

# Advanced Data Mining 1

**Christian Borgelt**

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

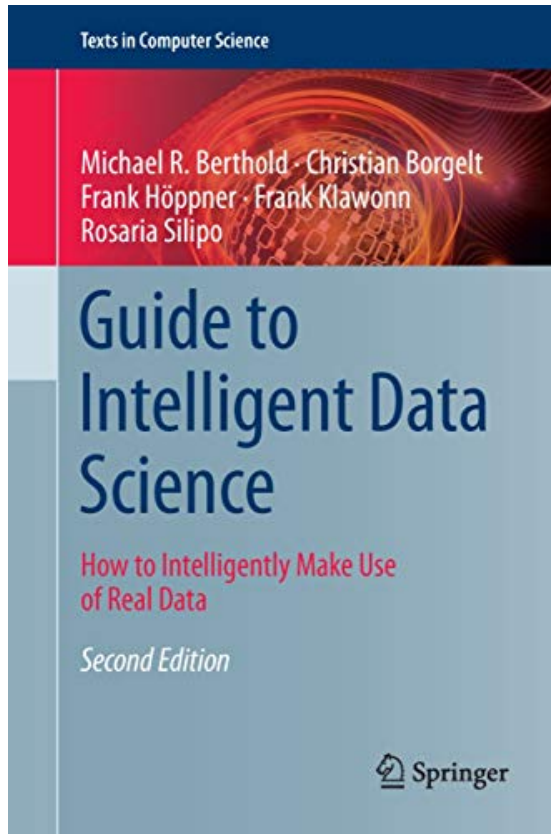
`christian.borgelt@plus.ac.at`

`christian@borgelt.net`

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

`https://meet.google.com/ear-vxzn-vyg`

# 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)

# Reminder: (Elementary) Data Mining

# Reminder: 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

# Reminder: The Knowledge Discovery Process (KDD Process)

## Preliminary Steps

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

## Main Steps

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

# Reminder: 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?*

# Reminder: Data Analysis / Data Mining Methods 1

- **Classical Statistics**  $\Rightarrow$  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**  $\Rightarrow$  Lecture "Elementary Data Mining"  
(lazy learning, similarity measures, neighbor weighting, data structures for fast search)  
tasks: classification, prediction
- **Bayes Classifiers**  $\Rightarrow$  Lecture "Elementary Data Mining"  
(probabilistic classification, naive and Gaussian Bayes classifiers, Bayesian network classifiers)  
tasks: classification, prediction
- **Decision and Regression Trees**  $\Rightarrow$  Lecture "Elementary Data Mining"  
(top down induction, attribute selection measures, pruning, regression trees)  
tasks: classification, prediction
- **Rule Learning** blue: in this lecture  
(extraction from decision trees, A<sup>9</sup>, CN2, version spaces, inductive logic programming)  
tasks: classification, prediction

# Reminder: Data Analysis / Data Mining Methods 2

- **Support Vector Machines**

blue: in this lecture

(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 “Elementary Data Mining”

dijon: ⇒ 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



# Reminder: 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)

# Reminder: 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.

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

# Reminder: 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.

# Reminder: Causes of Errors

One may distinguish between **four types of errors**:

- **pure / intrinsic / experimental / Bayes error**

Inherent in the data, independent of the choice of the model;  
no model can yield an error less than this error (on new data).

- **sample / variance error or scatter**

Caused by the fact that the given data is only a sample and  
thus an imperfect representation of the underlying distribution.

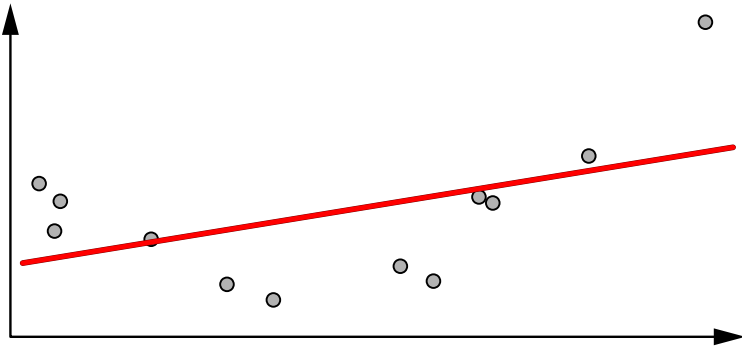
- **lack of fit / model error / bias error**

Caused by an improper choice of the model class.  
(see also: under- and overfitting)

- **algorithmic error**

Caused by the method used to fit the model or its parameters;  
heuristic optimization strategy may not find global optimum.

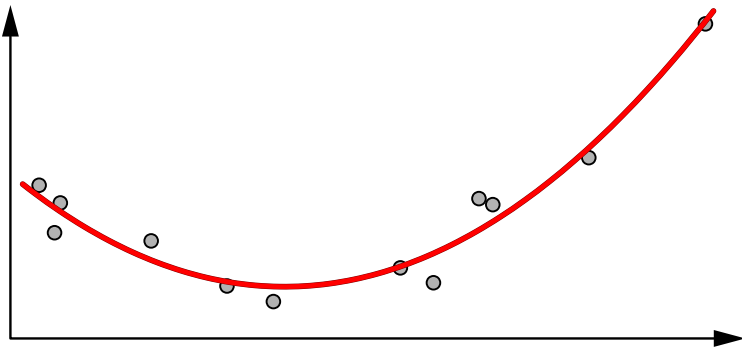
# Reminder: Under- and Overfitting



## Underfitting

[here: linear]

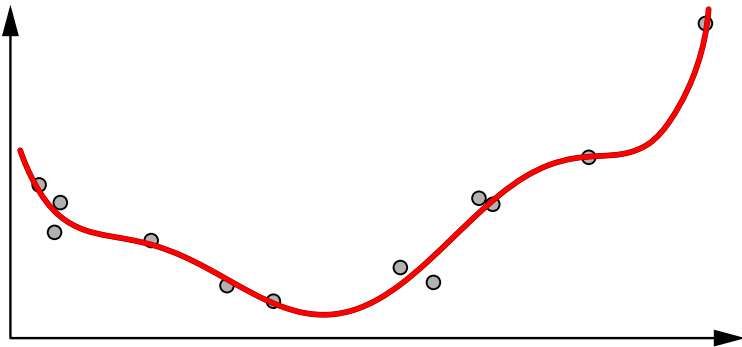
- Caused by model error / lack of fit.
- Model has not enough capacity to fit the regularities in the data.



## (Proper) Fitting

[here: quadratic]

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

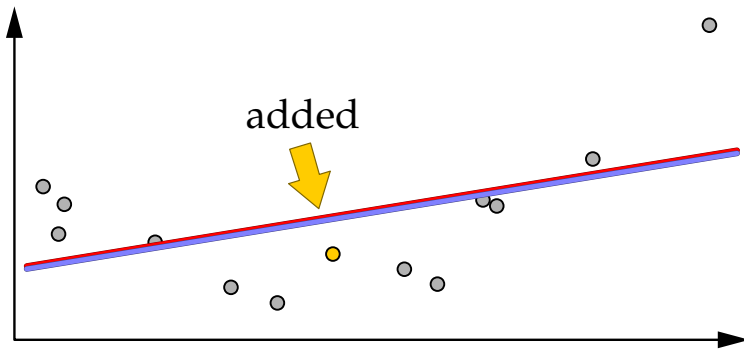


## Overfitting

[here: degree 6]

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

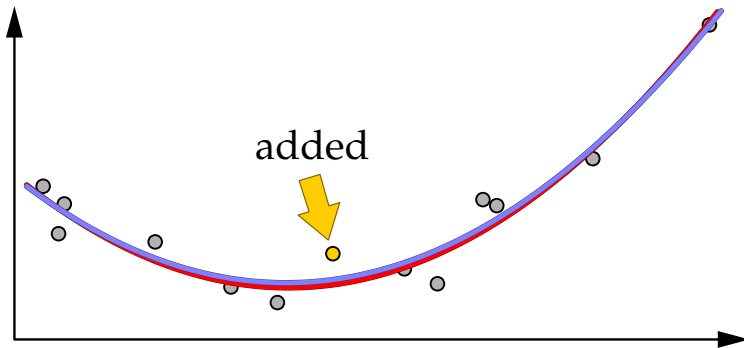
# Reminder: Under- and Overfitting



## Underfitting

[here: linear]

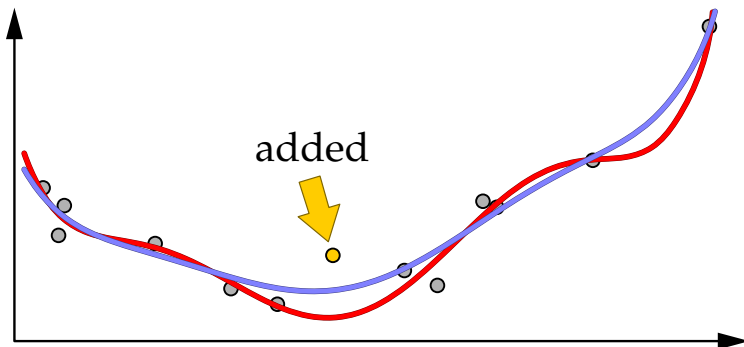
- Caused by model error / lack of fit.
- Model has not enough capacity to fit the regularities in the data.



## (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.

# Rule Learning



# The Reason of Rules

- **Rules** are logical **implications**, often expressed in a natural language form:
  - **If** a customer buys **bread** and **wine**, (antecedent of the rule)  
**then** she/he will probably also buy **cheese**. (consequent of the rule)
- Many domain experts are in favor of rules as a data mining result, because they view them as **more comprehensible** than other model types.
- Rules certainly look comprehensible, but are also **easily misinterpreted**.
- Suppose:
  - 200 customers bought something at a supermarket.
  - 140 customers bought cheese.
  - 100 customers bought both bread and wine.
  - 70 customers bought bread, wine and cheese.

Then:  $\hat{P}(\text{cheese}) = \frac{140}{200} = 70\%$  and  $\hat{P}(\text{cheese} \mid \text{bread, wine}) = \frac{70}{100} = 70\%$ .

$\Rightarrow$  The presence of the items in the antecedent does not change the probability.

