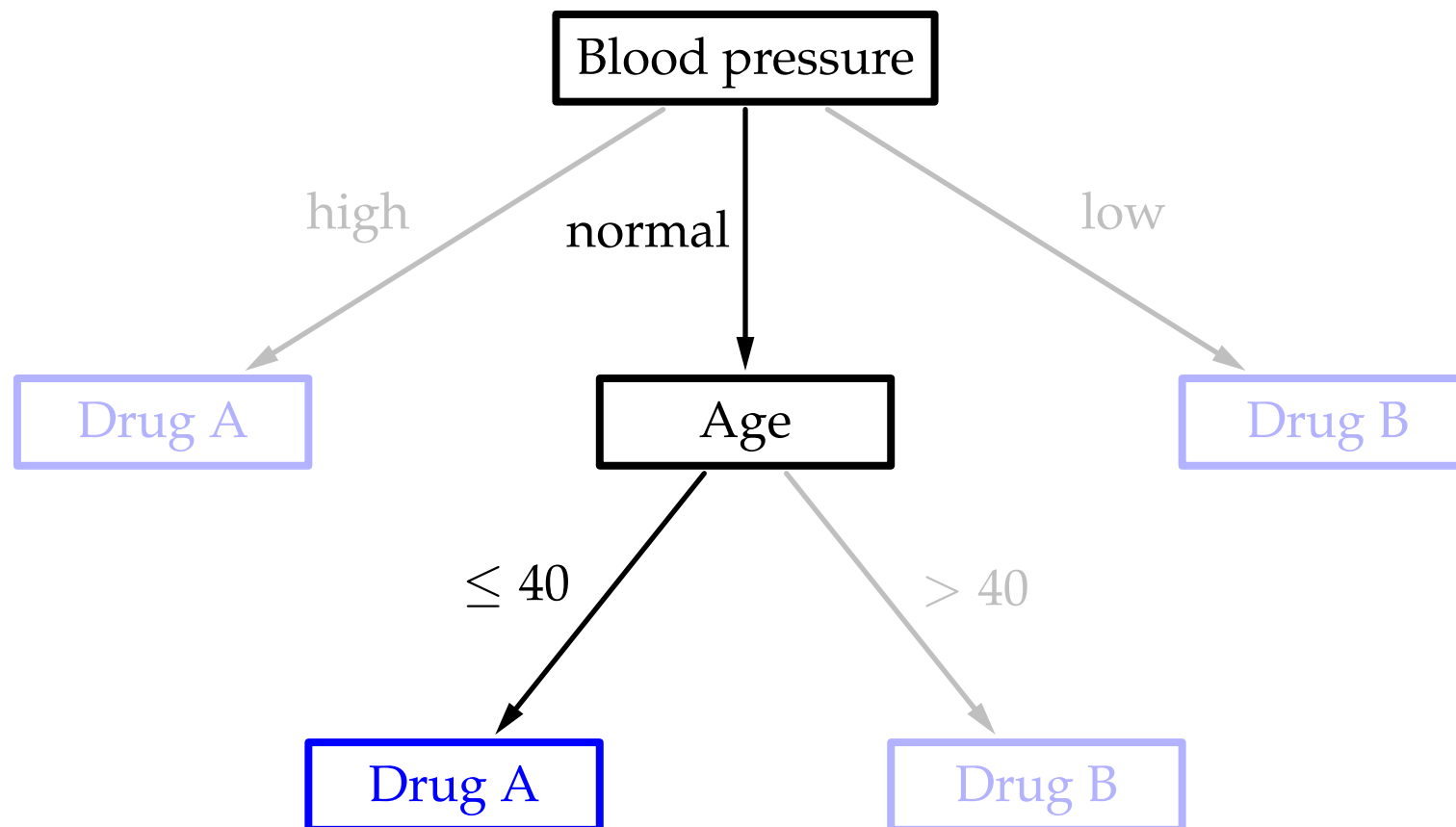
Classification in the Example

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** (TDIDT: top-down induction of decision trees)
 - 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 / tests.
 - Select the attribute / test 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 attributes / tests are available
 - some other termination criterion is satisfied

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	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

Induction of a Decision Tree: Example

Sex 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	Sex	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 threshold.
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

Age of the patient: Consider all possible thresholds and choose best.

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

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

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

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

...

- A metric attribute is turned into **multiple dichotomic / binary attributes** by considering all possible thresholds. The best threshold / attribute is chosen.
- The best threshold is never between cases with the same class.
(at least there is one of equal quality between cases of different classes).

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)

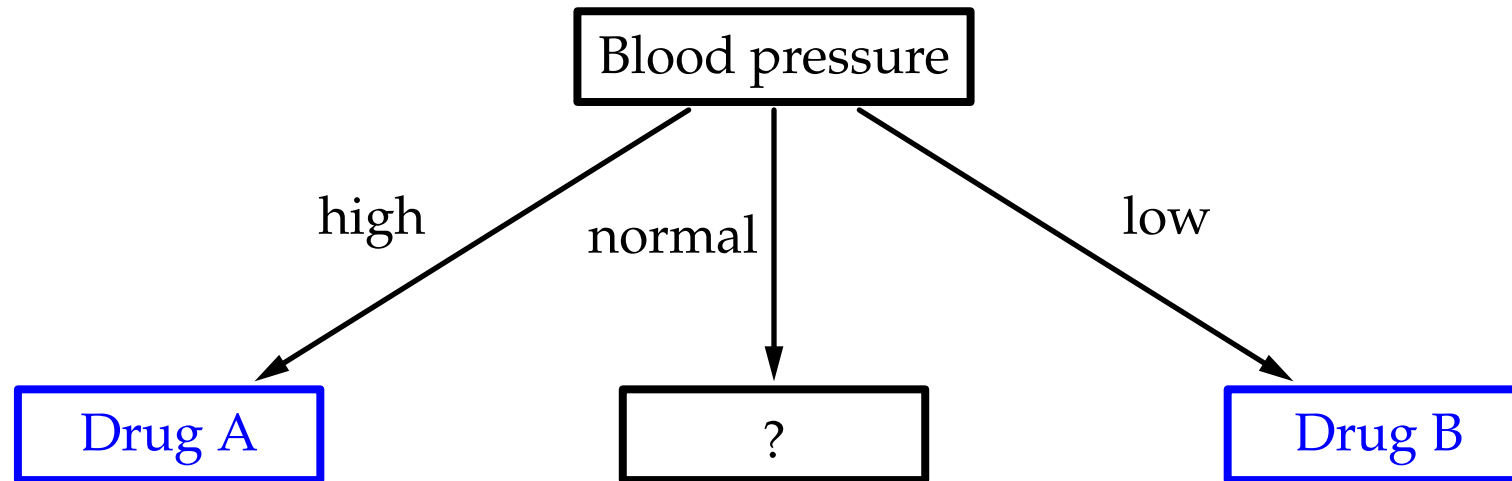
low: B 100% correct (in 3 of 3 cases)

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 Sex

- 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.	Sex	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 threshold.
here: ca. 40 years

Assignment of drug

≤ 40 : A 100% correct (3 of 3)

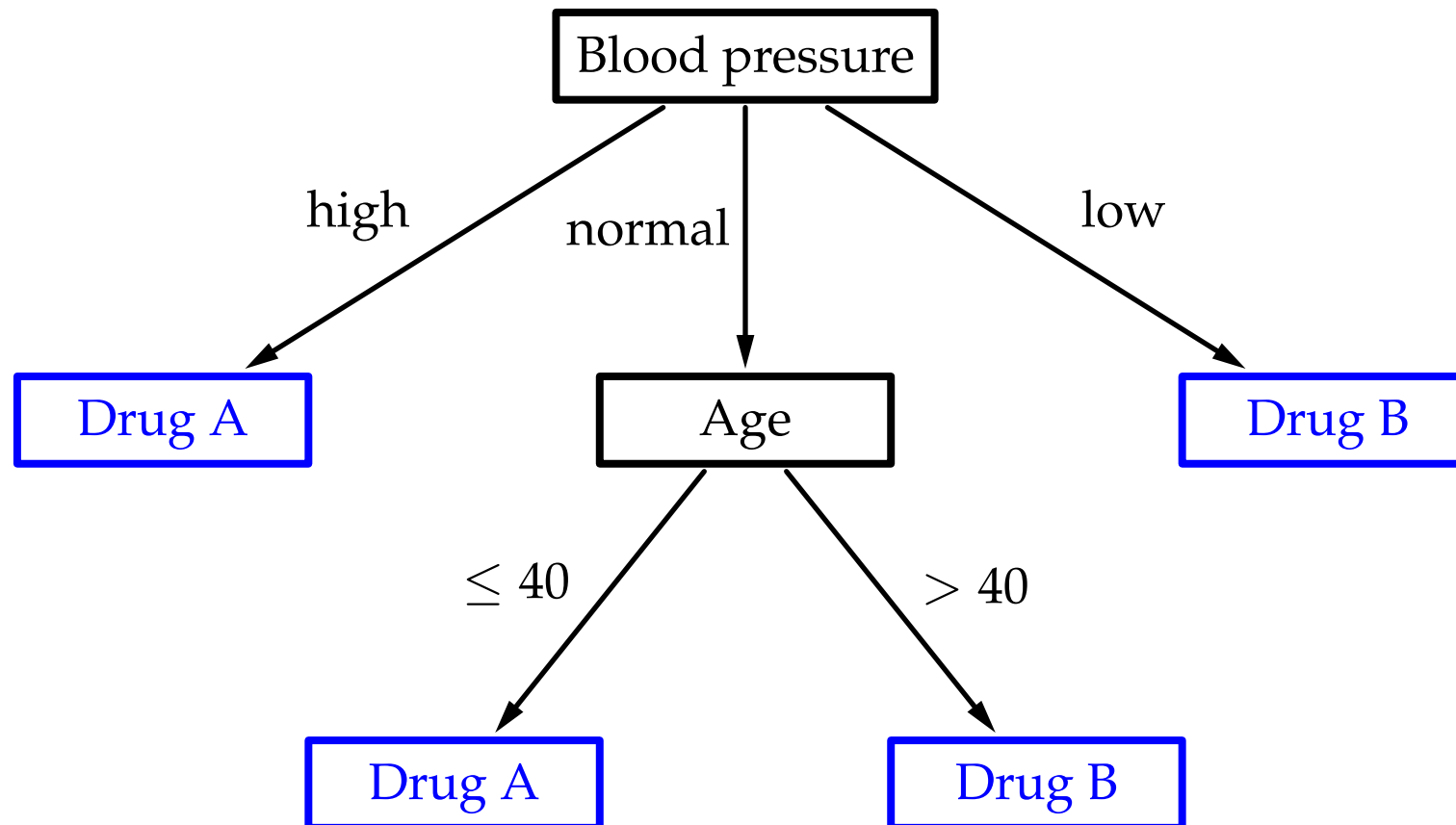
> 40 : B 100% correct (3 of 3)

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

D	a set of case or object descriptions (data)
C	the class attribute
$A^{(1)}, \dots, A^{(m)}$	other attributes (index dropped in the following)
$\text{dom}(C)$	$= \{c_1, \dots, c_{n_C}\}, \quad n_C: \text{number of classes}$
$\text{dom}(A)$	$= \{a_1, \dots, a_{n_A}\}, \quad n_A: \text{number of attribute values}$
$N_{..}$	total number of case or object descriptions i.e. $N_{..} = D $
$N_{i.}$	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: Notation