(Side remark: The title of this slide is a play on the title of the book

Geoffrey Brennan and James M. Buchanan: "The Reason of Rules: Constitutional Political Economy", 1985)

# The Reason of Rules

- However, a rule implicitly conveys the impression that the antecedent causes (or at least significantly changes the probability of) the consequent.
- A simple measure commonly used to assess this is the so-called **lift value**.

Consider a rule  $R : X \rightarrow Y$ . Then 
$$\text{lift}(R) = \frac{\hat{P}(Y | X)}{\hat{P}(Y | \emptyset)} = \frac{\hat{P}(X \wedge Y)}{\hat{P}(X)\hat{P}(Y)}.$$

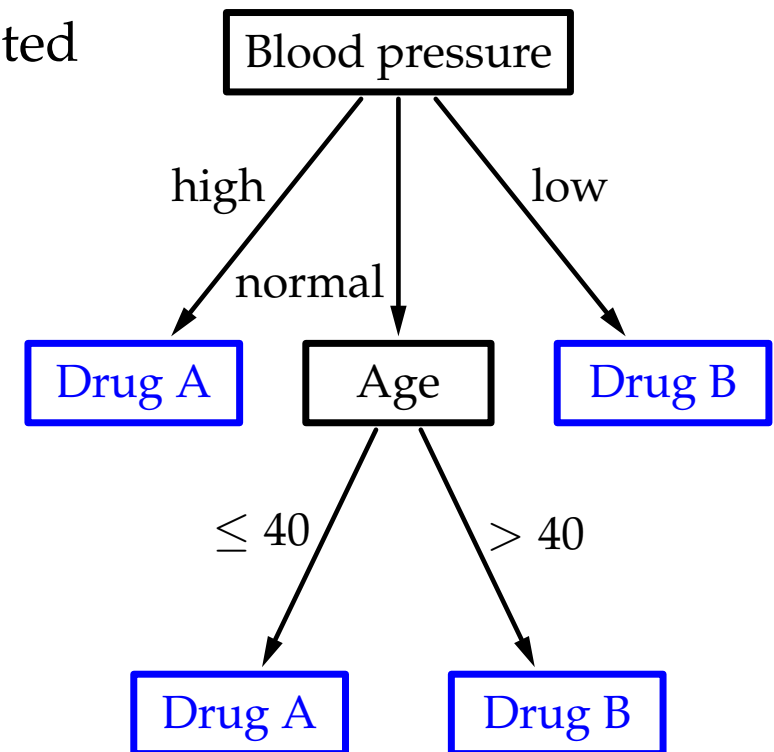
Given a rule  $R$ , we assume at least  $\text{lift}(R) > 1$ , although this might be false.

- The lift value already helps to filter out many uninformative rules.
- Furthermore, contrasting several rules is usually helpful to gain insights.  
(Example rules induced on the 1994 Census/Adult data set; numbers after are lift values.)
  - **If** education = Bachelors, **then** salary > \$50K (1.73)
  - **If** education = Masters, **then** salary > \$50K (2.29)
  - **If** education = Masters, **then** hours = overtime (1.58)
  - **If** education = Bachelors **and** hours = overtime, **then** salary > \$50K (2.29)

# Rule Extraction from Decision Trees

# Reminder: Decision Trees

- 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.
- A **decision tree** is a classifier with tree structure, in which the classification process starts at the root node and proceeds down the branches until a leaf is reached.
  - In inner nodes, descriptive attributes are tested and the case to classify is directed down the branch corresponding to the test outcome.
  - When finally a leaf node is reached, the leaf node class is assigned to the case.
- **Decision tree induction** is characterized by:
  - Top-down approach (from root to leaves).
  - Greedy selection of test attributes / tests.
  - Divide-and-conquer / recursive descent.



# Rule Extraction from Decision Trees

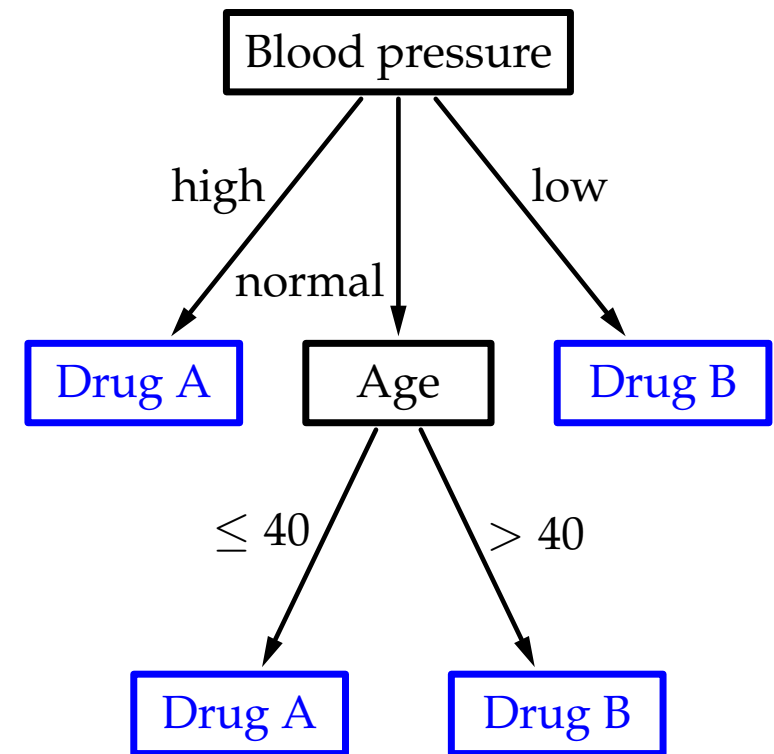
- Each of the paths from the root to the various leaf nodes can be seen as a rule.
- Extracting these rules allows us to represent the classifier in a different way.
- This representation allows for different ways of simplifying the classifier. (meaning: other than standard decision tree pruning.)
- The decision tree on the right is equivalent to the following four rules:

If Blood pressure = high,  
    **then** prescribe Drug A.

If Blood pressure = low,  
    **then** prescribe Drug B.

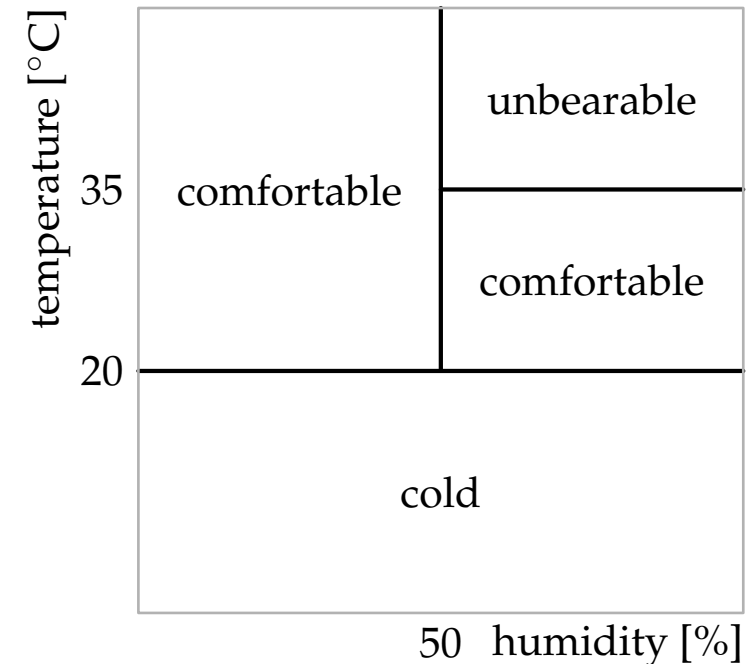
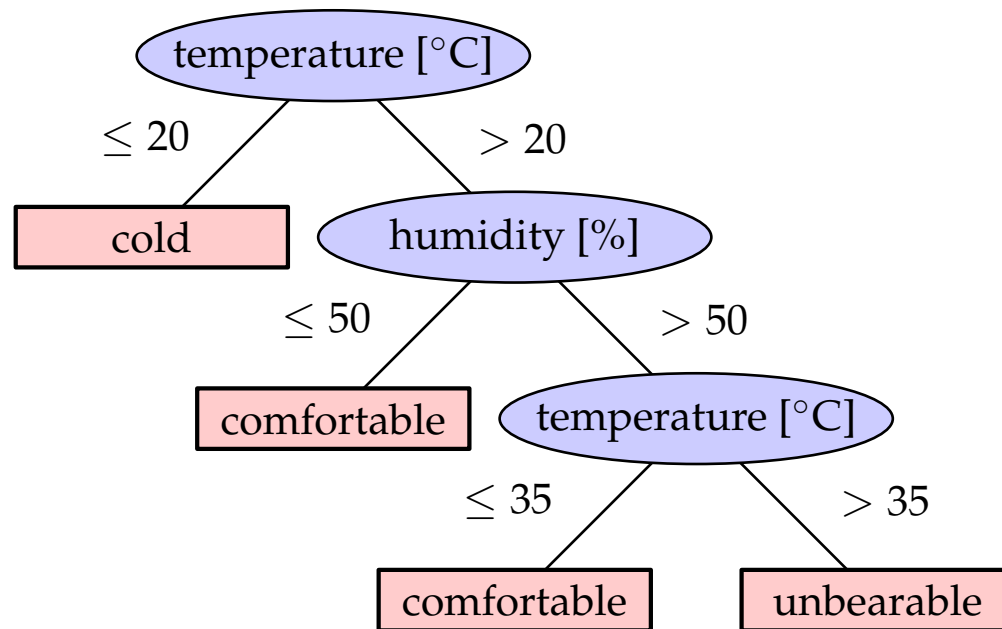
If Blood pressure = normal **and** Age  $\leq 40$ ,  
    **then** prescribe Drug A.

If Blood pressure = normal **and** Age  $> 40$ ,  
    **then** prescribe Drug B.



# Reminder: Data Space Partitioning

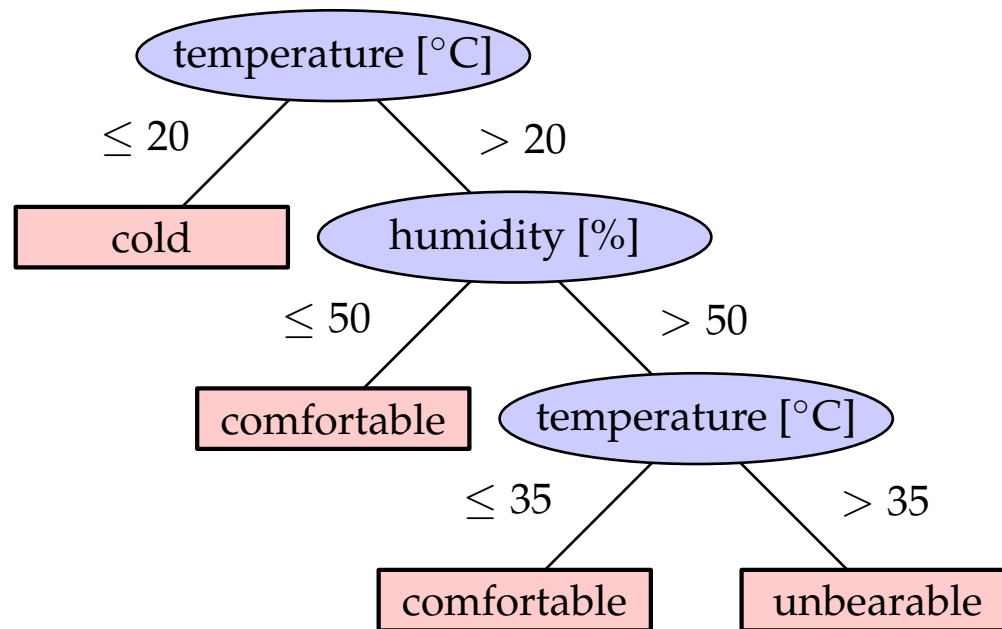
A decision tree partitions the data space.



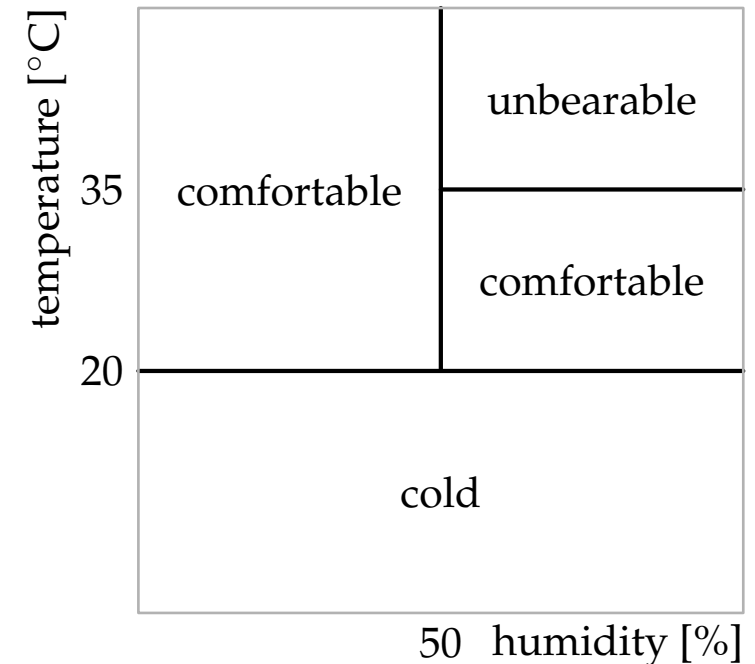
- Each inner node (test node) cuts a dimension at the given threshold.
- If there are parent / ancestor nodes, only a subspace is cut.  
(The "humidity" test only cuts above "temperature = 20°",  
the bottom "temperature" test only to the right of "humidity = 50%").

# Rule Extraction from Decision Trees

A decision tree partitions the data space.



Each rule refers to one partition.



- If temperature  $\leq 20$ , **then** cold.
- If temperature  $> 20$  **and** humidity  $\leq 50$ , **then** comfortable.
- If temperature  $> 20$  **and** humidity  $> 50$  **and** temp.  $\leq 35$ , **then** comfortable.
- If temperature  $> 20$  **and** humidity  $> 50$  **and** temp.  $> 35$ , **then** unbearable.

# Simplifying Extracted Rules

- We just extracted the following four rules from a decision tree:
  - If temperature  $\leq 20$ , **then** cold.
  - If temperature  $> 20$  **and** humidity  $\leq 50$ , **then** comfortable.
  - If temperature  $> 20$  **and** humidity  $> 50$  **and** temp.  $\leq 35$ , **then** comfortable.
  - If temperature  $> 20$  **and** humidity  $> 50$  **and** temp.  $> 35$ , **then** unbearable.
- The last two rules can be trivially simplified to:
  - If temperature  $\in (20, 35]$  **and** humidity  $> 50$ , **then** comfortable.
  - If temperature  $> 35$  **and** humidity  $> 50$ , **then** unbearable.

We simply collect and combine conditions referring to the same attribute.

- If we allow for disjunctions, we can combine rules for the same class.
  - If (temperature  $> 20$  **and** humidity  $\leq 50$ ),  
or (temperature  $\in (20, 35]$  **and** humidity  $> 50$ ), **then** comfortable.

(However, this may make rules too complex to be easily comprehensible.)



# Exploiting the Rule Order: Decision Lists

- Rules extracted (as shown) from a decision tree are **mutually exclusive**, that is, for any case / data point to classify exactly one rule is applicable. (A rule is applicable if its antecedent is satisfied.)
- As a consequence, the **order** in which the rules are tested is **irrelevant**.
- By **ordering the rules**, and thus forming a so-called **decision list**, the rules can be simplified further, exploiting that preceding rules were inapplicable.

If Blood pressure = high,  
    **then** prescribe Drug A.

If Blood pressure = low,  
    **then** prescribe Drug B.

If Blood pressure = normal **and** Age  $\leq 40$ ,  
    **then** prescribe Drug A.

If Blood pressure = normal **and** Age  $> 40$ ,  
    **then** prescribe Drug B.

If Blood pressure = high,  
    **then** prescribe Drug A.

If Blood pressure = low,  
    **then** prescribe Drug B.

If Age  $\leq 40$ ,  
    **then** prescribe Drug A.

If true,  
    **then** prescribe Drug B.

# Exploiting the Rule Order: Decision Lists

- In a **decision list** the rules have to be inspected in a specific order. The first rule in the list that is applicable yields the classification. All rules further down in the order are not inspected anymore.
- Note that this simplifies the rules, but also introduces dependencies between the rules (their order), which can make decision lists less easy to understand.
- As a consequence, pure decision lists are not so popular for rule sets.

If temperature  $\leq 20$ ,  
    **then** cold.

If temperature  $> 20$  **and** humidity  $\leq 50$ ,  
    **then** comfortable.

If temperature  $\in (20, 35]$  **and** humidity  $> 50$ ,  
    **then** comfortable.

If temperature  $> 35$  **and** humidity  $> 50$ ,  
    **then** unbearable.

If temperature  $\leq 20$ ,  
    **then** cold.

If humidity  $\leq 50$ ,  
    **then** comfortable.

If temperature  $\leq 35$ ,  
    **then** comfortable.

If true,  
    **then** unbearable.

# Reminder: Pruning Decision Trees

## Pruning decision trees serves the purpose

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

## Basic ideas:

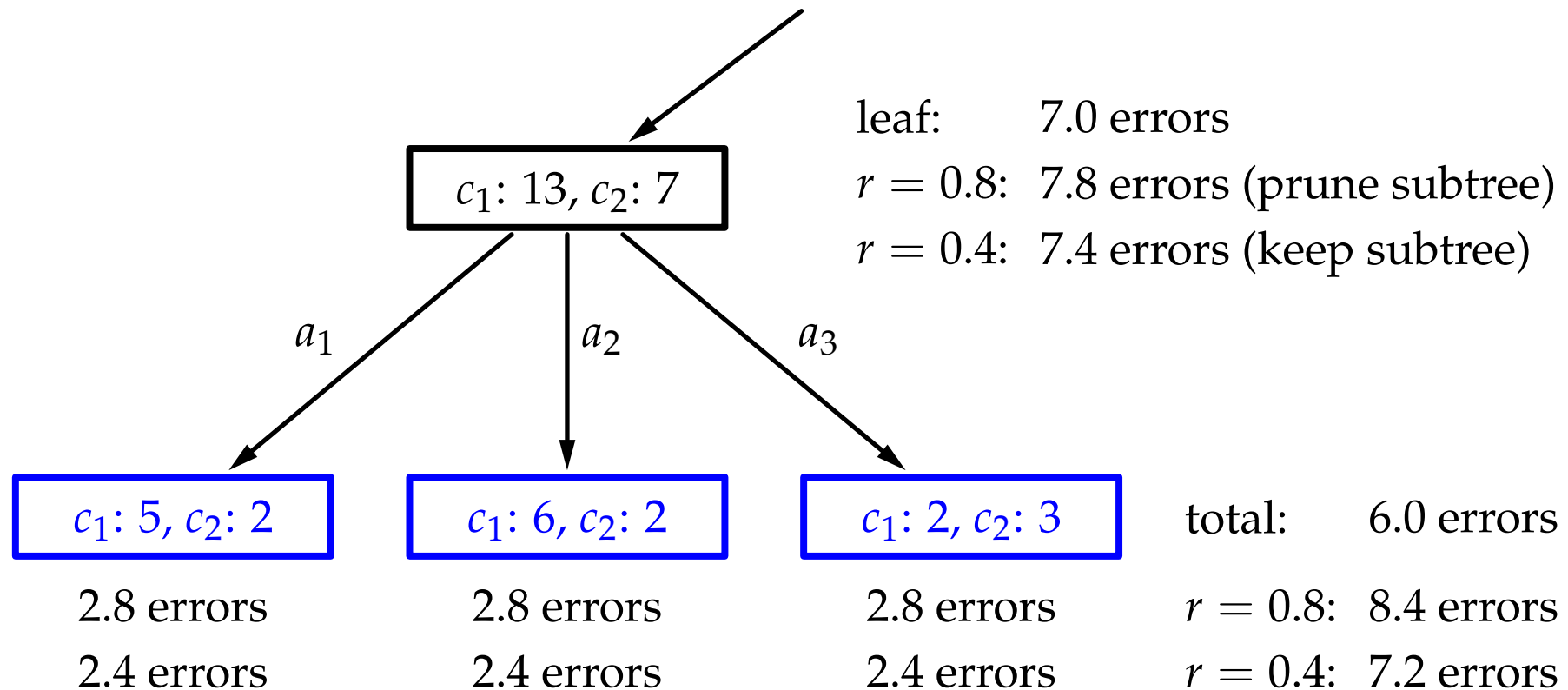
- Replace “bad” branches (subtrees) by leaves.
- Replace a subtree by its largest branch if it is better.