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

$C \triangleq \text{Drug}$

$\text{dom}(C) = \{A, B\}$

$A^{(1)} \triangleq \text{Sex}$

$\text{dom}(A^{(1)}) = \{\text{female}, \text{male}\}$

$A^{(2)} \triangleq \text{Age}$

$\text{dom}(A^{(2)}) = \mathbb{N}$ or $\text{dom}(A^{(2)}) = \mathbb{R}$

$A^{(3)} \triangleq \text{Blood pressure}$

$\text{dom}(A^{(3)}) = \{\text{low}, \text{normal}, \text{high}\}$

$N_{..} = 12$

$C : N_{1.} = 6, N_{2.} = 6$

$A^{(1)}: N_{.1} = 6, N_{.2} = 6,$

$N_{11} = 3, N_{12} = 3,$

$N_{21} = 3, N_{22} = 3.$

$A^{(2)}: (\text{depends on threshold/discretization})$

$A^{(3)}: N_{.1} = 3, N_{.2} = 6, N_{.3} = 3,$

$N_{11} = 3, N_{12} = 3, N_{13} = 0,$

$N_{21} = 0, N_{22} = 3, N_{23} = 3.$

Decision Tree Induction: General Algorithm

```
function grow_tree ( $D$  : set of cases) : node;  
begin  
     $best\_v :=$  WORTHLESS;  
    for all untested attributes  $A$  do  
        compute frequencies  $N_{ij}, N_{i.}, N_{.j}$  for  $1 \leq i \leq n_C$  and  $1 \leq j \leq n_A$ ;  
        compute value  $v$  of an evaluation measure using  $N_{ij}, N_{i.}, N_{.j}$ ;  
        if  $v > best\_v$  then  $best\_v := v$ ;  $best\_A := A$ ; end;  
    end  
    if  $best\_v =$  WORTHLESS  
    then create leaf node  $x$ ;  
        assign majority class of  $D$  to  $x$ ;  
    else create test node  $x$ ;  
        assign test on attribute  $best\_A$  to  $x$ ;  
        for all  $a \in \text{dom}(best\_A)$  do  $x.\text{child}[a] := \text{grow\_tree}(D|_{best\_A=a})$ ; 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 variants and normalizations.
 - χ^2 **measure** (well-known in statistics).

An Information-theoretic Evaluation Measure

Information Gain (used in ID3) [Kullback and Leibler 1951, Quinlan 1986]

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

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

$H(C)$ 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 $P_s(s_i), i = 1, \dots, n$, satisfying $\sum_{i=1}^n P_s(s_i) = 1$ (i.e., S is exhaustive).

- **Shannon Entropy:**

$$H(S) = - \sum_{i=1}^n P_s(s_i) \log_2 P_s(s_i) = \mathbb{E}_{s \sim P_s}(-\log_2 P_s(s))$$

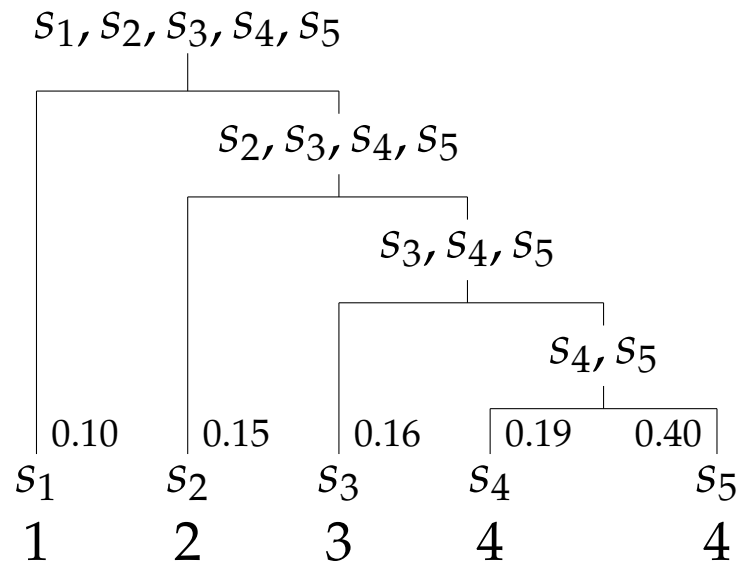
- **Intuitively: Expected number of yes/no questions that have to be asked in order to determine the obtaining alternative.**
 - Suppose there is an oracle, which knows the obtaining alternative, but responds only if the question can be answered with “yes” or “no”.
 - A better question scheme than asking 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

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

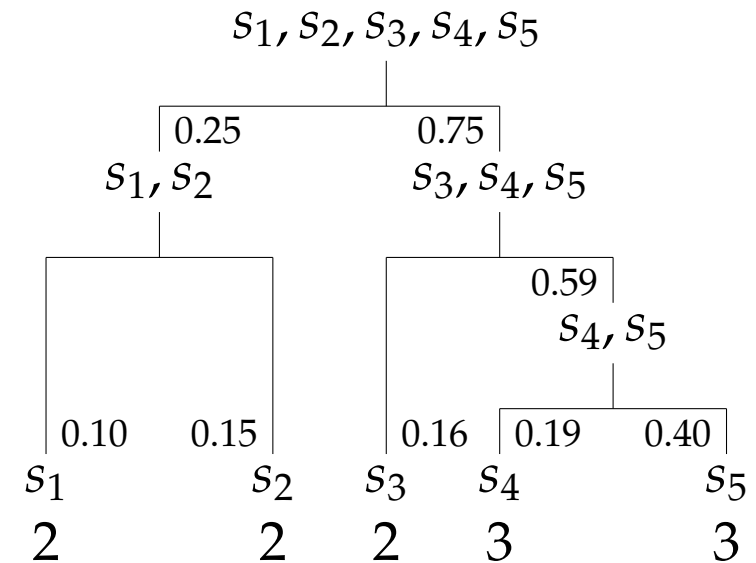
$$\text{Shannon entropy: } -\sum_i P_s(s_i) \log_2 P_s(s_i) = 2.15 \text{ bit/symbol}$$

Linear Traversal



Code length: 3.24 bit/symbol
Code efficiency: 0.664

Equal Size Subsets



Code length: 2.59 bit/symbol
Code efficiency: 0.830

Question/Coding Schemes

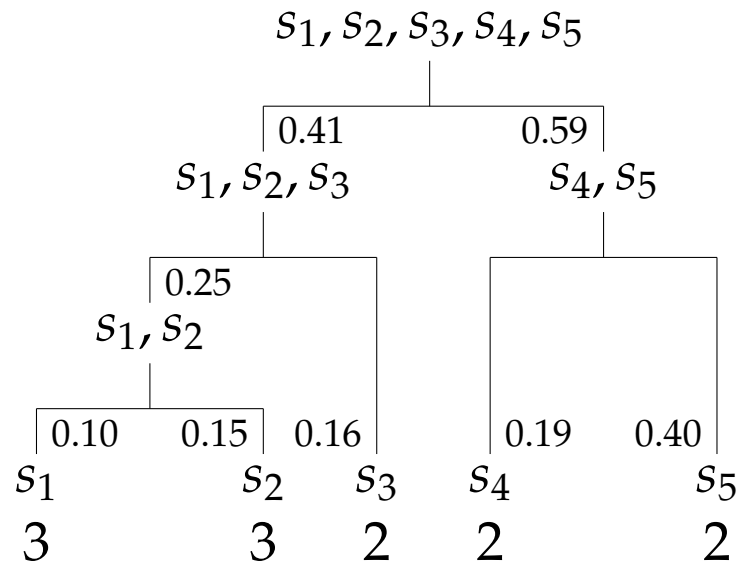
- Splitting into subsets of about equal size can lead to a bad arrangement of the alternatives into subsets → high expected number of questions.
- Good question schemes take the probability of the alternatives into account.
- **Shannon-Fano Coding** (1948)
 - Build the question/coding scheme top-down.
 - 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

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

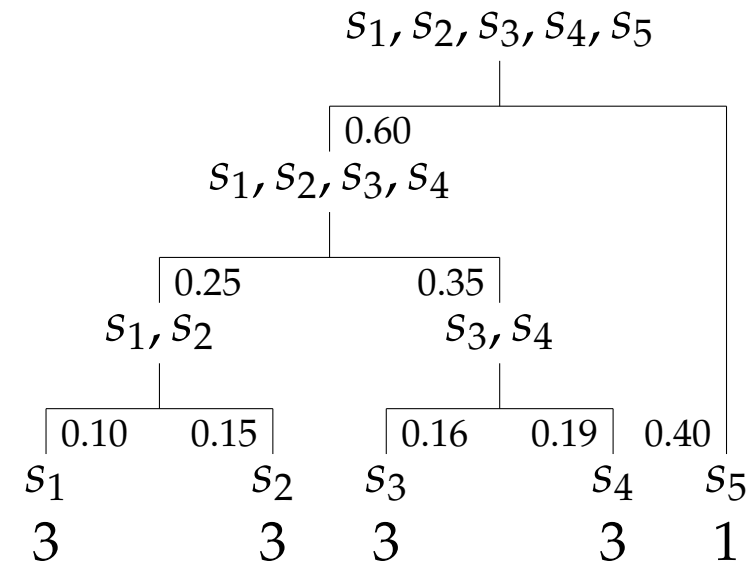
$$\text{Shannon entropy: } -\sum_i P_s(s_i) \log_2 P_s(s_i) = 2.15 \text{ bit/symbol}$$

Shannon–Fano Coding (1948)



Code length: 2.25 bit/symbol
Code efficiency: 0.955