## Common approaches:

- Limiting the number of leaf cases (e.g. at least  $k$  sample cases per leaf)
- Reduced error pruning (use errors from new sample cases)
- Pessimistic pruning (constant error increment per leaf)
- Confidence level pruning (upper bound of confidence interval)
- Minimum description length pruning (model yielding best transmission)

# Reminder: Pruning Decision Trees

Pessimistic Pruning with  $r = 0.8$  and  $r = 0.4$ :

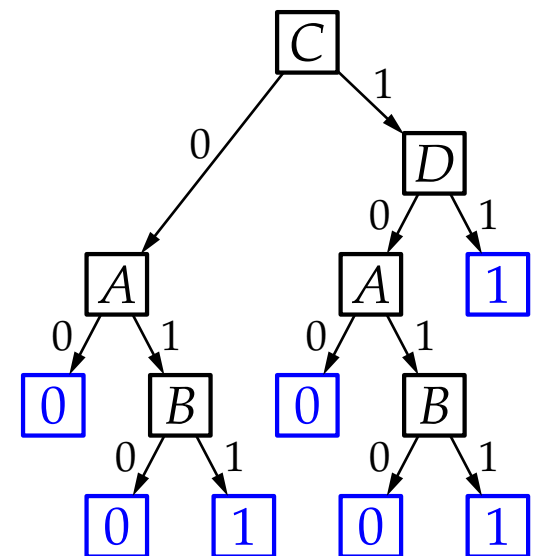
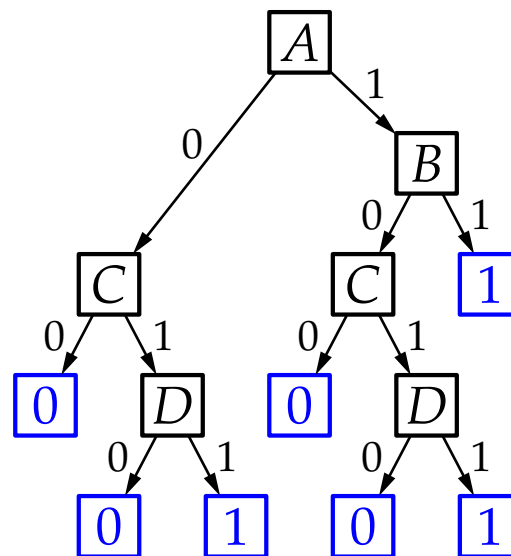


- **Confidence level pruning** follows essentially the same scheme, but estimates the expected number of errors by increasing the observed number to the upper bound of a confidence interval.

# Pruning Extracted Rules

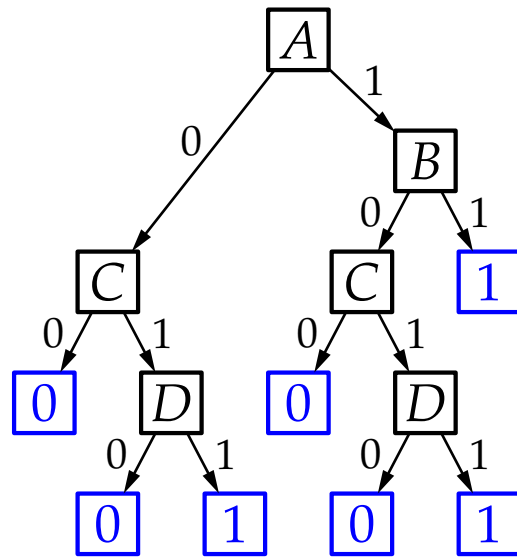
A	B	C	D	Class
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

- For binary variables, consider the concept  $\text{Class} = (A = 1 \wedge B = 1) \vee (C = 1 \wedge D = 1)$  that is shown in the table on the left.
- This concept can be described by decision trees as



- Note that these trees contain the same subtree twice, and that each of these subtrees could stand alone.

# Pruning Extracted Rules



- This tree's decisions could be captured by the rules:  
If  $A = 1$  and  $B = 1$ , then Class = 1.  
If  $C = 1$  and  $D = 1$ , then Class = 1.  
If true, then Class = 0.  
(Note that the first two rules could be exchanged.)
- These rules are a much simpler description, but these rules are **not mutually exclusive**.

- How can we prune rules extracted from a decision tree in this way?
- Consider these rules extracted from the decision tree above:  
If  $A = 0$  and  $C = 1$  and  $D = 1$ , then Class = 1.  
If  $A = 1$  and  $B = 0$  and  $C = 1$  and  $D = 1$ , then Class = 1.
- The condition  $A = 0$  and the conditions  $(A = 1 \text{ and } B = 0)$  are superfluous.

# Pruning Extracted Rules

- How can we identify superfluous conditions in extracted rules?
- Consider a rule  $R_{+X} : A \wedge X \rightarrow C$  with a potentially superfluous condition  $X$ .
- From the training data, we can set up a contingency table for the sample cases satisfying the condition(s)  $A$  (but not necessarily the condition  $X$ ):

	C false	C true	$\Sigma$
X false	$n_{00}$	$n_{01}$	$n_{0.}$
X true	$n_{10}$	$n_{11}$	$n_{1.}$
$\Sigma$	$n_{.0}$	$n_{.1}$	$n_{..}$

We can now compare the quality of the reduced rule  $R_{-X} : A \rightarrow C$  and the original rule  $R_{+X} : A \wedge X \rightarrow C$  and choose the better rule.

- The original rule has the error rate  $E_{+X} = \frac{n_{10}}{n_{1.}} = \frac{n_{10}}{n_{10} + n_{11}}.$

The reduced rule has the error rate  $E_{-X} = \frac{n_{.0}}{n_{..}} = \frac{n_{00} + n_{10}}{n_{00} + n_{10} + n_{01} + n_{11}}.$

- However, on the training data the error rates are usually underestimated. (This is in complete analogy to the pruning of decision trees.)

# Pruning Extracted Rules

- **Reduced error pruning** as for decision trees may very well be applicable.
- **Pessimistic pruning** works because of subtree leaves versus a single leaf (multiple error increments versus one), which is not applicable here.
- However, **confidence level pruning** can be applied well, [Quinlan 1993] even though it is based on the same scheme as pessimistic pruning.
- The upper boundary of an error rate confidence interval for  $e$  errors in  $n$  cases and a confidence level of  $1 - \alpha$  can be approximated as [Wilson 1927]

$$U_{\alpha}(e, n) = \begin{cases} 1 - \sqrt[n]{\frac{\alpha}{2}} & \text{if } e = 0 \quad (\text{exact}), \\ \frac{1}{n+z^2} \left( e + \frac{z^2}{2} + \sqrt{z^2 \left( e \left( 1 - \frac{e}{n} \right) + \frac{z^2}{4} \right)} \right) & \text{otherwise (approximate),} \end{cases}$$

where  $z = Q_{\Phi}(\frac{\alpha}{2})$  and  $Q_{\Phi}$  is the quantile function of the std. normal distribution.

- Compute  $U_{\alpha,+X} = U_{\alpha}(n_{10}, n_{1.})$  for rule  $R_{+X}$  and  $U_{\alpha,-X} = U_{\alpha}(n_{.0}, n_{..})$  for rule  $R_{-X}$  and discard the condition  $X$  (that is, prefer rule  $R_{-X}$ ) if  $U_{\alpha,-X} \leq U_{\alpha,+X}$ .



# Pruning Extracted Rules

- In order to handle **multiple conditions**, all of which may be discardable, testing all subsets of conditions is possible, but may sometimes be too costly.
- In such a case, a **greedy strategy** can be applied to a rule  $R : X_1 \wedge \dots \wedge X_k \rightarrow C$ :
  - Compute  $U_{\alpha, -X_i}$  for all  $i = 1, \dots, k$  (error rate if condition  $X_i$  is discarded).
  - Discard condition  $X_j$  with  $U_{\alpha, -X_j} = \min_{i=1}^k U_{\alpha, -X_i}$ , provided  $U_{\alpha, -X_j} \leq U_{\alpha, +X_j}$ .
  - Repeat with reduced condition list until no condition can be discarded without increasing the error above the current (reduced)  $U_{\alpha}$ .
- Apply this rule reduction approach to all extracted rules.
- Note that the resulting rules are **not (necessarily) mutually exclusive**, because discarding conditions from rules can lead to more than one rule being applicable to a sample case.
- In order to handle such cases, we need a **conflict resolution strategy**.
- A very simple such strategy is to construct a **decision list**.