Huffman Coding (1952)



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

$$P_s(s_1) = \frac{1}{2}, \quad P_s(s_2) = \frac{1}{4}, \quad P_s(s_3) = \frac{1}{8}, \quad P_s(s_4) = \frac{1}{16}, \quad P_s(s_5) = \frac{1}{16}$$

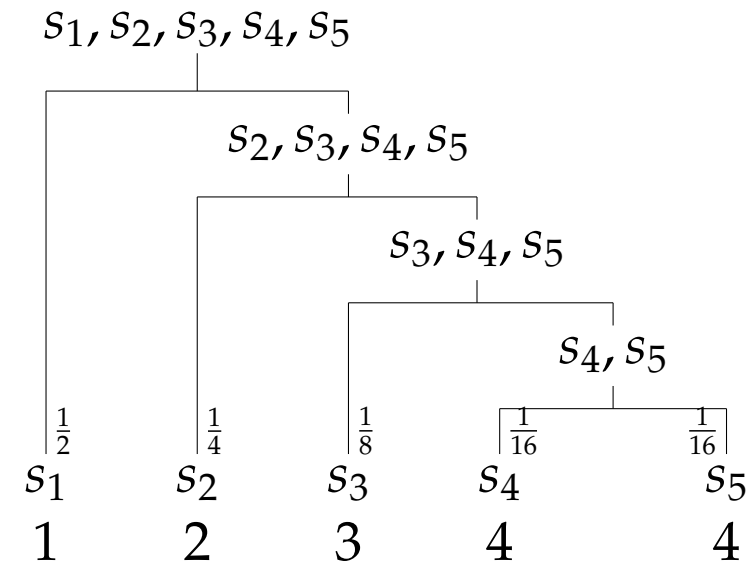
$$\text{Shannon entropy: } -\sum_i P_s(s_i) \log_2 P_s(s_i) = 1.875 \text{ bit/symbol}$$

If the probability distribution allows for a perfect Huffman code (code efficiency 1), the Shannon entropy can easily be interpreted as follows:

$$\begin{aligned} & -\sum_i P_s(s_i) \log_2 P_s(s_i) \\ &= \sum_i \underbrace{P_s(s_i)}_{\text{occurrence probability}} \cdot \underbrace{\log_2 \frac{1}{P_s(s_i)}}_{\text{path length in tree}}. \end{aligned}$$

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

Kullback–Leibler (Information) Divergence

- Let p_1 and p_2 be two strictly positive probability distributions on the same space \mathcal{E} of events. Then [Kullback and Leibler 1951]

$$I_{\text{KLdiv}}(p_1, p_2, \mathcal{E}) = \sum_{E \in \mathcal{E}} p_1(E) \log_2 \frac{p_1(E)}{p_2(E)} = \mathbb{E}_{E \sim p_1} \left(\log_2 \frac{p_1(E)}{p_2(E)} \right)$$

is called the **Kullback-Leibler (information) divergence** of p_1 and p_2 .

- The Kullback-Leibler information divergence is non-negative.
- It is zero if and only if $p_1 \equiv p_2$; it is generally *not* symmetric.
- Information gain** is the Kullback–Leibler information divergence of $p_1(c_i, a_j) = p_{ij}$ and $p_2(c_i, a_j) = p_{i.} p_{.j}$ for $\mathcal{E} = \{c_1, \dots, c_{n_C}\} \times \{a_1, \dots, a_{n_A}\}$.

$$\begin{aligned} I_{\text{gain}}^{(\text{Shannon})}(C, A) &= - \sum_{i=1}^{n_C} p_{i.} \log_2 p_{i.} - \sum_{j=1}^{n_A} p_{.j} \left(- \sum_{i=1}^{n_C} p_{i|j} \log_2 p_{i|j} \right) \\ &= \sum_{i=1}^{n_C} \sum_{j=1}^{n_A} p_{ij} \log_2 \frac{p_{ij}}{p_{i.} p_{.j}} = \mathbb{E}_{(c_i, a_j) \sim P_{CA}} \left(\log_2 \frac{p_{ij}}{p_{i.} p_{.j}} \right) \end{aligned}$$

Other Information-theoretic Evaluation Measures

Normalized Information Gain

- Information gain is biased towards many-valued attributes.
- Normalization removes / reduces this bias.

Information Gain Ratio (used in C4.5)

[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]

$$I_{\text{sgr}}^{(1)}(C, A) = \frac{I_{\text{gain}}(C, A)}{H(A, C)} \quad \text{or} \quad I_{\text{sgr}}^{(2)}(C, A) = \frac{I_{\text{gain}}(C, A)}{H(A) + H(C)}$$

“Symmetric” means that $I_{\text{sgr}}^{(1)}(C, A) = I_{\text{sgr}}^{(1)}(A, C)$ and $I_{\text{sgr}}^{(2)}(C, A) = I_{\text{sgr}}^{(2)}(A, C)$.

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 it is always

$$I_{\text{gain}}(C, AB) \geq I_{\text{gain}}(C, B),$$

with equality holding only if the attributes C and A are conditionally independent given B , i.e., if $P(C = c \mid A = a, B = b) = P(C = c \mid B = b)$.

(A detailed proof of this theorem can be found, for example, in [Borgelt and Kruse 2002], p. 311ff.)

Generalized Entropy and Gini Impurity

- Shannon entropy can be generalized, for example, as: [Daróczy 1970]

$$H_{\beta}^{(\text{general})} = \frac{2^{\beta-1}}{2^{\beta-1} - 1} \sum_{i=1}^n p_i (1 - p_i^{\beta-1}) = \frac{2^{\beta-1}}{2^{\beta-1} - 1} \left(1 - \sum_{i=1}^n p_i^{\beta} \right)$$

- Shannon entropy** results as a special case in the limit for $\beta \rightarrow 1$:

$$H^{(\text{Shannon})} = \lim_{\beta \rightarrow 1} H_{\beta}^{(\text{general})} = - \sum_{i=1}^n p_i \log_2 p_i$$

- Another commonly used variant is **quadratic entropy**, which results for $\beta = 2$:

$$H^{(\text{quad})} = H_2^{(\text{general})} = 2 \sum_{i=1}^n p_i (1 - p_i) = 2 - 2 \sum_{i=1}^n p_i^2$$