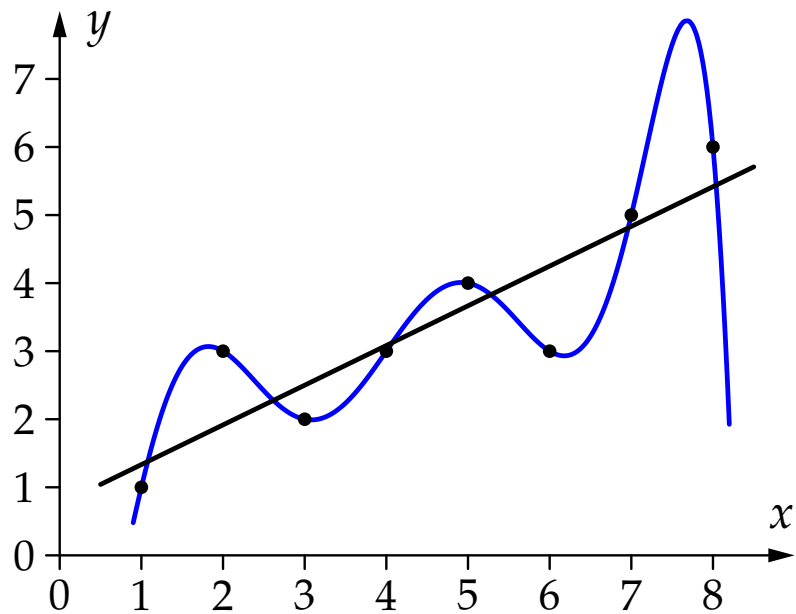
# Building a Rule-based Classifier

- However, as already mentioned, **a pure decision list is problematic**, because of the severe dependencies introduced by the ordering.
- A better approach consists in **grouping all rules that predict the same class**, and then to order only these class rule subsets. [Quinlan 1993]
- The idea is that the order of rules predicting the same class does not matter.
- All rules predicting one class precede all rules predicting the next class etc. There is also a default rule (“If true, then ...”) or default class at the end.
- Building a rule-based classifier with this approach has four steps:
  - Simplify individual rules by discarding conditions. (already considered)
  - Group rules predicting the same class and simplify each group by discarding “bad” rules. (“model selection” in a way)
  - Order the class rule subsets in order to minimize errors.
  - Choose a default class (for cases for which no rule is applicable).

# Excursion: 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.*  
(Consider especially predictions based on the black versus the blue model.)

## Excursion: Information Criteria

- There is a **tradeoff between model complexity and fit to the data**.

Question: How much better must a more complex model fit the data in order to justify the higher complexity?

- One approach to quantify the tradeoff: **Information Criteria**

Let  $D$  be the data,  $M$  a model and  $\Theta$  the set of free parameters of  $M$ . Then:

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

where  $D$  are the sample data and  $\kappa$  is a parameter.

Special cases:

- **Akaike Information Criterion (AIC):**  $\kappa = 2$
- **Bayesian Information Criterion (BIC):**  $\kappa = \ln n$ ,  
where  $n$  is the sample size
- The lower the value of these measures, the better the model.  
They are commonly used in statistics, e.g. for regression models.

## Excursion: Minimum Description Length

- Idea: Consider the transmission of the data from a sender to a receiver.

Since the transmission of information is costly,  
the length of the message to be transmitted should be minimized.

- A good model of the data can be used to transmit the data with fewer bits.

However, the receiver does not know the model the sender used  
and thus cannot decode the message.

Therefore: if the sender uses a model, he/she has to transmit the model as well.

- **description length** = **length of model description**  
+ **length of data description**

(A more complex model increases the length of the model description,  
but reduces the length of the data description.)

- The model that leads to the smallest total description length is the best.

This model selection idea is used in both statistics and machine learning.

# Minimum Description Length: Simple Example

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

$$l_1 = \log_2 k^n = n \log_2 k, \quad (\text{or: pure data description})$$

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

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

$$l_2 = \underbrace{\log_2 \frac{(n+k-1)!}{n!(k-1)!}}_{\text{model description}} + \underbrace{\log_2 \frac{n!}{n_1! \dots n_k!}}_{\text{data description}}$$

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

# Minimum Description Length: Simple Example

## Model Description Codebook

Lists one distribution of  $n$  cases onto  $k$  values per page, that is,

- how many cases have value 1,
- how many cases have value 2,
- $\vdots$
- how many cases have value  $k$ .

	contents of page					
	1	2	$\dots$	$i$	$\dots$	$\frac{(n+k-1)!}{n!(k-1)!}$
1	$n$	$n-1$		$n_{i,1}$		0
2	0	1		$n_{i,2}$		0
3	0	0		$n_{i,3}$		0
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$
$k$	0	0		$n_{i,k}$		$n$

## Data Description Codebook

Lists for given counts  $n_1, n_2, \dots, n_k$  the value for each of the  $n$  cases:

- value of case 1,
- value of case 2,
- $\vdots$
- value of case  $n$ .

Attention: **not all** possible assignments, which would be  $k^n$ , but only those with counts  $n_1, n_2, \dots, n_k$ .

	contents of page					
	1	2	...	$i$	...	$\frac{n!}{n_1!n_2!\cdots n_k!}$
1	1	1		$v_{i,1}$		$k$
2	1	1		$v_{i,2}$		$k$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$
$n_1$	1	2		$v_{i,n_1}$		
$n_1+1$	2	1		$v_{i,n_1+1}$		$v$
$\vdots$	$\vdots$		$\vdots$	$\vdots$		$\vdots$
$n$	$k$	0		$v_{i,n}$		1

# Pruning Class Rule Sets

- Discarding conditions may lead to **duplicates**, which are **discarded**.
- Rules with an unacceptably high error rate are discarded.
- Since rules may now “overlap” (sometimes more than one rule is applicable), additional rules may be discarded without harming performance.
- The **performance of a (sub)set**  $R$  of rules for a class  $C$  is assessed based on:
  - number of samples covered by  $R$  not belonging to class  $C$ : **false positives**,
  - number of samples not covered by  $R$  belonging to class  $C$ : **false negatives**.
- The assessment relies on the **minimum description length principle**:
  - Idea: Transmit the class assignments from a sender to a receiver.  
(Both sender and receiver are assumed to know the values of the descriptive attributes already.)
  - By transmitting the rules (or rule sets or generally: a model of the data), the receiver can assign classes to all sample cases.
  - We have to indicate the **exceptions**, i.e., where rule predictions are wrong.



# Pruning Class Rule Sets

- The assessment relies on the **minimum description length principle**:
  - Idea: Transmit the class assignments from a sender to a receiver.
  - **description length** = **length of model description** (here: rule set)  
+ **length of data description** (here: exceptions)
  - The model that leads to the smallest total description length is the best.
- Some (more) basics of description length computations / approximations:
  - The number of bits needed to encode an alternative from a probability distribution  $p$  is (at least)  $H(p)$ , where  $H(p)$  is the **Shannon entropy**.
  - The number of bits needed to encode an alternative from a set of size  $k$ , each element of which has the same probability  $\frac{1}{k}$ , is  $\log_2(k)$ .

(This is also known as *Hartley entropy* or *Hartley information* and results as a special case of Shannon entropy:  $H(\mathcal{U}(k)) = -\sum_{i=1}^k \frac{1}{k} \log_2(\frac{1}{k}) = k \frac{1}{k} \log_2(k) = \log_2(k)$ , where  $\mathcal{U}(k)$  is a uniform distribution over  $k$  objects.)
  - The number of bits needed to encode an integer  $k$  is (at least)  $\log_2(k)$  bits.

(This is somewhat sloppy. Better approximations are  $\log_2(m)$  bits, where  $m$  is the largest possible integer, or  $\log_2^*(k) + \log_2(c)$  bits,  $\log_2^*(k) = \log_2(k) + \log_2 \log_2(k) + \log_2 \log_2 \log_2(k) \dots$  and  $c \approx 2.865$  [Rissanen 1983])

# Reminder: 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”.
  - Even the best scheme (*Huffman coding*) needs at least as many questions (in expectation) as the Shannon entropy of the probability distribution.
  - If sequences of (independent) situations are to be processed, one can get closer to this limit by combining consecutive instances.
  - **However, there is no way to get below the Shannon entropy.**

# Pruning Class Rule Sets

- We follow the rule set encoding computations in C4.5 R8. [Quinlan 1993]
- For simplicity, we assess the description length for each class rule set separately. This allows us to skip coding the class, as it is the same for all rules in the set.
- To encode a rule, we must specify the conditions in the antecedent. We start by encoding their number  $k$ , which takes (at least)  $\log_2(k)$  bits.
- For each condition  $C$  we must specify the test carried out (with  $b(C)$  bits).
  - If an attribute  $A$  is nominal, the condition is  $A = a$  and we need  $H(p_A)$  bits, where  $p_A$  is the probability distribution of the attribute's values.
  - If an attribute is metric, we need to specify threshold and comparison, hence we need  $\log_2(k)+1$  bits;  $k$  is the number of possible thresholds. Account for a non-uniform distribution: heuristically adapt to  $\frac{1}{2} \log_2(k)+1$ .

(In both cases we rely on the fact that the receiver already knows the values of the descriptive attributes for all sample cases and hence can compute (an estimate of) the probability distribution of a nominal attribute's values as well as the number of possible thresholds for a metric attribute.)

- Next we have to encode which attribute is tested in a condition.

# Pruning Class Rule Sets

- To specify which attribute is tested in a condition, we need some probability distribution over the attributes.
- In C4.5 R8 this somewhat wild (though not unreasonable) heuristic is used:  
The probability  $p_{\text{cond}}(A)$  that attribute  $A$  appears in a condition is proportional to the number of bits needed to encode a condition on  $A$ .  
Hence specifying the attribute of a condition takes (at least)  $H(p_{\text{cond}})$  bits.
- The condition order is irrelevant (conjunction is commutative and associative). Since there are  $k!$  possible orders of  $k$  conditions, any of which may be used, we account for this by reducing the description length by  $\log_2(k!)$ .
- An analogous argument applies to the rules in a given class rule set. If there are  $r$  rules, we reduce the description length by  $\log_2(r!)$ .
- In summary, the description length of (the model of) a class rule set  $\mathcal{R}$  is:

$$L(\mathcal{M}_{\mathcal{R}}) = \sum_{j=1}^r \left( \log_2(k_j) + \sum_{i=1}^{k_j} \left( H(p_{\text{cond}}) + b(C_{j,i}) \right) - \log_2(k_j!) \right) - \log_2(r!).$$

# Pruning Class Rule Sets

- In addition to the rules, we have to specify the exceptions.  
Let  $n$  be the total number of cases and  $n_{\mathcal{R}}$  the number of cases covered by  $\mathcal{R}$ , that is, the cases to which at least one rule of the rule set  $\mathcal{R}$  is applicable.

- Specifying the exceptions among all sample cases requires

$$L(\mathcal{E}_{\mathcal{R}}) = \log_2 \left( \binom{n_{\mathcal{R}}}{\text{fp}} \right) + \log_2 \left( \binom{n - n_{\mathcal{R}}}{\text{fn}} \right) \text{ bits},$$

where fp and fn are the numbers of false positives and negatives, respectively.