- Intuitively: choosing each alternative s_i with its probability p_i leads to:

$$P(\text{wrong guess}) = \sum_{i=1}^n p_i (1 - p_i) = 1 - \sum_{i=1}^n p_i^2 = \frac{1}{2} H^{(\text{quad})} = I_{\text{Gini}}$$

This is also known as **Gini impurity** (= half of quadratic entropy).

Evaluation Measures based on Quadratic Entropy

- **Shannon information gain** may be written in two ways:

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

- For quadratic entropy the two versions differ: [Breiman, Friedman, Olshen & Stone 1984]
Using the former variant leads to the **Gini index**: (used in CART)

$$\text{Gini}(C, A) = \frac{1}{2} \left(H^{(\text{quad})}(C) - H^{(\text{quad})}(C|A) \right) = \sum_{j=1}^{n_A} p_{.j} \sum_{i=1}^{n_C} p_{i|j}^2 - \sum_{i=1}^{n_C} p_{i.}^2$$

- Using the latter leads to what may be called **quadratic information gain**:

$$I_{\text{gain}}^{(\text{quad})}(C, A) = H^{(\text{quad})}(C) + H^{(\text{quad})}(A) - H^{(\text{quad})}(CA)$$

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.

$$\chi^2(C, A) = N_{..} \sum_{i=1}^{n_C} \sum_{j=1}^{n_A} \frac{(p_{i.} p_{.j} - p_{ij})^2}{p_{i.} p_{.j}} = \sum_{i=1}^{n_C} \sum_{j=1}^{n_A} \frac{(N_{i.} N_{.j} - N_{ij} N_{..})^2}{N_{i.} N_{.j} N_{..}}$$

- Reminder: Information gain can also be interpreted as a difference measure.

$$I_{\text{gain}}(C, A) = \sum_{i=1}^{n_C} \sum_{j=1}^{n_A} p_{ij} \log_2 \frac{p_{ij}}{p_{i.} p_{.j}} = \sum_{i=1}^{n_C} \sum_{j=1}^{n_A} p_{ij} (\log_2 p_{ij} - \log_2 (p_{i.} p_{.j}))$$

Information gain uses relative comparison (ratio or difference of logarithms).

General Approach: Discretization

- **Preprocessing I**

- Form equally sized or equally populated intervals.

- **During the tree construction**

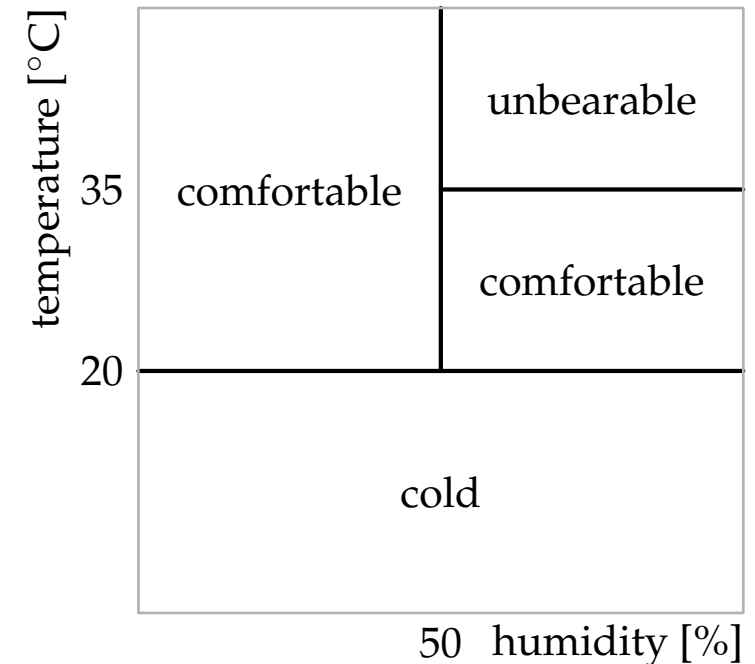
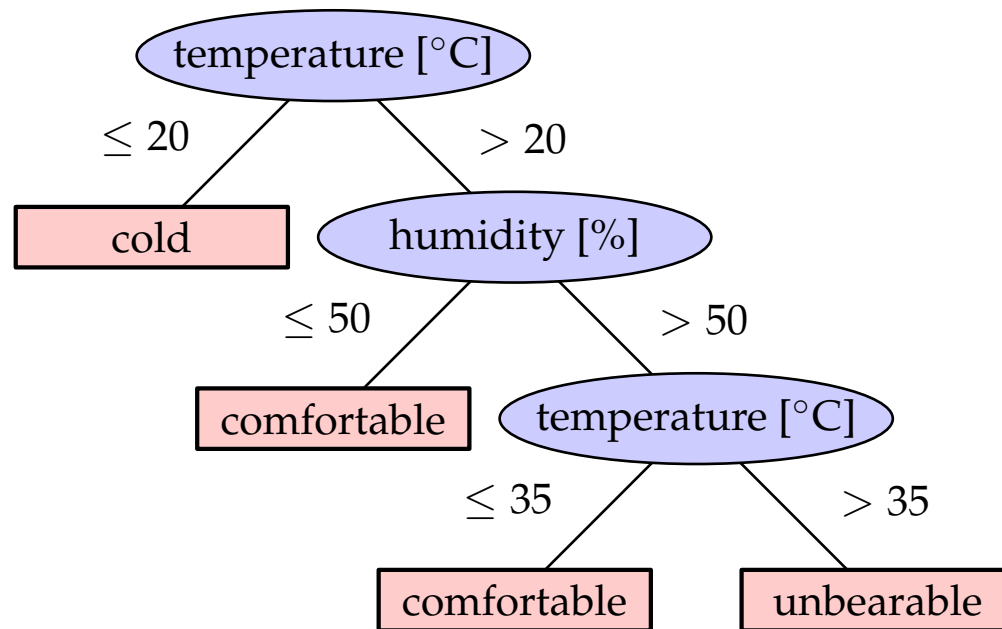
- Sort the example cases according to the attribute's values.
- Construct a binary symbolic attribute for every possible split (values: " \leq threshold" and " $>$ threshold").
- Compute the evaluation measure for these binary attributes.
- Possible improvements: Add a penalty depending on the number 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.

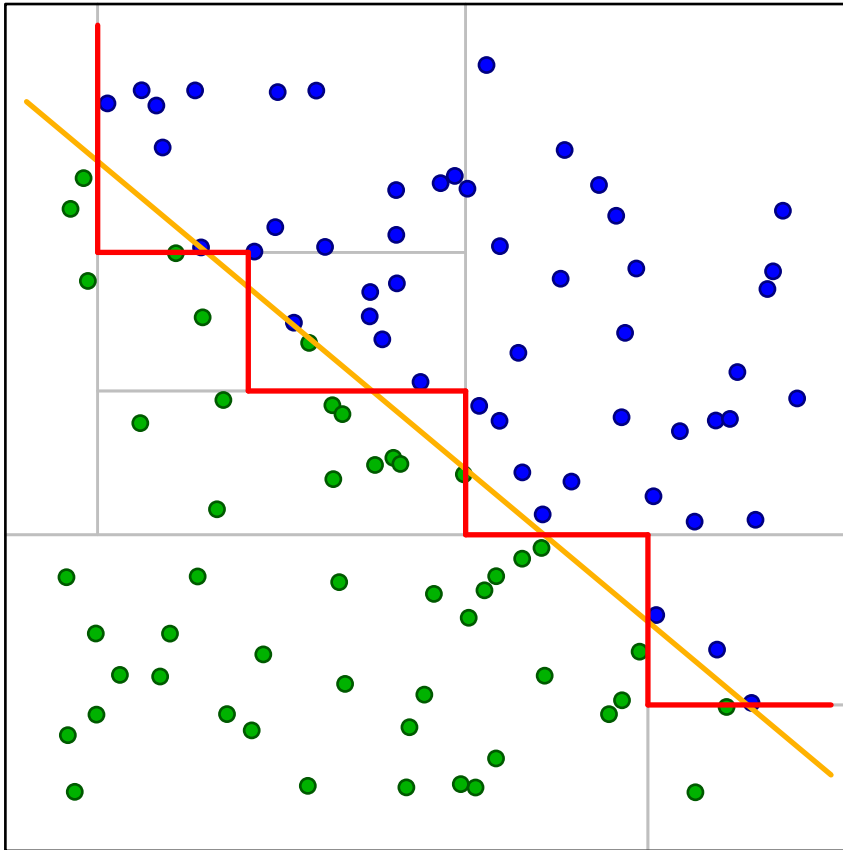
Data Space Partitioning

A decision tree partitions the data space.



- Each inner node (test node) cuts a dimension at the given threshold.
- If there are parent / ancestor nodes, only a subspace is cut.
(The “humidity” test only cuts above “temperature = 20°”,
the bottom “temperature” test only to the right of “humidity = 50%”).

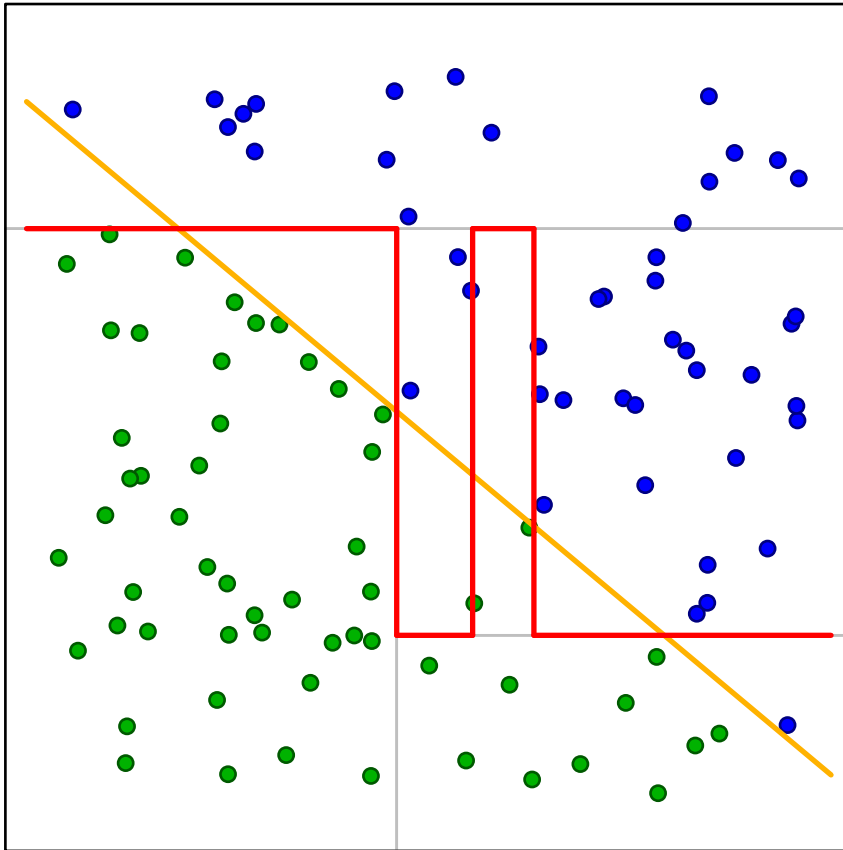
Treatment of Numeric Attributes



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 suboptimal, 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

Treatment of Missing Values

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 decision trees serves the purpose

- to simplify the tree (improve interpretability),
- to avoid overfitting (improve generalization).

Basic ideas:

- Replace “bad” branches (subtrees) by leaves (from bottom/leaves upward).
- Replace a subtree by its largest branch if it is better.

Common approaches:

- Limiting the number of leaf cases
- Reduced error pruning
- Pessimistic pruning
- Confidence level pruning
- Minimum description length pruning

Limiting the Number of Leaf Cases

- A decision tree may be grown until either the set of sample cases is class-pure or the set of descriptive attributes is exhausted.
- However, this may lead to leaves that capture only very few, in extreme cases even just a single sample case.
- Thus a decision tree may become very similar to a 1-nearest-neighbor classifier.
- In order to prevent such results, it is common to let a user specify a **minimum number of sample cases per leaf**.
- In such an approach, splits are usually limited to binary splits.
(nominal attributes: usually one attribute value against all others)
- A split is then adopted only if on both sides of the split at least the minimum number of sample cases are present.
- Note that this approach is not an actual pruning method, as it is already applied during induction, not after.

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

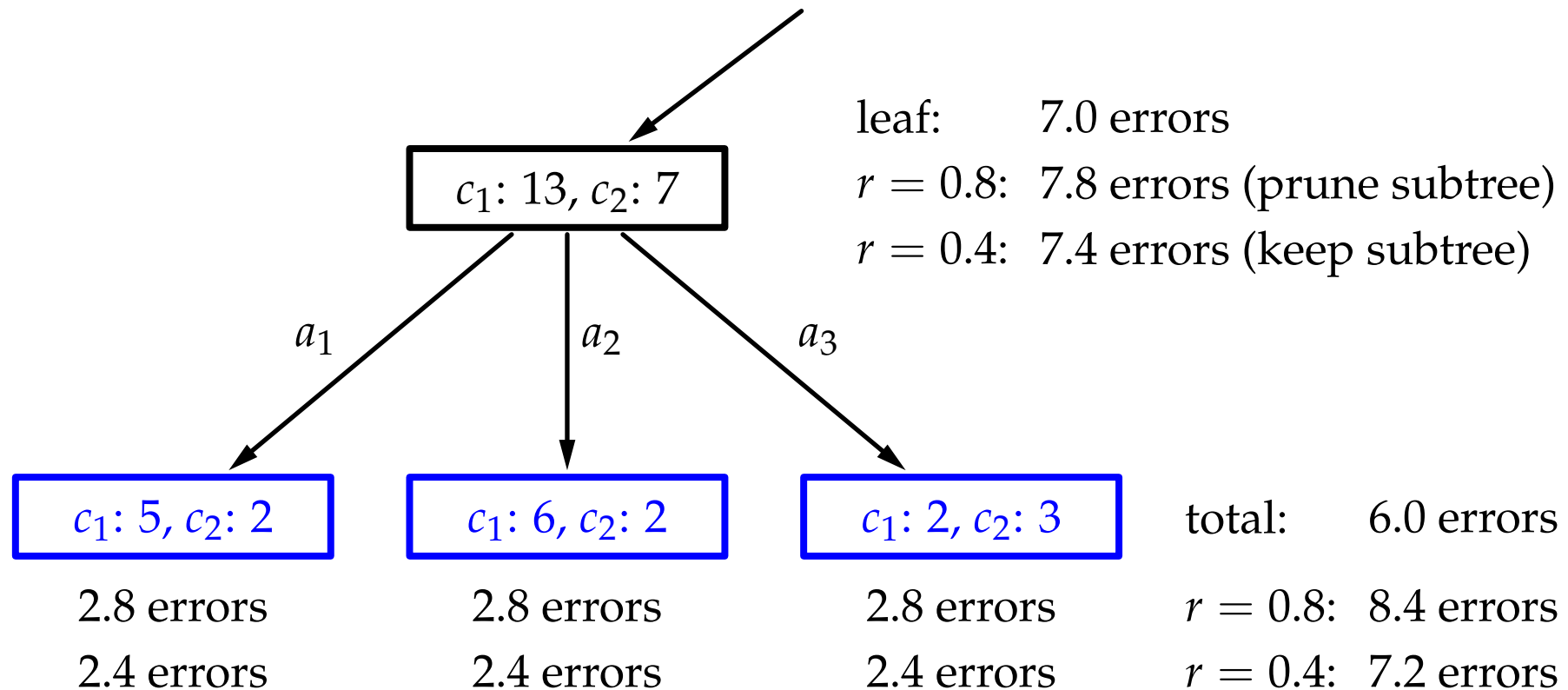
- Like pessimistic pruning, but the number of errors is computed as follows:
 - See classification in a leaf as a Bernoulli experiment (error / no error).
 - Estimate an interval for the error probability based on a user-specified confidence level α .
(use approximation of the binomial distribution by 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 $r = 0.4$:



- **Confidence level pruning** follows essentially the same scheme, but estimates the expected number of errors by increasing the observed number to the upper bound of a confidence interval.

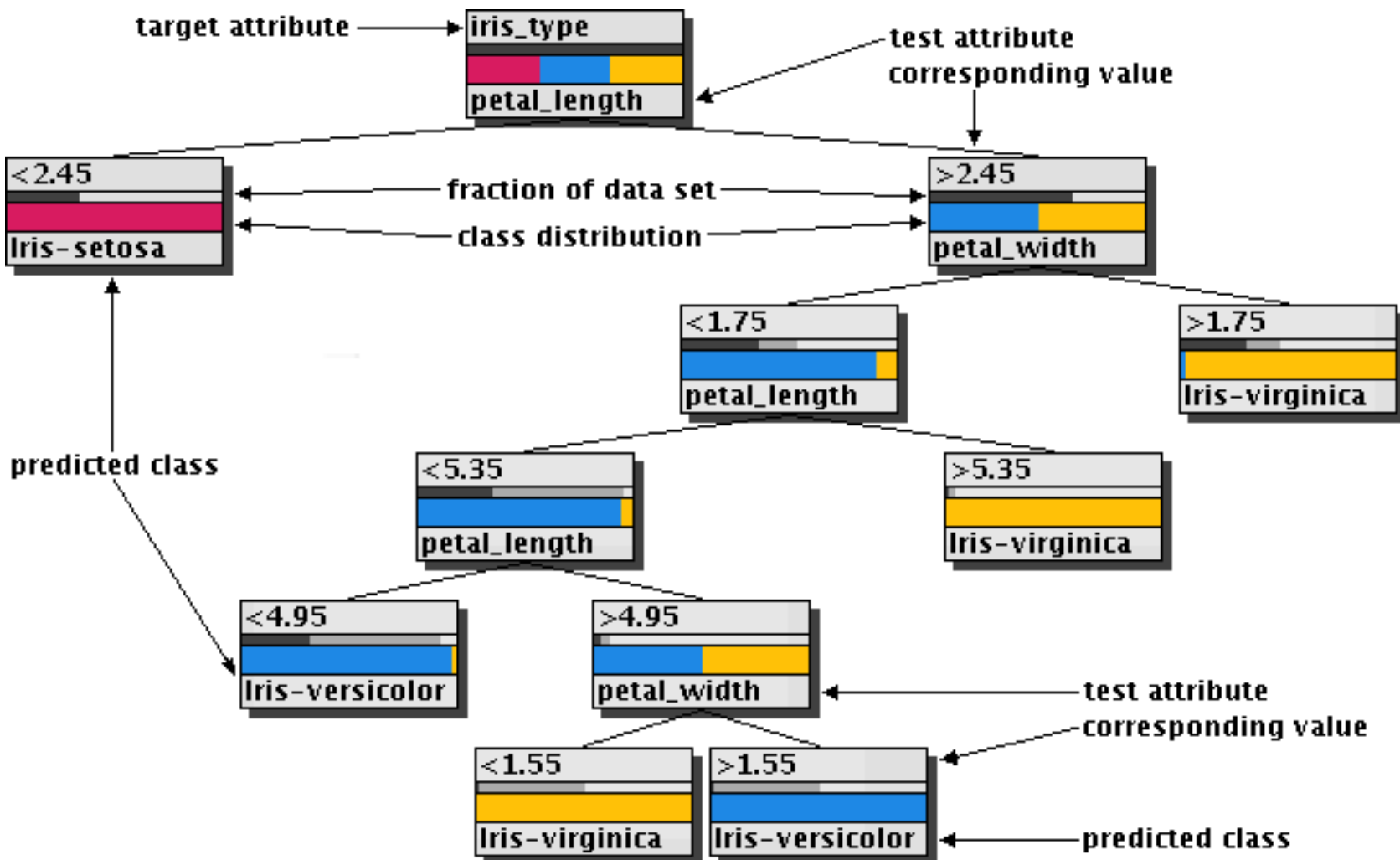
Reminder: The Iris Data

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.

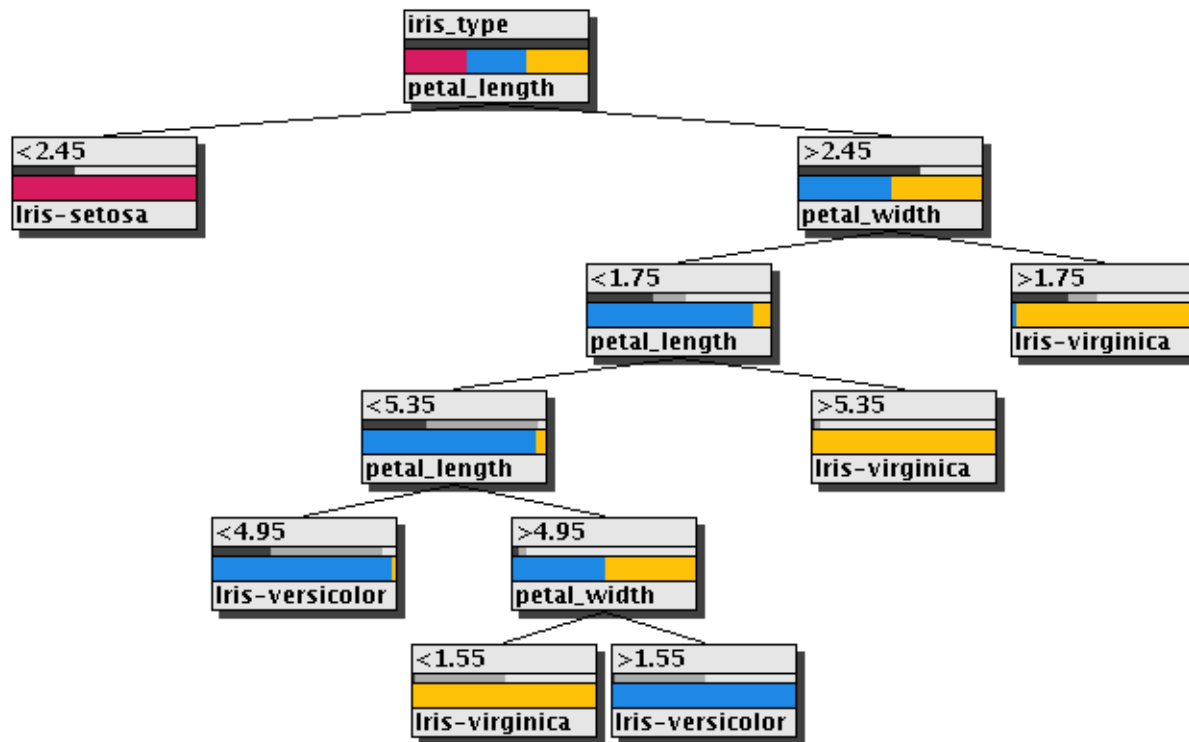
Decision Trees: An Example

A decision tree for the Iris data
(induced with information gain ratio, unpruned)

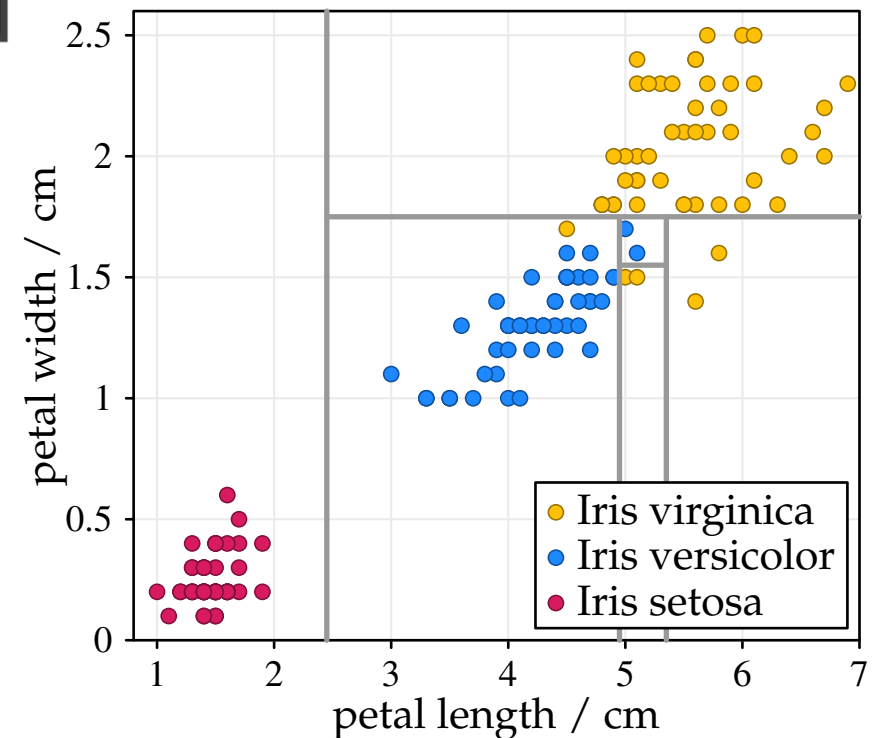


Decision Trees: An Example

A decision tree for the Iris data
(induced with information gain ratio, unpruned)



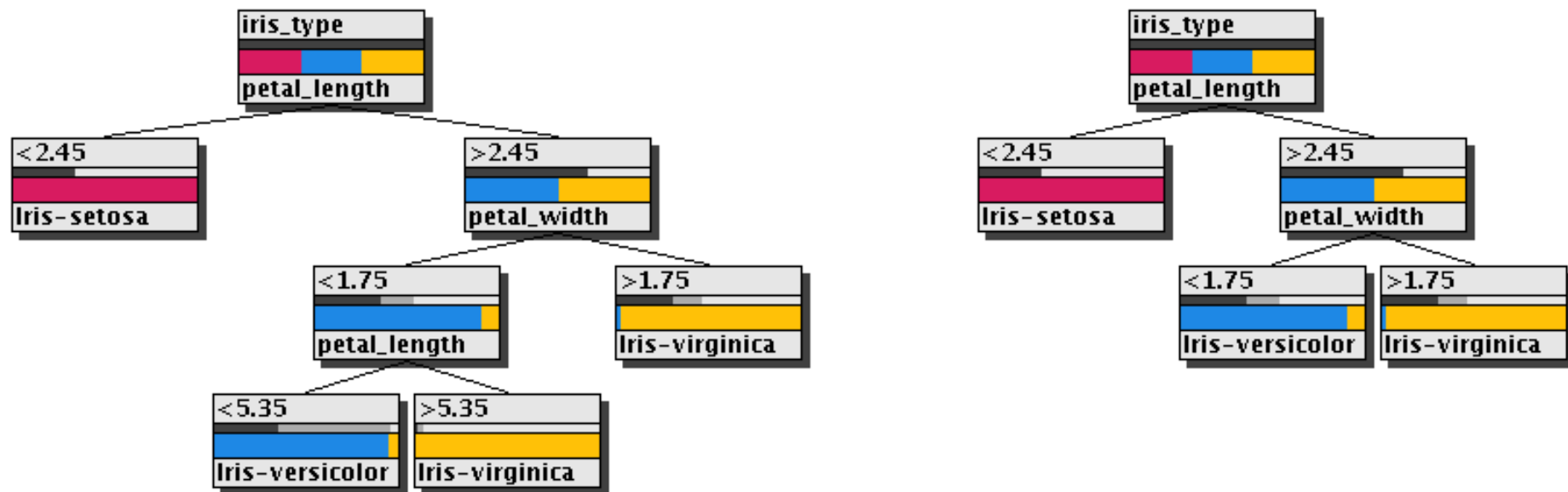
partitioning of the data space



Decision Trees: An Example

Decision trees for the Iris data

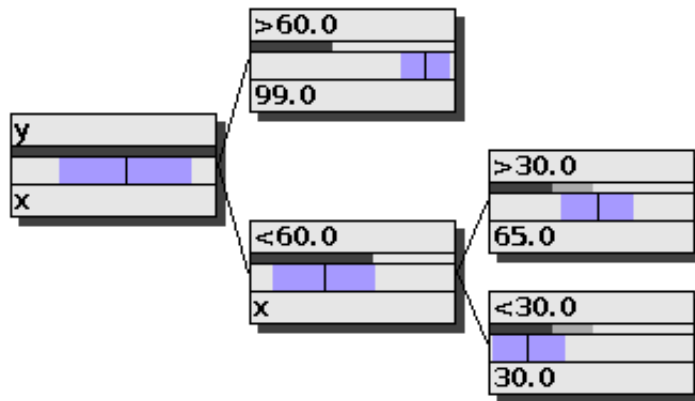
(pruned with confidence level pruning, $\alpha = 0.8$, and pessimistic pruning, $r = 2$)



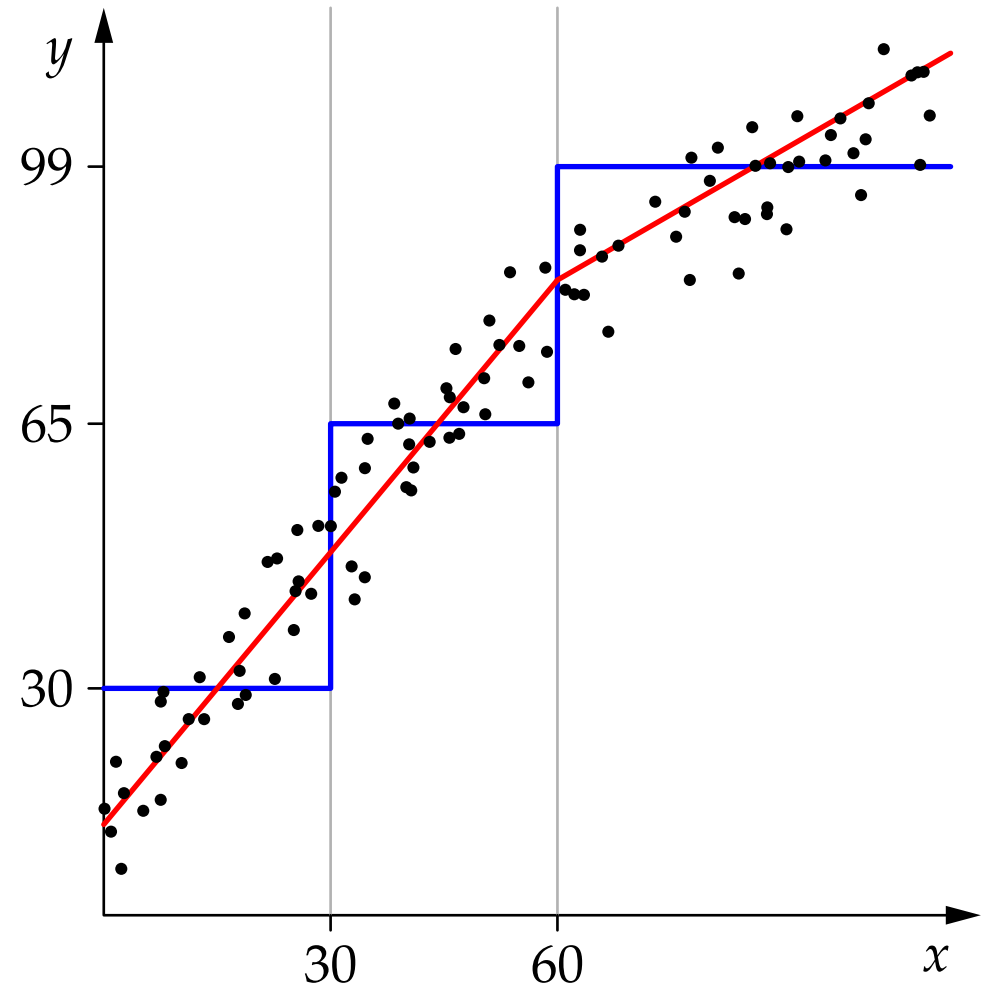
- Left: 7 instead of 11 nodes, 4 instead of 2 misclassifications.
- Right: 5 instead of 11 nodes, 6 instead of 2 misclassifications.
- The right tree is “minimal” for the three classes (only three leaf nodes).

Regression Trees

- Target variable is not a class, but a numeric quantity.
- Simple regression trees: predict constant values in leaves. (blue lines)

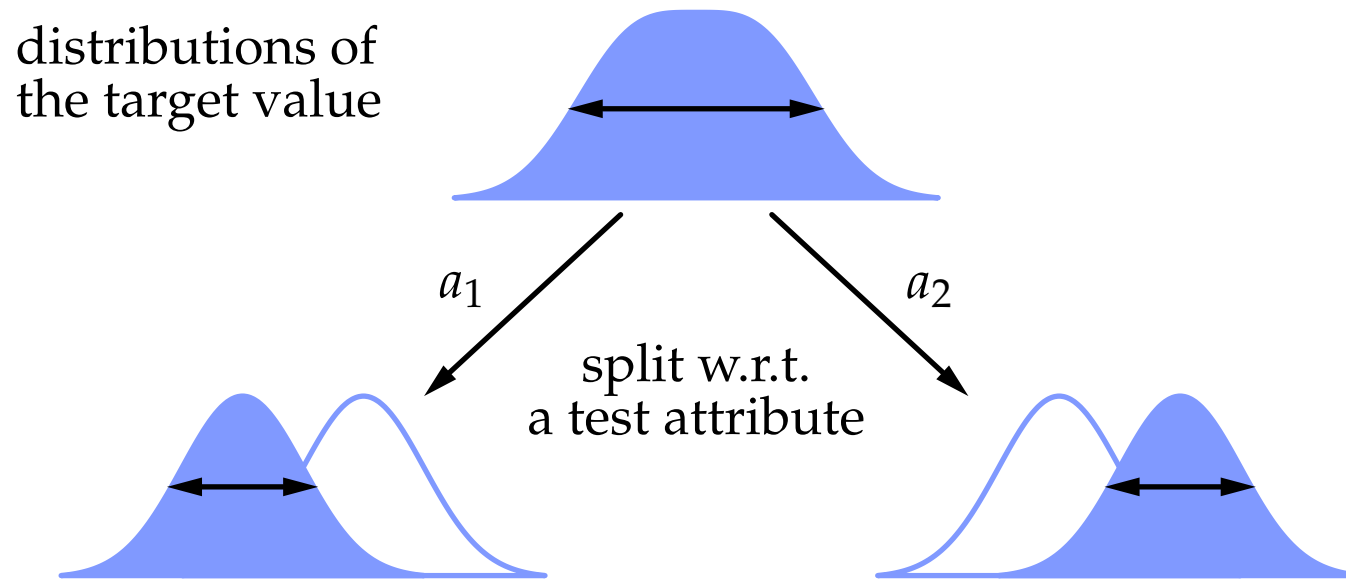


- More complex regression trees: predict linear functions in leaves. (red line)



x : input variable, y : target variable

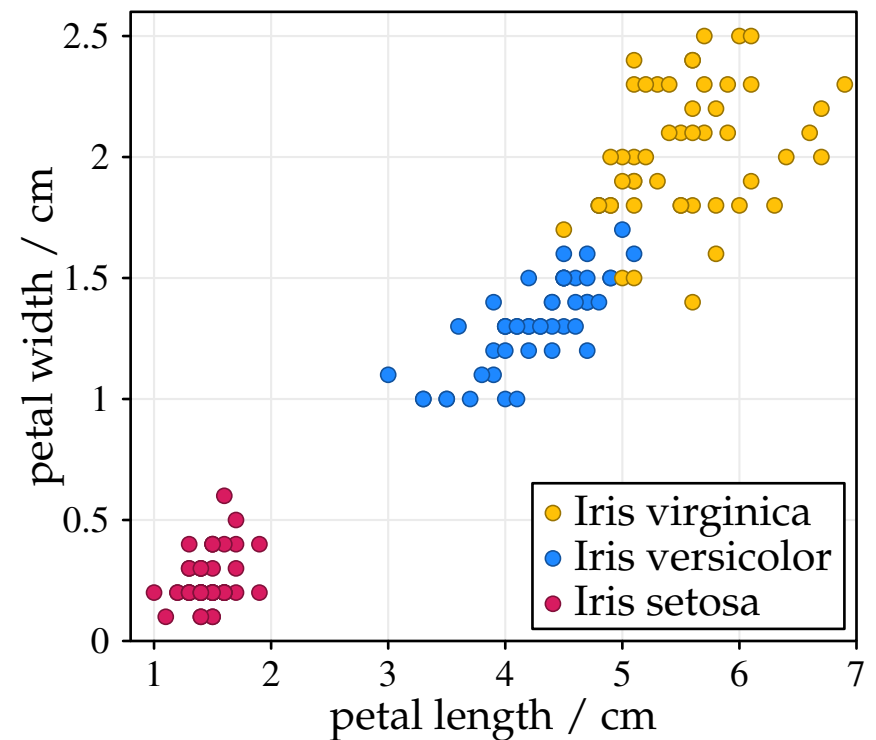
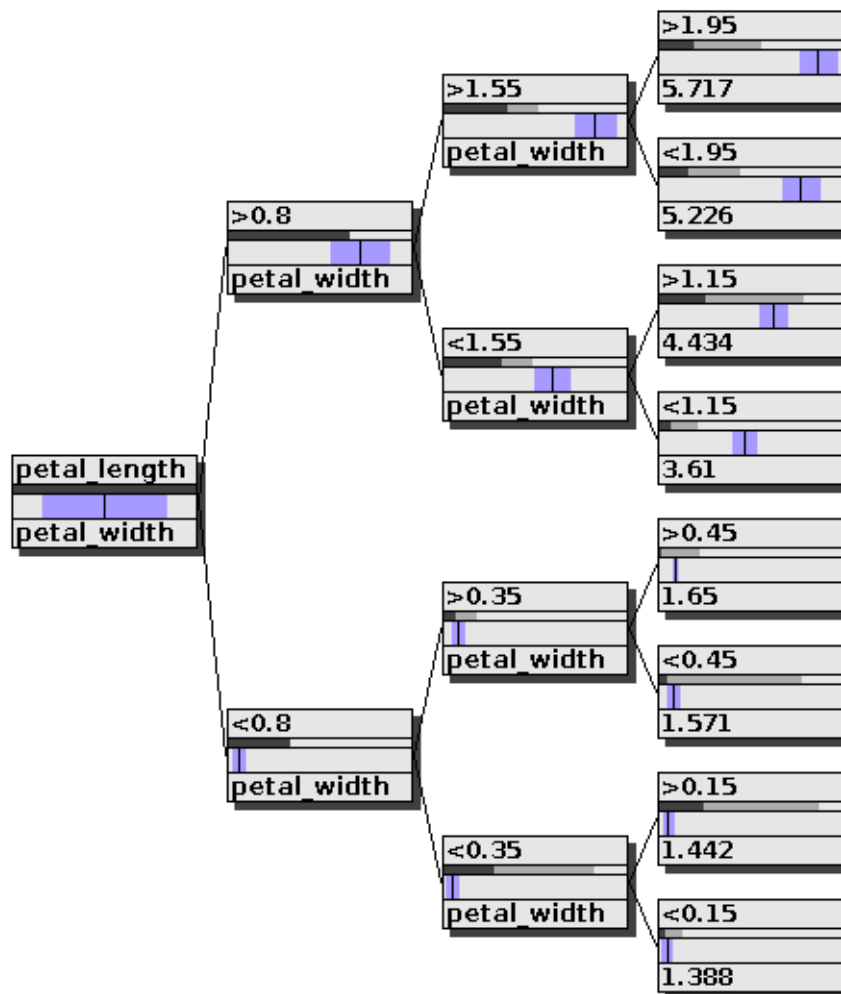
Regression Trees: Attribute Selection



- The variance / standard deviation is compared to the variance / standard deviation in the branches.
- The attribute that yields the largest reduction is selected.

Regression Trees: An Example

A regression tree for the Iris data (petal width \rightarrow petal length)
(induced with reduction of sum of squared errors; restricted to height 4)



Summary Decision and Regression Trees

- **Decision Trees are Classifiers with Tree Structure**

- Inner node: Test of a descriptive attribute (“split rule”)
- Edge/branch: Labeled with the outcome of a test
- 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 / split rules
- Attributes are selected based on an *evaluation measure*, e.g. information gain (ratio), χ^2 measure, Gini index
- Recommended: *Pruning* of the decision tree

- **Numeric Target: Regression Trees**

Clustering

Clustering

- **General Idea of Clustering**
 - Similarity and distance measures
- **Prototype-based Clustering**
 - Classical c -means (or k -means) clustering