(Intuition: Consider a code book listing all selections of  $k$  cases from a total of  $n$  cases, one selection per page. This book has  $\binom{n}{k}$  pages. Then transmit the page number, assuming all pages (selections) are equally likely.)

- The total description length of a class rule set is then

$$L(\mathcal{R}) = L(\mathcal{M}_{\mathcal{R}}) + L(\mathcal{E}_{\mathcal{R}}) \quad \text{or} \quad L(\mathcal{R}) = w \cdot L(\mathcal{M}_{\mathcal{R}}) + L(\mathcal{E}_{\mathcal{R}}),$$

where  $w < 1$  is a parameter mitigating overestimates of  $L(\mathcal{M}_{\mathcal{R}})$  (e.g.  $w = \frac{1}{2}$ ).

- With this result we can compare different rule sets for the same class.  
The rule set with the shorter description length is the better rule set.

# Pruning Class Rule Sets

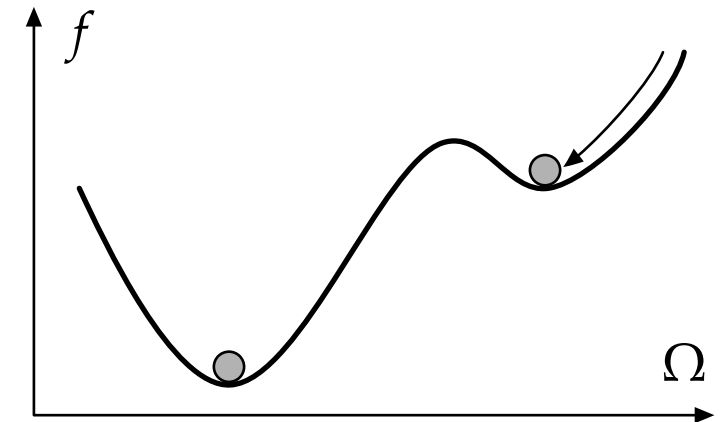
- If there are few rules in a rule set, **all subsets** can be tested.
- Otherwise, a first idea would be to **greedily discard rules** from a rule set as long as this does not increase the (total) description length.  
(Note: This would be in complete analogy to the greedy condition reduction for individual rules.)
- However, this approach does not work so well in practice.  
A better, because less greedy approach, is **simulated annealing**.
  1. Let  $\mathcal{R}_c$  be the set of rules predicting class  $c$ .
  2. Start with the full class rule set  $\mathcal{R} = \mathcal{R}_c$ .
  3. Repeat: (Tentatively) discard a randomly chosen rule  $R \in \mathcal{R}$  from  $\mathcal{R}$  or add a randomly chosen rule  $R \in \mathcal{R}_c - \mathcal{R}$  to  $\mathcal{R}$  and compute the resulting change  $\Delta L$  of the description length.
  4. If  $\Delta L \leq 0$  (new rule set is better or equal), always accept the change.
  5. If  $\Delta L > 0$  (new rule set is worse), accept the change with a probability that depends on  $\Delta L$  and a temperature parameter that is lowered over time.

# Excursion: Simulated Annealing

May be seen as an extension of gradient or random ascent/descent that tries to avoid (or at least reduce the risk of) getting stuck.

**Idea:** transitions from higher to lower (local) minima should be more probable than *vice versa*.

[Metropolis *et al.* 1953; Kirkpatrick *et al.* 1983]



## Principle of Simulated Annealing:

- Random variants of the current solution (candidate) are created.
- Better solution (candidates) are always accepted.
- Worse solution (candidates) are accepted with a probability that depends on
  - the quality difference between the old and the new solution (candidate) and
  - a temperature parameter that is lowered over time.

# Excursion: Simulated Annealing

- **Motivation:**

Physical minimization of energy (to be more specific: the atom lattice energy), if a heated piece of metal is cooled down slowly.

This process is called **annealing**.

It serves the purpose to make a piece of metal easier to work or to machine by releasing internal tensions and instabilities.

(The atom lattice becomes more regular → lower atom lattice energy.)

- **Alternative Motivation:**

A ball rolls around on an unevenly curved (“wavy”) surface.

The function to minimize is the potential energy of the ball.

At the beginning the ball has a certain kinetic energy, due to which it can roll uphill for some distance. However, due to friction between ball and surface the energy of the ball reduces, so that they finally comes to rest in a valley.

- **Attention:** There is *no guarantee* that the global optimum will be found!  
Only the chances are better that a “good” (local) optimum will be found.



# Excursion: Simulated Annealing

1. Choose a (random) starting point  $\omega_0 \in \Omega$ .
2. Choose a (random) point  $\omega' \in \Omega$  “in the vicinity” of the current point  $\omega_i$  (e.g. by a small random variation of  $\omega_i$ ).

3. Set

$$\omega_{i+1} = \begin{cases} \omega', & \text{if } f(\omega') \leq f(\omega_i), \\ & (\geq \text{ for ascent/maximum}) \\ \omega' & \text{with probability } p = e^{-\frac{\Delta f}{kT}}, \\ \omega_i & \text{with probability } 1 - p, \end{cases} \quad \text{otherwise.}$$

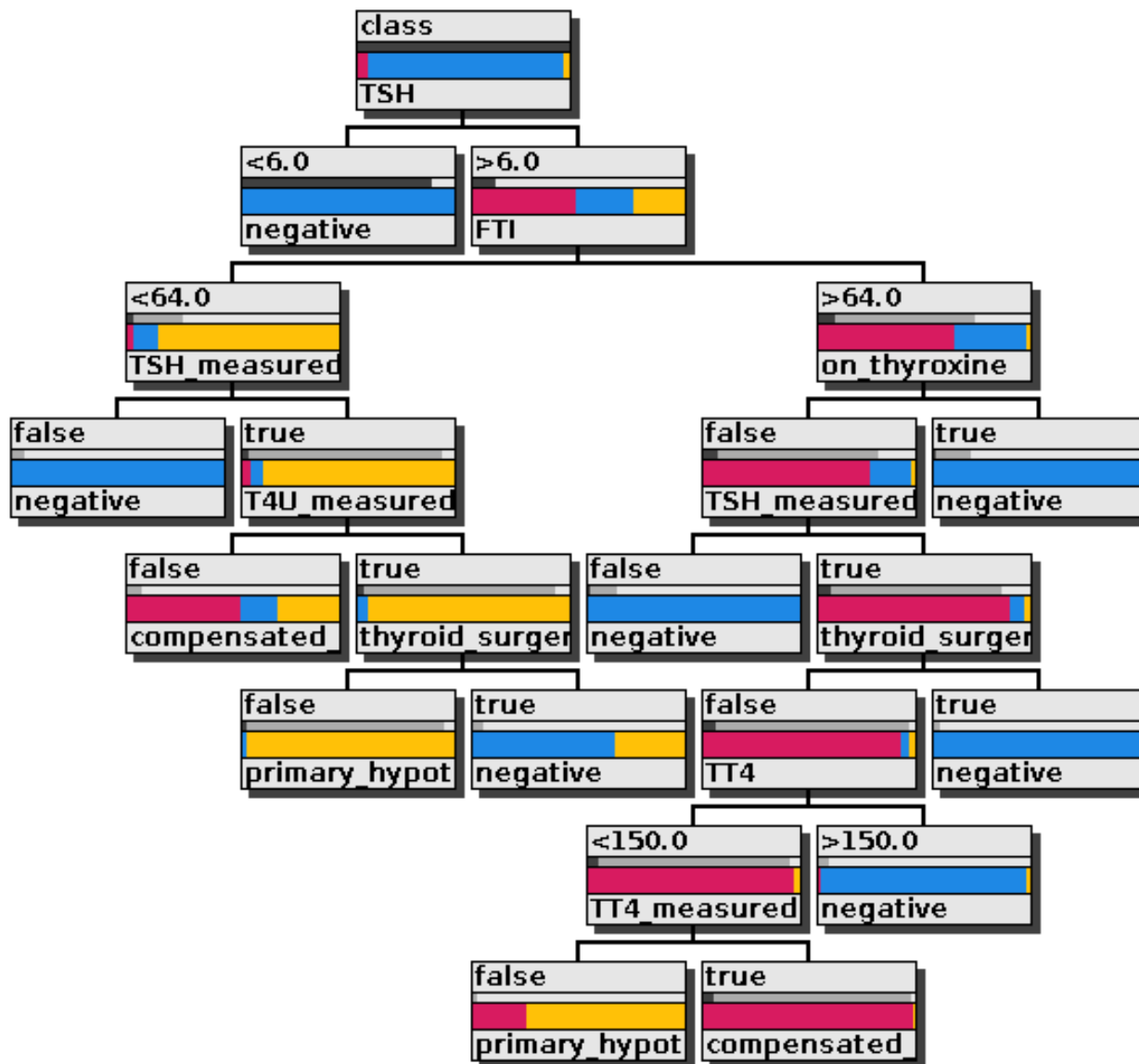
$\Delta f = |f(\omega_i) - f(\omega')|$       quality difference of the solution candidates  
 $k = \Delta f_{\max}$       (estimate of the) range of the function values  
 $T$       temperature parameter; is (slowly) reduced over time

4. Repeat steps 2 and 3, until some termination criterion is satisfied.
- For small  $T$  the method approaches hill climbing (random ascent/descent).  
For larger  $T$  there is still a tendency to improve the solution candidates.

# Ordering Class Rule Sets, Default Class

- Once a subset of each class rule set has been selected, these class rule (sub)sets need to be ordered to obtain a full classifier.
- Recall that each class rule subset causes false positives and false negatives.
- A good heuristic is to defer those rule subsets that cause many false positives, because preceding class rule sets may already have (correctly) covered those.
- As a consequence, the **class rule sets are ordered** as follows:
  - The class rule set with the fewest false positives is chosen as first rule set.
  - False positives (and false negatives) are recomputed on the uncovered cases.
  - The class rule set with the fewest (updated) false positives is chosen next etc.
- Sample cases not covered (and thus not classified) by any rule are handled by a final **default rule** (“If true, then ...”) or **default class**.
- The **default class** is the majority class in the uncovered sample cases, with ties being broken in favor of the class that is more frequent overall.