 - Cluster center initialization
 - (Fuzzy c -means clustering) \Rightarrow Advanced Data Mining 2
 - (Expectation maximization for Gaussian mixtures) \Rightarrow Adv. DM 2
- **Hierarchical Agglomerative Clustering**
 - Merging clusters: Dendrograms
 - 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 $\forall \vec{x}, \vec{y}, \vec{z} \in \mathbb{R}^m :$

$$(i) \quad d(\vec{x}, \vec{y}) = 0 \iff \vec{x} = \vec{y},$$

$$(ii) \quad d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x}) \quad (\text{symmetry}),$$

$$(iii) \quad d(\vec{x}, \vec{z}) \leq d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z}) \quad (\text{triangle inequality}).$$

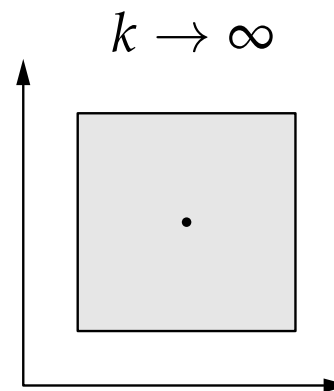
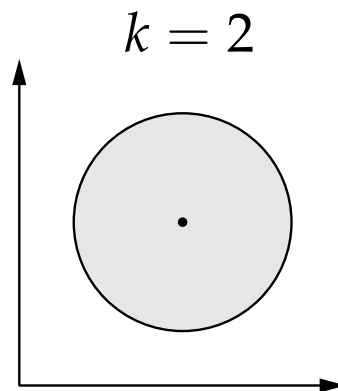
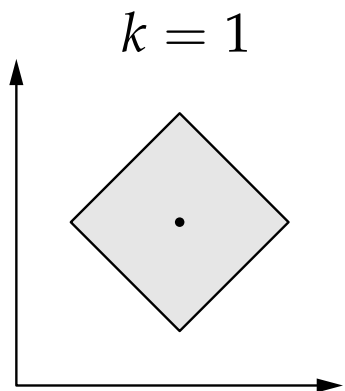
Distance Functions

Illustration of distance functions: Minkowski Family

$$d_k(\vec{x}, \vec{y}) = \sqrt[k]{\sum_{i=1}^m |x_i - y_i|^k} = \left(\sum_{i=1}^m |x_i - y_i|^k \right)^{\frac{1}{k}}$$

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. $d_\infty(\vec{x}, \vec{y}) = \max_{i=1}^n |x_i - y_i|$.

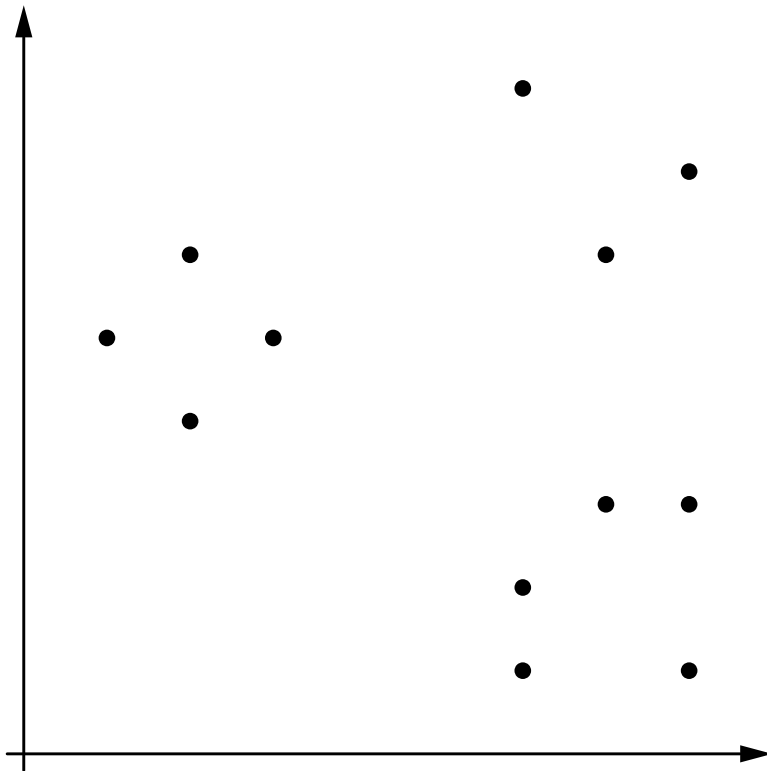


circle: set of points having the same distance from a given point, called the center.

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 mass/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.

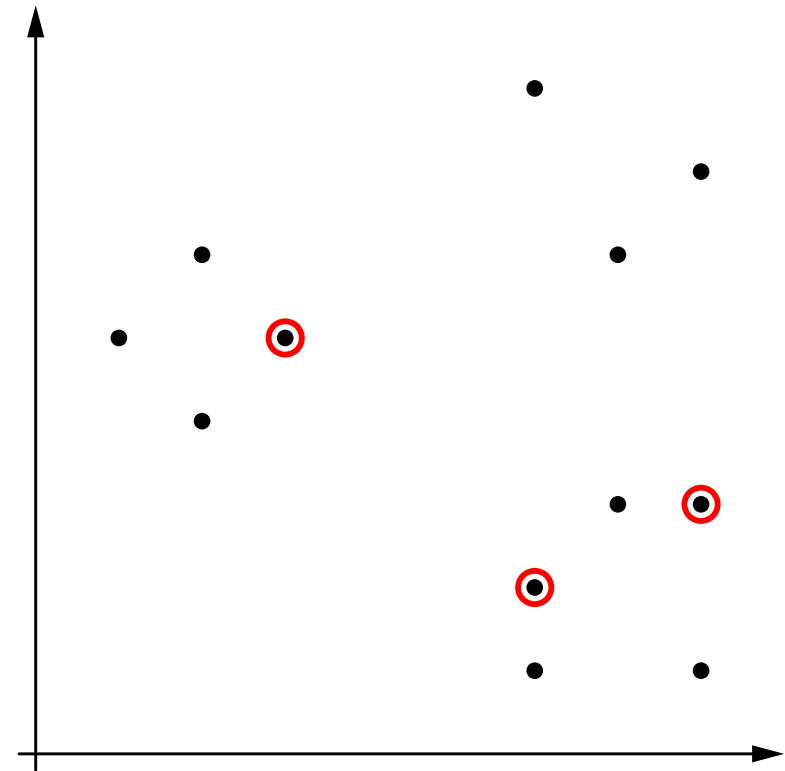
c -Means Clustering: Example



Data set to cluster.

Choose $c = 3$ clusters.

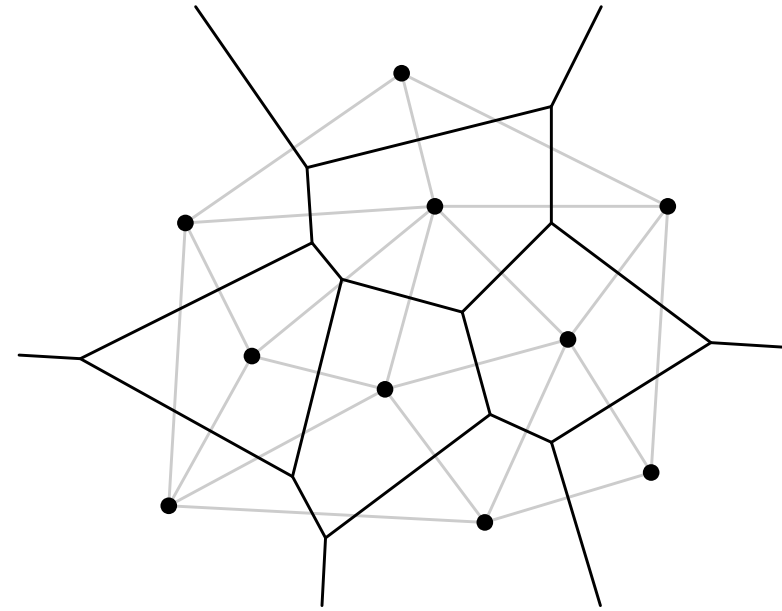
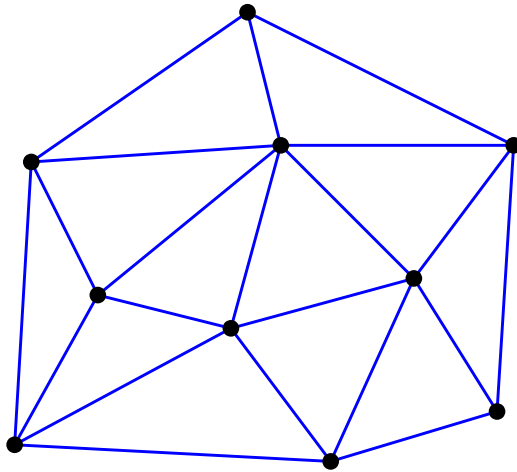
(From visual inspection, can be difficult to determine in general.)



Initial position of cluster centers.

Randomly selected data points.
(Alternative methods include
e.g. latin hypercube sampling)

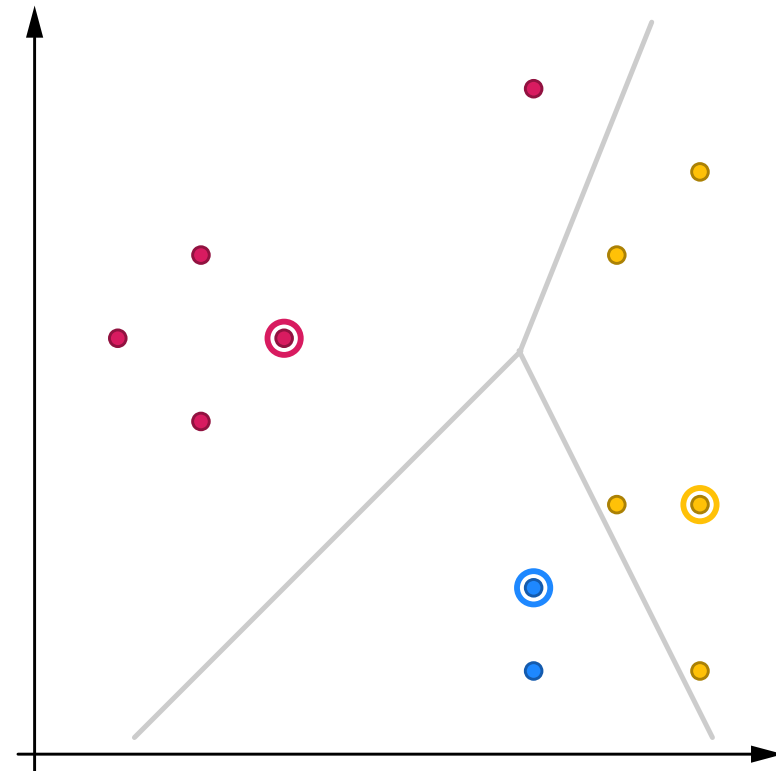
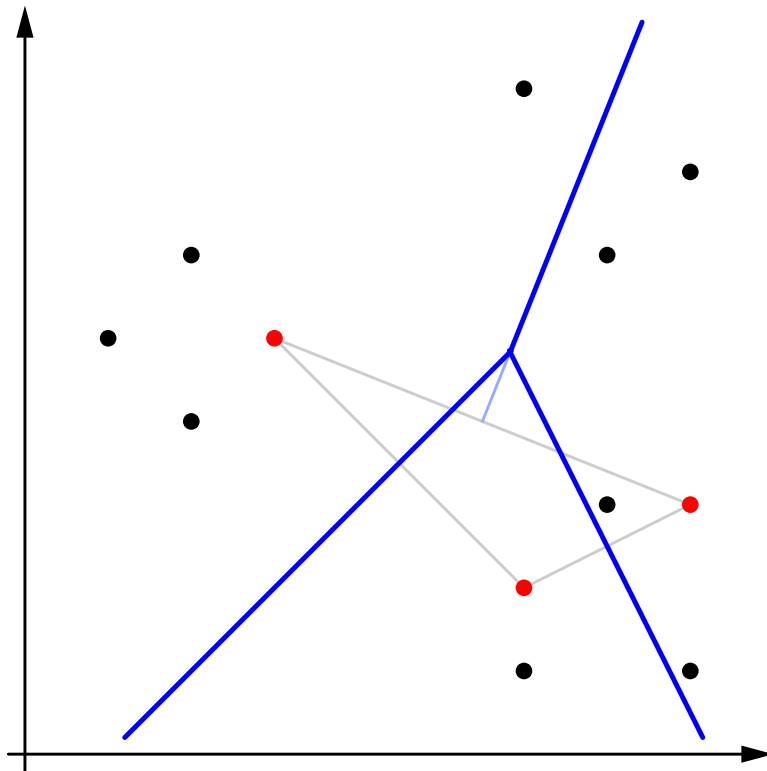
Delaunay Triangulations and Voronoi Diagrams



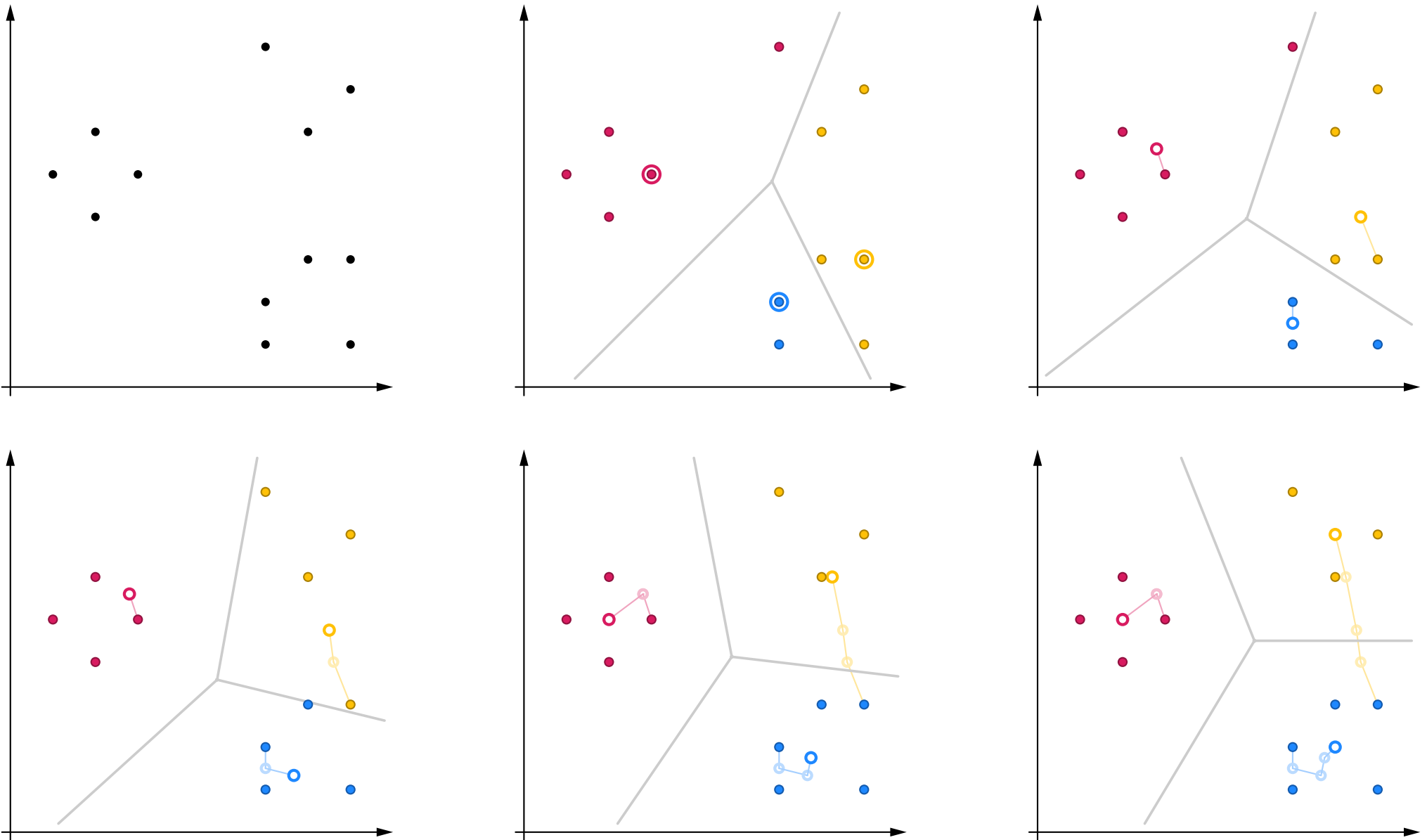
- Dots represent cluster centers (quantization vectors).
- Left: **Delaunay Triangulation**
(The circle through the corners of a triangle does not contain another point.)
- Right: **Voronoi Diagram**
(Midperpendiculars of the Delaunay triangulation: boundaries of the regions of points that are closest 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)



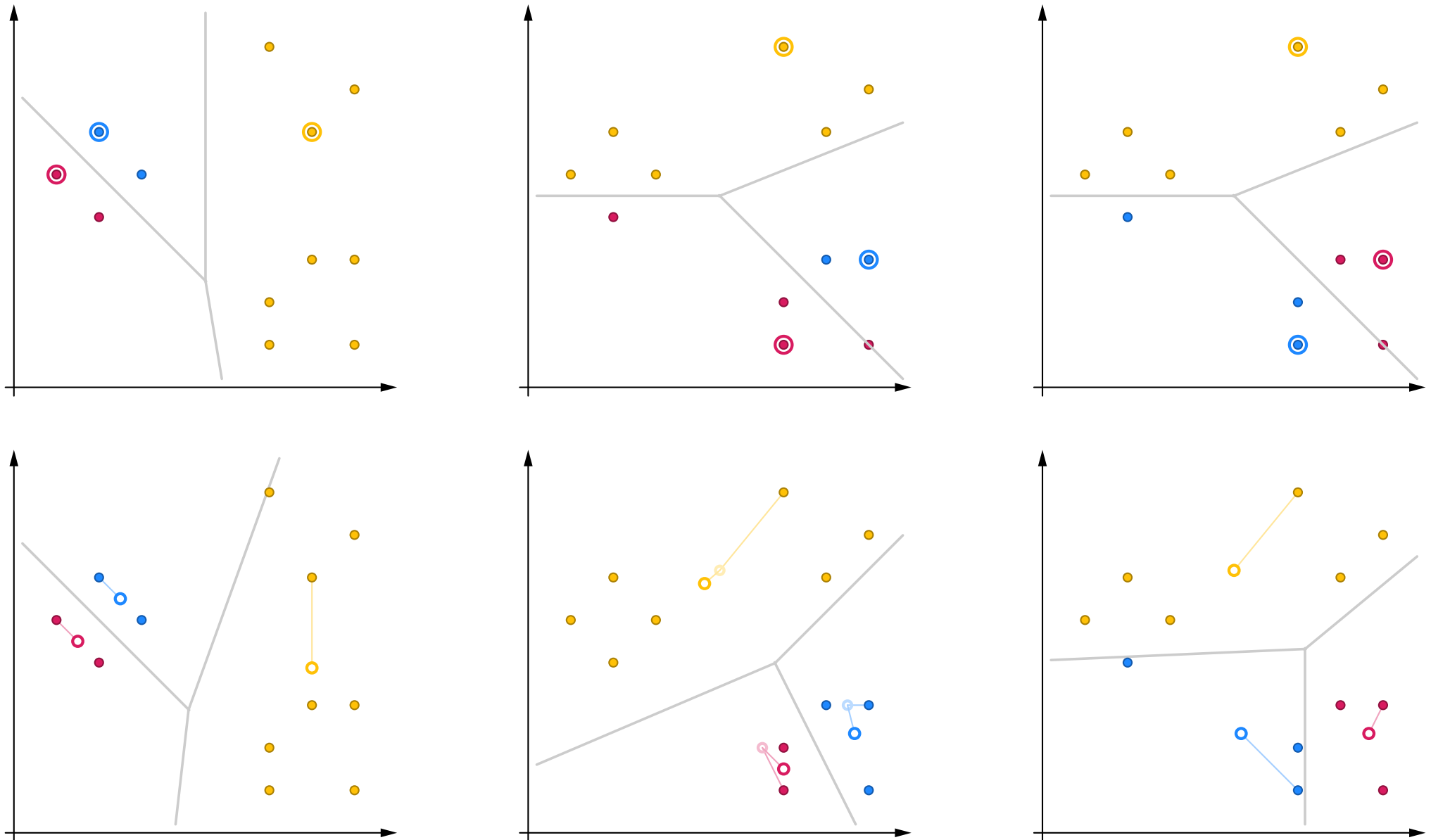
c -Means Clustering: Example



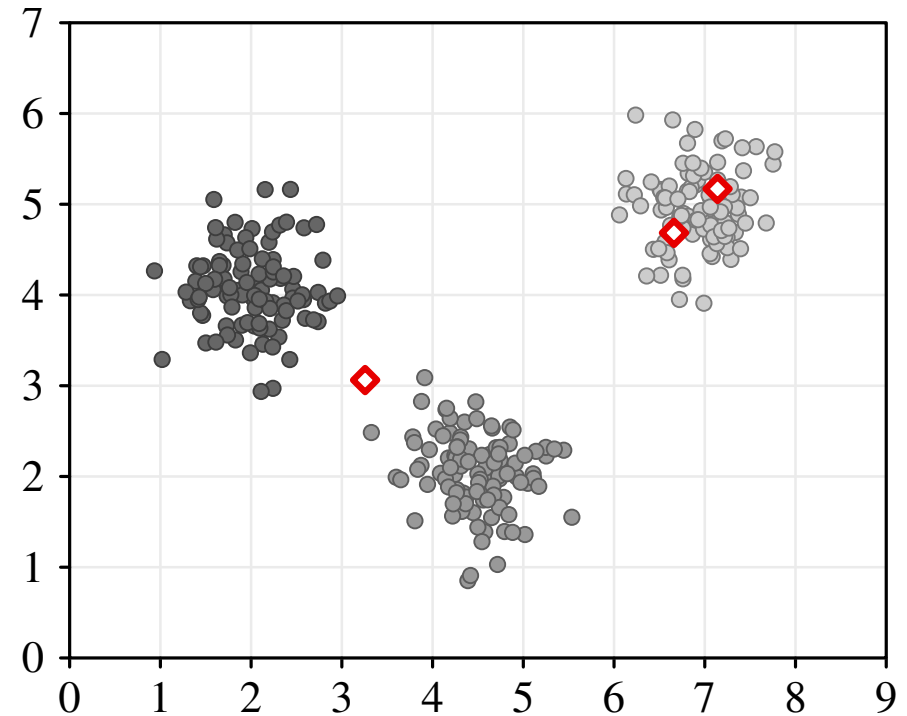
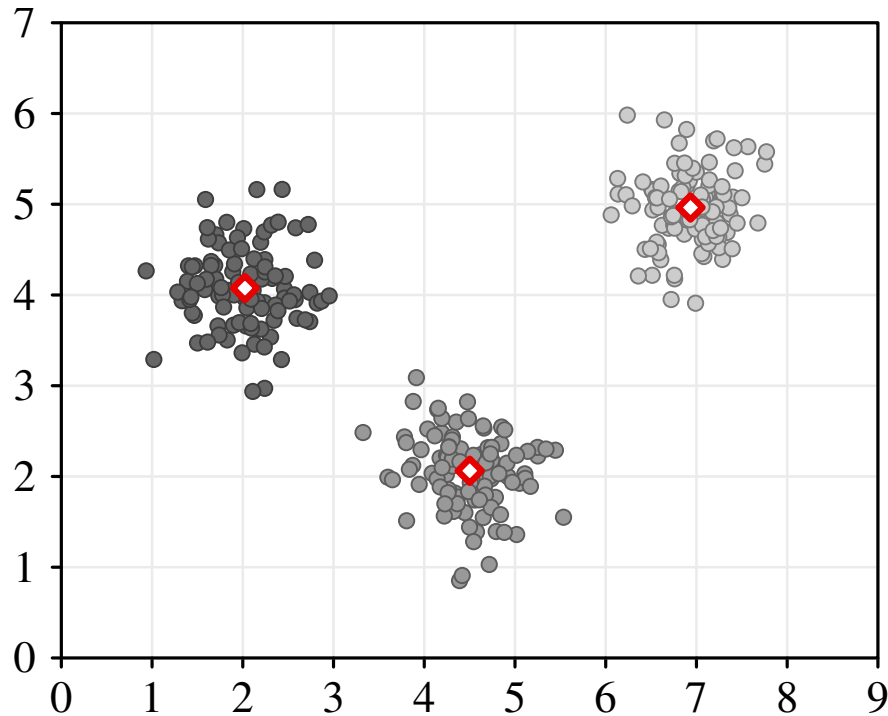
c -Means Clustering: Local Minima

- 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 (\Rightarrow Adv. DM 2).
- 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 (\Rightarrow Adv. DM 2), $u_{ij} \in [0, 1]$ 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
- $\forall j; 1 \leq j \leq n : \quad \sum_{i=1}^c u_{ij} = 1.$

\Rightarrow hard/crisp data point assignment:

each data point is assigned to one cluster and one cluster only.

(“Fuzzy” and probabilistic data point assignments \Rightarrow Adv. DM 2.)

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 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 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 step k , $0 < k \leq c$, 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 $d_{\max}(k, t) = \max_{j=1}^t d_j(k)$.

If $d_{\max}(k, t) > d_{t+1}(s)$ 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 $d_{\max}(k, t)$.
- As long as $s_{t+1} < k$ and $d_{t+1}(s_{t+1}) > d_{\max}(k, t)$ (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 $d_{\max}(k, t + 1)$ accordingly.

Cluster Center Initialization

- **Maximin Initialization:** Python code (using NumPy)

```
ctrs[0] = data[npr.randint(n)] # choose first cluster center randomly
dsts = np.array([dist(ctrs[0], x) for x in data])
cids = np.zeros(n, dtype=int)   # init. highest used cluster indices
for i in range(1,c):           # select the remaining clusters
    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(ctrs[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
    ctrs[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)

```
ctrs[0] = data[npr.randint(n)] # choose first cluster center randomly
dsts = np.array([sqrdist(ctrs[0], x) for x in data])
                                # compute distances to first center
for i in range(1,c):           # select the remaining clusters
    if i > 1:                   # update the minimum distances
        dsts = np.minimum(dsts, [sqrdist(ctrs[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:               # sample randomly from  $d^2$  distribution
        m = size-1             # and ensure that the index is in range
    dsts[m] = 0.0               # mark the data point as selected
    ctrs[i] = data[m]          # and add it to the set of centers
```

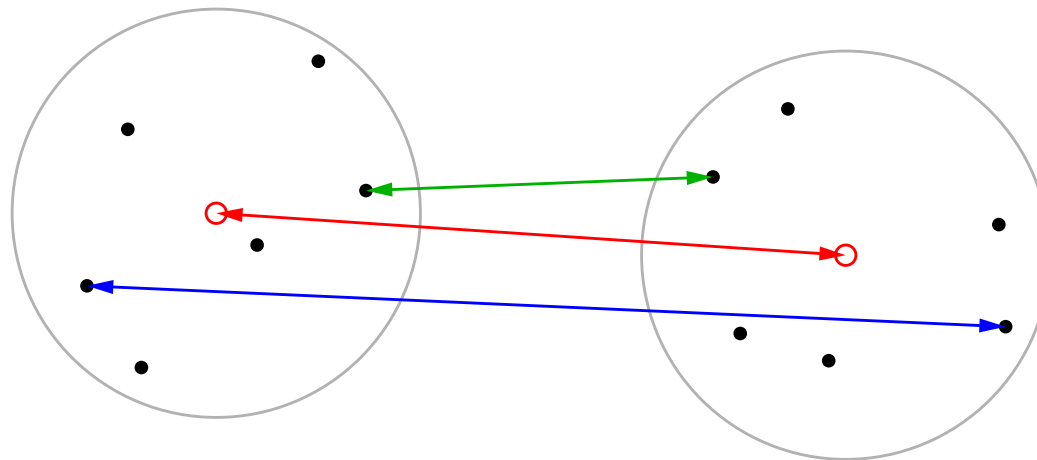
- The computational costs may be reduced
by using a Markov Chain Monte Carlo approach for the sampling.
[Bachem, Lucic, Hassani and Krause 2016]
This approach avoids having to compute all $n \cdot c$ distances.