# Pruning Decision Tree Rules: Example



- Hypothyroid data from Garvan Institute of Medical Research, Darlinghurst, Australia
- 21 binary attributes, 7 metric attributes, 2514 cases, 4 classes:  
primary\_hypothyroid  
secondary\_hypothyroid  
compensated\_hypothyroid  
negative
- Decision tree on the left was induced with C4.5 Release 8 [Quinlan 1993]
- Tree has 11 leaves  
⇒ 11 initial rules

# Pruning Decision Tree Rules: Example

- 1: **If** TSH  $\leq$  6,  
    **then** class = negative.
  - 2: **If** TSH > 6 **and** FTI  $\leq$  64  
    **and** TSH\_measured = false,  
    **then** class = negative.
  - 3: **If** TSH > 6 **and** FTI  $\leq$  64  
    **and** TSH\_measured = true  
    **and** T4U\_measured = false,  
    **then** class = compensated  
        hypothyroid.
  - 4: **If** TSH > 6 **and** FTI  $\leq$  64  
    **and** TSH\_measured = true  
    **and** T4U\_measured = true  
    **and** thyroid\_surgery = false,  
    **then** class = primary  
        hypothyroid.
  - 5: **If** TSH > 6 **and** FTI  $\leq$  64  
    **and** TSH\_measured = true  
    **and** T4U\_measured = true  
    **and** thyroid\_surgery = true,  
    **then** class = negative.
  - 6: **If** TSH > 6 **and** FTI > 64  
    **and** on\_thyroxine = false  
    **and** TSH\_measured = false,  
    **then** class = negative.
  - 7: **If** TSH > 6 **and** FTI > 64  
    **and** on\_thyroxine = false  
    **and** TSH\_measured = true  
    **and** thyroid\_surgery = false  
    **and** TT4  $\leq$  150  
    **and** TT4\_measured = false,  
    **then** class = primary  
        hypothyroid.
  - 8: **If** TSH > 6 **and** FTI > 64  
    **and** on\_thyroxine = false  
    **and** TSH\_measured = true  
    **and** thyroid\_surgery = false  
    **and** TT4  $\leq$  150  
    **and** TT4\_measured = true,  
    **then** class = compensated  
        hypothyroid.
  - 9: **If** TSH > 6 **and** FTI > 64  
    **and** on\_thyroxine = false  
    **and** TSH\_measured = true  
    **and** thyroid\_surgery = false  
    **and** TT4 > 150,  
    **then** class = negative.
  - 10: **If** TSH > 6 **and** FTI > 64  
    **and** on\_thyroxine = false  
    **and** TSH\_measured = true  
    **and** thyroid\_surgery = true,  
    **then** class = negative.
  - 11: **If** TSH > 6 **and** FTI > 64  
    **and** on\_thyroxine = true,  
    **then** class = negative.
- Unpruned rules and unpruned rule set,  
as extracted from the decision tree.

# Pruning Decision Tree Rules: Example

- We consider the simplification of the rule: (rule 5 on preceding slide)

**If** TSH > 6 **and** FTI ≤ 64 **and** TSH\_measured = true **and** T4U\_measured = true **and** thyroid\_surgery = true, **then** class = negative.

- The full rule is applicable to three sample cases; it produces one error. (conf. level  $1 - \alpha = 0.75$ )

condition deleted	negative $n_{01} + n_{11}$	other $n_{00} + n_{10}$	pessimistic error rate
none	2	1	65.0%

- Each of the 5 conditions is considered in turn.
- Without “FTI ≤ 64” the pessimistic error rate is 35%.
- Since this error rate is better, the condition is deleted.

condition deleted	negative $n_{01} + n_{11}$	other $n_{00} + n_{10}$	pessimistic error rate
TSH > 6	3	1	53.7%
FTI ≤ 64	6	1	35.0%
TSH_measured = true	2	1	65.0%
T4U_measured = true	2	1	65.0%
thyroid_surgery = true	3	59	97.5%

- The reduced rule (after one simplification step) is:

**If** TSH > 6 **and** TSH\_measured = true **and** T4U\_measured = true **and** thyroid\_surgery = true, **then** class = negative.

# Pruning Decision Tree Rules: Example

- Each of the 4 remaining conditions is considered in turn with updated counts.

condition deleted	negative $n_{01} + n_{11}$	other $n_{00} + n_{10}$	pessimistic error rate
TSH > 6	31	1	8.9%
TSH_measured = true	6	1	35.0%
T4U_measured = true	7	1	31.4%
thyroid_surgery = true	44	179	83.2%

- Without "TSH > 6" the pessimistic error rate is 8.9%.

- The reduced rule (after two simplification steps) is:  
If TSH\_measured = true **and** T4U\_measured = true **and** thyroid\_surgery = true, **then** class = negative.

- Each of the remaining 3 conditions is checked.

condition deleted	negative $n_{01} + n_{11}$	other $n_{00} + n_{10}$	pessimistic error rate
TSH_measured = true	31	1	8.9%
T4U_measured = true	34	1	8.2%
thyroid_surgery = true	1962	179	9.1%

- w/o "T4U\_measured = true":  
pessimistic error rate 8.2%.

- The reduced rule (after three simplification steps) is:  
If TSH\_measured = true **and** thyroid\_surgery = true, **then** class = negative.

# Pruning Decision Tree Rules: Example

- Each of the remaining 2 conditions is checked.

condition deleted	negative $n_{01} + n_{11}$	other $n_{00} + n_{10}$	pessimistic error rate
TSH_measured = true	34	1	8.2%
thyroid_surgery = true	2064	194	9.3%

- w/o “TSH\_measured = true”: pessimistic error rate 8.2%.

- The reduced rule (after four simplification steps) is:  
**If thyroid\_surgery = true, then class = negative.**

- Removing the remaining condition is also checked.

condition deleted	negative $n_{01} + n_{11}$	other $n_{00} + n_{10}$	pessimistic error rate
thyroid_surgery = true	2320	194	8.35%

- w/o “thyroid\_surgery = true”: pessimistic error rate 8.35%.

- Removing the last condition worsens the pessimistic error rate estimate. Therefore this condition is kept and the simplification process terminates.

- The final rule is: **If thyroid\_surgery = true, then class = negative.**  
(Note that this rule covers 35 cases and produces one error.)

# Pruning Decision Tree Rules: Example

1: If  $TSH \leq 6$ ,  
then class = negative.

2: If  $TSH\_measured = false$ ,  
then class = negative.

3: If  $TSH > 6$ ,  
then class = compensated  
hypothyroid.

4: If  $TSH > 6$  and  $FTI \leq 64$   
and  $thyroid\_surgery = false$ ,  
then class = primary  
hypothyroid.

5: If  $thyroid\_surgery = true$ ,  
then class = negative.

6: If  $FTI > 64$ ,  
then class = negative.

7: If  $TSH > 6$   
and  $on\_thyroxine = false$   
and  $TT4\_measured = false$ ,  
then class = primary  
hypothyroid.

8: If  $TSH > 6$  and  $FTI > 64$   
and  $on\_thyroxine = false$   
and  $thyroid\_surgery = false$   
and  $TT4 \leq 150$ ,  
then class = compensated  
hypothyroid.

9: If  $TT4 > 150$ ,  
then class = negative.

10: If  $FTI > 64$ ,  
then class = negative.

11: If  $FTI > 64$   
and  $on\_thyroxine = true$ ,  
then class = negative.

Note: Rules 6 and 10  
have become identical.

Rules pruned by checking all condition subsets.

- Some of the original rules are actually inapplicable, that is, there are no sample cases that satisfy the antecedent of the rule.
- Such rules can result from the **handling of missing values** in the data.



# Pruning Decision Tree Rules: Example

Considered up to now:

3: **If** TSH > 6 **and** FTI ≤ 64  
    **and** TSH\_measured = true  
    **and** T4U\_measured = false,  
    **then** class = compensated  
        hypothyroid.

The additional rule  
helps illustrating  
rule set pruning.

In C4.5 book, rule is split:

3.1: **If** TSH > 6 **and** FTI ≤ 64  
    **and** TSH\_measured = true  
    **and** T4U\_measured = false  
    **and** TSH ≤ 17,  
    **then** class = compensated  
        hypothyroid.

3.2: **If** TSH > 6 **and** FTI ≤ 64  
    **and** TSH\_measured = true  
    **and** T4U\_measured = false  
    **and** TSH > 17,  
    **then** class = primary  
        hypothyroid.

After simplification:

3.1: **If** TSH > 6  
    **and** TSH ≤ 17,  
    **then** class = compensated  
        hypothyroid.

3.2: **If** FTI ≤ 64  
    **and** TSH > 17,  
    **then** class = primary  
        hypothyroid.

We consider simplifying the rule set for the class “primary hypothyroid”:

3.2: **If** FTI ≤ 64  
    **and** TSH > 17,  
    **then** class = primary  
        hypothyroid.

4: **If** TSH > 6 **and** FTI ≤ 64  
    **and** thyroid\_surgery = false,  
    **then** class = primary  
        hypothyroid.

7: **If** TSH > 6  
    **and** on\_thyroxine = false  
    **and** TT4\_measured = false,  
    **then** class = primary  
        hypothyroid.

# Pruning Decision Tree Rules: Example

- The different rule subset selections yield the following coding costs:

model		exceptions				total
selected rules	coding cost	covered cases	false positives	false negatives	coding cost	coding cost
—	0.0	0	0	64	425.8	425.8
3.2	17.2	54	2	12	116.8	125.4
4	19.9	59	1	6	64.0	73.9
7	15.8	4	1	61	411.8	419.7
3.2, 4	36.1	62	3	5	64.6	82.6
3.2, 7	32.0	58	3	9	97.8	113.8
4, 7	34.7	63	2	3	42.1	<b>59.5</b>
3.2, 4, 7	50.3	66	4	2	41.0	66.1

- The selected rules are:
 

4: If TSH > 6 and FTI ≤ 64  
and thyroid\_surgery = false,  
then class = primary  
hypothyroid.

7: If TSH > 6  
and on\_thyroxine = false  
and TT4 measured = false,  
then class = primary  
hypothyroid.