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 $\deltaέντρον$ (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?

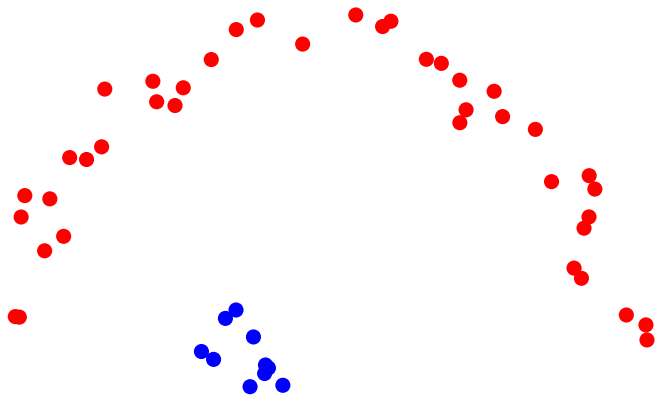
Measuring the Distance between Clusters

- **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.

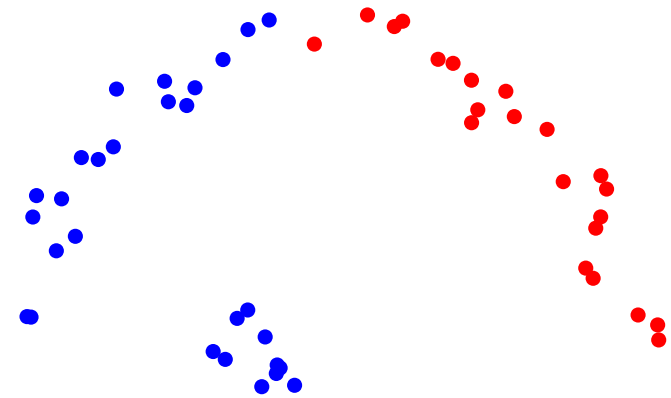


Measuring the Distance between Clusters

- 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

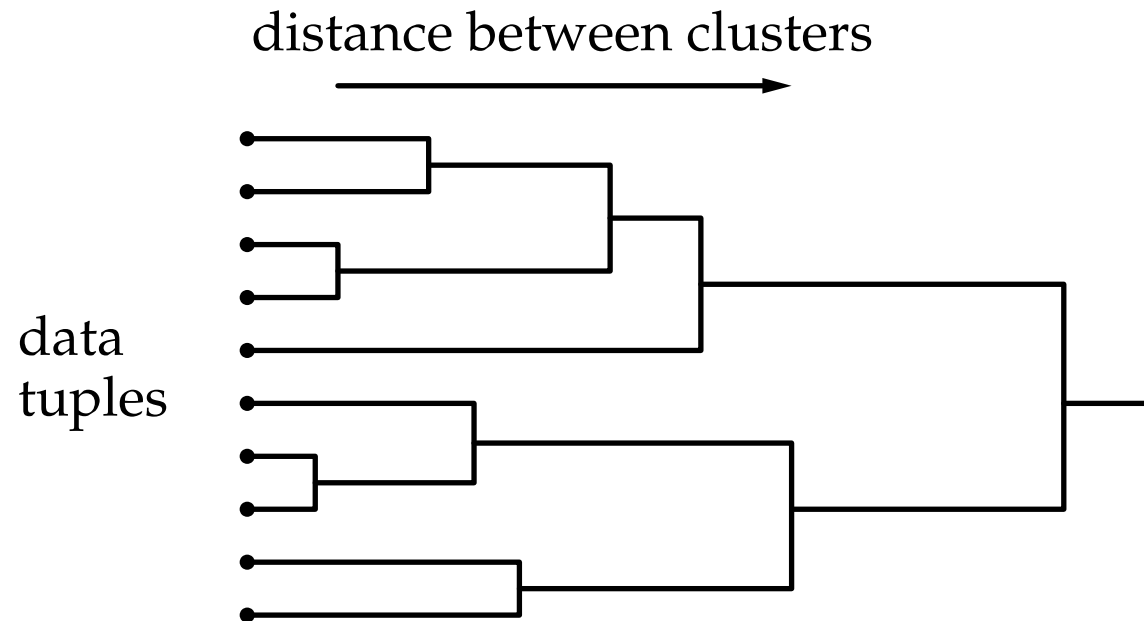


Complete Linkage

(These are the actual results that are computed for this data set; see also below.)

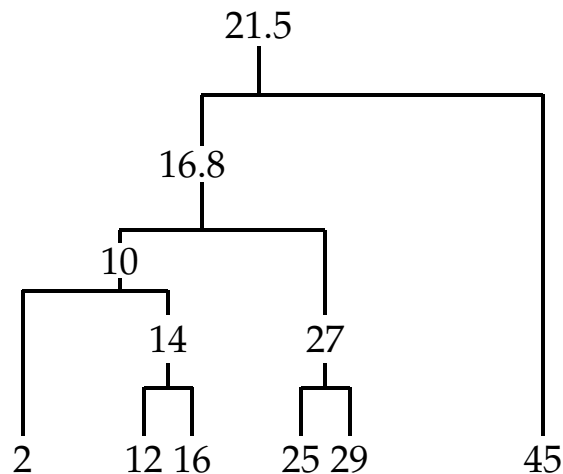
Dendrograms

- 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.

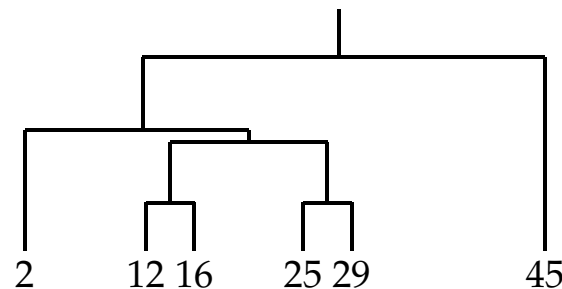


Dendrograms

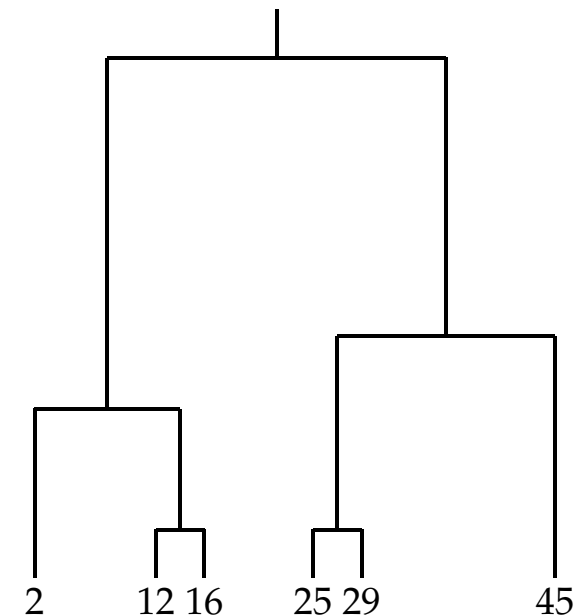
- Example: Clustering of the 1-dimensional data set $\{2, 12, 16, 25, 29, 45\}$.
- All three approaches to measure the distance between clusters lead to different dendrograms.



Centroid



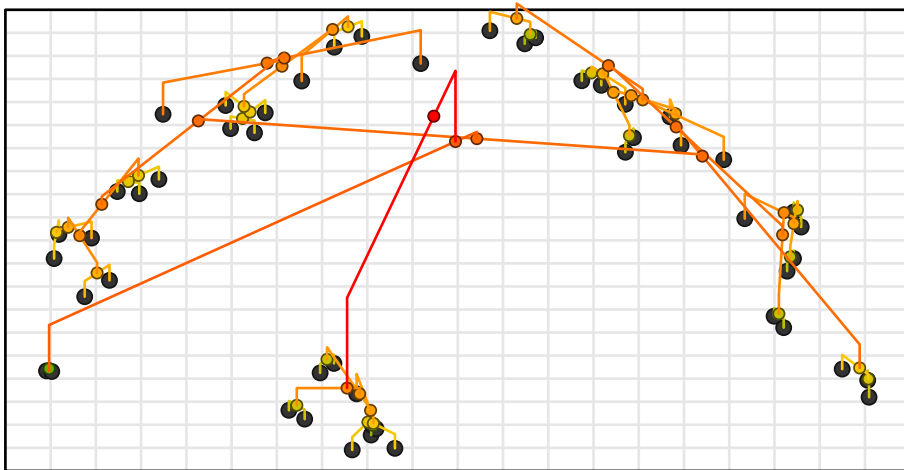
Single Linkage



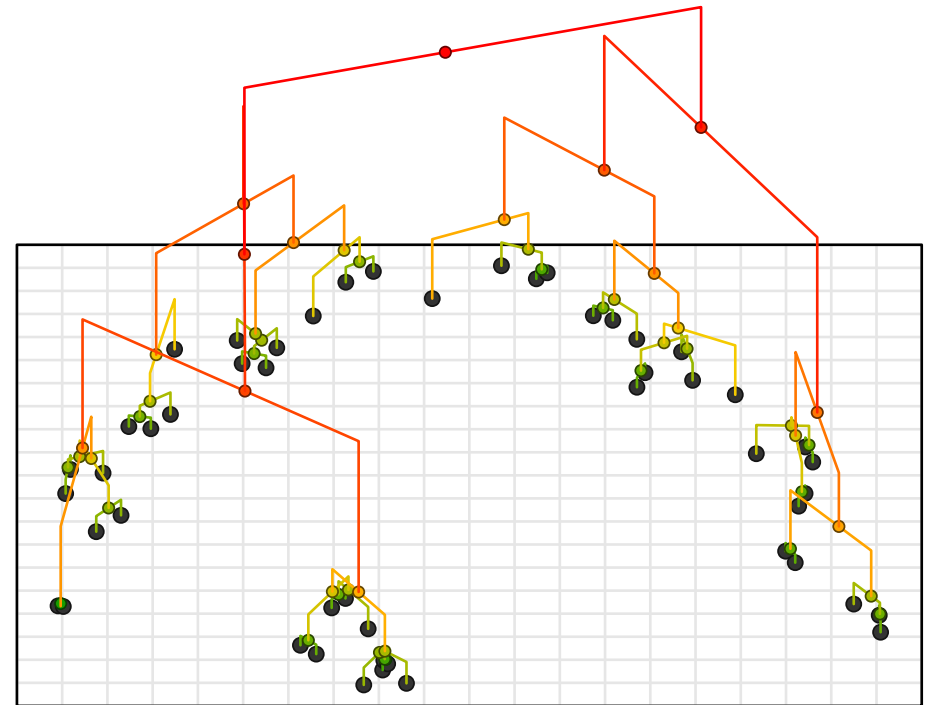
Complete Linkage

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 dendrograms 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 <i>not</i> 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):

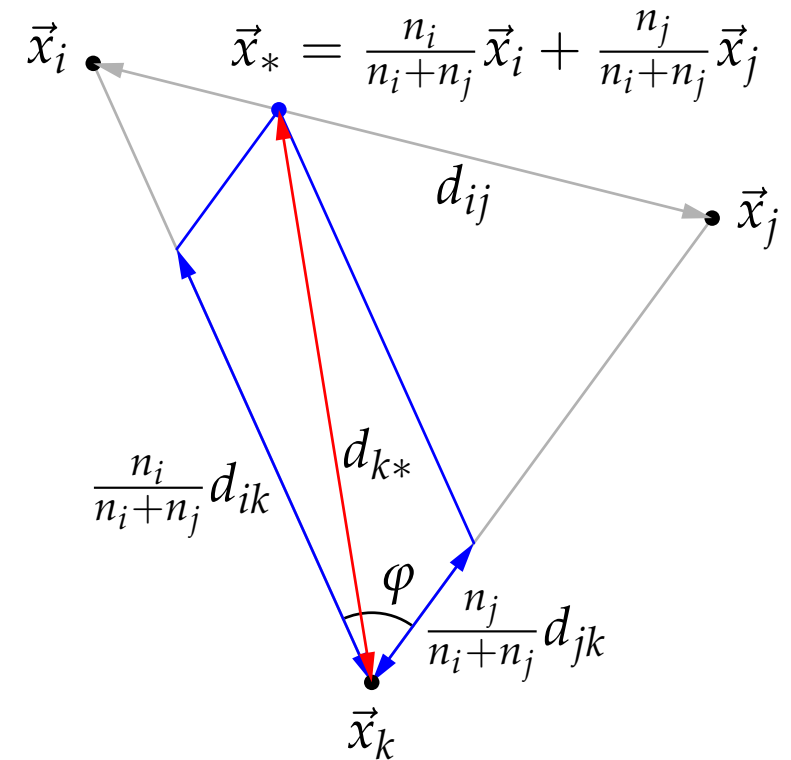
$$(a) \quad d_{ij}^2 = d_{ik}^2 + d_{jk}^2 - 2d_{ik}d_{jk} \cos \varphi$$

$$\Leftrightarrow \quad 2d_{ik}d_{jk} \cos \varphi = d_{ik}^2 + d_{jk}^2 - d_{ij}^2$$

$$(b) \quad d_{k*}^2 = \left(\frac{n_i}{n_i + n_j} d_{ik} \right)^2 + \left(\frac{n_j}{n_i + n_j} d_{jk} \right)^2$$

$$+ \underbrace{2 \frac{n_i}{n_i + n_j} d_{ik} \frac{n_j}{n_i + n_j} d_{jk} \cos \varphi}_{= \frac{n_i n_j}{(n_i + n_j)^2} (d_{ik}^2 + d_{jk}^2 - d_{ij}^2); \text{ see (a)}}$$

$$= \underbrace{\frac{n_i}{n_i + n_j}}_{\alpha_i} d_{ik}^2 + \underbrace{\frac{n_j}{n_i + n_j}}_{\alpha_j} d_{jk}^2 - \underbrace{\frac{n_i n_j}{(n_i + n_j)^2}}_{\beta} d_{ij}^2 + \underbrace{0}_{\gamma} |d_{ik}^2 - d_{jk}^2|$$



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 assignments and cluster parameters.
- Crisp/hard assignment of a datum to a cluster (data partitioning).
- Local minima of the objective function can pose problems.
- A good initialization of the cluster centers is important.
- (Fuzzy/probabilistic approaches are more robust.) \Rightarrow Data Mining 2

- **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.