# Pruning Decision Tree Rules: Example

- Pruning the rules for class “primary hypothyroid” discards rule 3.2.  
There are no rules for class “secondary hypothyroid”. (only one case)  
Pruning the rules for class “compensated hypothyroid” discards rule 3 (or 3.1).  
Pruning the rules for class “negative” discards rules 6 and 10. (identical rules)

class	simplified rules	selected rules	covered cases	false positives	false negatives
primary	2	2	66	2	3
secondary	0	0	0	0	1
compensated	2	1	120	0	9
negative	6	5	2316	2	3

- Based on this table we may select the order compensated  $\prec$  primary  $\prec$  negative.
- The final rules leave 5 cases uncovered: 2 primary, 2 compensated, 1 negative.  
Since overall there are more “compensated” cases than “primary” cases,  
the default class is “compensated hypothyroid”.

# Pruning Decision Tree Rules: Example

If on\_thyroxine = false  
and thyroid\_surgery = false  
and TSH > 6  
and TT4 ≤ 150  
and FTI > 64,  
    **then** class = compensated  
                    hypothyroid

If thyroid\_surgery = false  
and TSH > 6  
and FTI ≤ 64  
    **then** class = primary  
                    hypothyroid

If on\_thyroxine = false  
and TT4\_measured = false  
and TSH > 6,  
    **then** class = primary  
                    hypothyroid

If TSH ≤ 6,  
    **then** class = negative

If on\_thyroxine = true  
and FTI > 64,  
    **then** class = negative

If TSH\_measured = false,  
    **then** class = negative

If TT4 > 150,  
    **then** class = negative

If thyroid\_surgery = true,  
    **then** class = negative

If true,  
    **then** class = compensated  
                    hypothyroid

This last rule is the  
“default rule”, which  
applies if none of the  
other rules is applicable.

- Instead of 11 rules, the final rule set has only 9 rules (including the default rule), most of which are considerably simpler than the original rules.

# Direct Rule Learning

# Inducing Rules Directly

- Direct rule induction usually proceeds by successive specialization.
- General procedure of **successive specialization**:
  - Start with a rule with an empty antecedent (or “true” as the antecedent).
  - Specialize the current (set of) rule(s) by adding a condition (to each rule).
  - Limit the resulting set of rules to the  $k$  “best” rules (“beam search”).
  - Repeat specializing the rules and limiting them to a subset, until some quality objective is reached; then select the “best” rule.
  - Remove all covered cases, and start over if uncovered cases remain.
- Different algorithms based on this general scheme differ by
  - the criteria used to assess the quality of a rule,
  - whether rules are induced for a specific, pre-defined class, or whether the class is determined from the covered cases.
- In the following we consider the **A<sup>q</sup> algorithm** and the **CN2 algorithm**.

# Inducing Rules Directly

- The **conditions** with which rules can be specialized (also called *selectors*) are determined from the values of the attributes occurring in the data set.
  - If  $A$  is a nominal attribute, the possible conditions are “ $A = a$ ” with  $a \in \text{dom}(A)$ .
  - If  $A$  is a metric attribute, let  $a_1, \dots, a_\ell \in \mathbb{R}$  be a sorted list of (distinct) values that occur for attribute  $A$  in the given data set. Then the possible conditions are “ $A \leq \frac{1}{2}(a_i + a_{i+1})$ ” and “ $A > \frac{1}{2}(a_i + a_{i+1})$ ” for  $i = 1, \dots, \ell - 1$ .
- In practice, it is usually impossible to check all combinations of conditions, because their number grows exponentially with the number of attributes.
- Therefore one has to limit the combinations of conditions that are explored.
- The most common approach is a **beam search** (with beam size  $k$ ), which limits the search to the  $k$  “best” conjunctions in each step.
- **Greedy search** is a special beam search with beam size 1 (that is,  $k = 1$ ).  
**Widening** tries to select good, but “different” rules (instead of simply the best).

# A<sup>q</sup> Algorithm

- The **A<sup>q</sup> algorithm** [Michalski 1969] induces **rules for a specific class**.  
(If there are multiple classes, each class is treated separately.)
- The algorithm takes as **input** two sets of examples / sample cases:
  - a set  $D_{\text{pos}}$  of positive examples (cases belonging to the target class).
  - a set  $D_{\text{neg}}$  of negative examples (cases not belonging to the target class).
- Since the target class is fixed, **only rule antecedents need to be found**.
- The A<sup>q</sup> algorithm tries to find a conjunction of conditions (also called a *complex*) that covers as many positive examples as possible, but no negative examples.
- The A<sup>q</sup> algorithm employs two quality criteria for conjunctions of conditions:
  - $Q_1$  for **limiting the search set**:  
sum of positive examples covered and negative examples excluded,
  - $Q_2$  for **selecting the best rule**: number of positive examples covered.
- Termination: No rule in the search set covers a negative example.



# A<sup>q</sup> Algorithm

```
function Aq ( $D_{\text{pos}}, D_{\text{neg}}, k$ ) : set of conjunctions          (*  $k$ : beam size *)
   $C \leftarrow \emptyset$                                            (* initialize a set of conjunctions, the cover *)
  while  $D_{\text{pos}} \neq \emptyset$  do                                (* while not all positive examples covered *)
     $d_{\text{pos}} \leftarrow$  choose randomly from  $D_{\text{pos}}$           (* choose a positive seed example *)
     $S \leftarrow \text{star}(d_{\text{pos}}, D_{\text{pos}}, D_{\text{neg}}, k)$           (* get a set of conjunctions that cover  $d_{\text{pos}}$ , *)
                                                                (* but that cover no examples in  $D_{\text{neg}}$  *)
     $s_{\text{best}} \leftarrow \text{argmax}_{s \in S} Q_2(s, D_{\text{pos}})$       (*  $s_{\text{best}}$  covers most positive examples *)
     $C \leftarrow C \cup \{s_{\text{best}}\}$                             (* add best conjunction to the cover *)
     $D_{\text{pos}} \leftarrow \{d \in D_{\text{pos}} \mid s_{\text{best}} \not\triangleleft d\}$   (* remove covered examples *)
  return  $C$                                                     (* return the found set of conjunctions *)
```

- The notation “ $s \triangleleft d$ ” means that  $s$  covers  $d$ , i.e., the conjunction  $s$  is true for  $d$ .
- The choice of a (positive) seed example introduces a random element, so that two runs of the algorithm may yield different results.
- Note that this version only produces rule antecedents (conjunctions). Rules are obtained by generating “ $s \rightarrow \text{class} = \text{target}$ ” for all  $s \in C$ .

# A<sup>9</sup> Algorithm

```
function star ( $d_{\text{pos}}, D_{\text{pos}}, D_{\text{neg}}, k$ ) : set of conjunctions          (*  $k$ : beam size *)
   $S \leftarrow \{\text{true}\}$                                           (* initialize a set of conjunctions *)
  while true do                                              (* conjunction specialization loop *)
     $D_{\text{cn}} \leftarrow \{d \in D_{\text{neg}} \mid \exists s \in S: s \triangleleft d\}$   (* get covered negative examples *)
    if  $D_{\text{cn}} = \emptyset$  then break                          (* if no neg. examples covered, abort *)
     $d_{\text{neg}} \leftarrow$  choose randomly from  $D_{\text{cn}}$           (* get a covered negative example *)
     $C_{\text{ext}} \leftarrow \{c \in C_{\text{all}} \mid c \triangleleft d_{\text{pos}} \wedge c \not\triangleleft d_{\text{neg}}\}$   (* get conditions excluding  $d_{\text{neg}}$  *)
    for  $s \in S$  do                                          (* traverse the current conjunctions *)
      if  $s \triangleleft d_{\text{neg}}$  then                             (* if conjunction covers  $d_{\text{neg}}$  *)
         $S_{\text{ext}} \leftarrow \{s \wedge c \mid c \in C_{\text{ext}}\}$       (* specialize conjunction  $s$  *)
         $S \leftarrow (S - \{s\}) \cup S_{\text{ext}}$                 (* replace  $s$  by its specializations *)
    while  $|S| > k$  do                                       (* while more conj. than beam size *)
       $S \leftarrow S - \{\text{argmin}_{s \in S} Q_1(s, D_{\text{pos}}, D_{\text{neg}})\}$   (* remove worst conj. in  $S$  *)
  return  $S$                                                   (* return found set of conjunctions *)
```

- The choice of a (negative) covered example introduces a random element, so that two runs of the algorithm may yield different results.

# A<sup>q</sup> Algorithm: Example

## Patient database

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

## Objective: Assignment of Drug

- Find rules for one of the classes, either Drug A or Drug B, by applying the A<sup>q</sup> algorithm.
- Or find rules for both classes, Drug A and Drug B, by applying the A<sup>q</sup> algorithm twice.

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

- If only one class is covered by rules, the other can be handled by a default rule.
- Here, rules for both class will be created using a greedy approach ( $k = 1$ ).

# A<sup>9</sup> Algorithm: Example

- Get **all possible conditions** that may be used in rule antecedents:
  - Sex = female  
Sex = male
  - Age  $\leq$  23.0      Age  $\leq$  45.0      Age  $>$  23.0      Age  $>$  45.0  
Age  $\leq$  27.5      Age  $\leq$  50.0      Age  $>$  27.5      Age  $>$  50.0  
Age  $\leq$  29.5      Age  $\leq$  53.0      Age  $>$  29.5      Age  $>$  53.0  
Age  $\leq$  31.5      Age  $\leq$  57.5      Age  $>$  31.5      Age  $>$  57.5  
Age  $\leq$  35.0      Age  $\leq$  67.0      Age  $>$  35.0      Age  $>$  67.0  
Age  $\leq$  39.5           Age  $>$  39.5
  - Blood pressure = low  
Blood pressure = normal  
Blood pressure = high
- Start with class “Drug A” and split data into **positive and negative examples**:
  - $D_{\text{pos}}$  : cases 1, 3, 5, 6, 10, 12      ( $n_{\text{pos}}$  = 6 cases where class is “Drug A”)
  - $D_{\text{neg}}$  : cases 2, 4, 7, 8, 9, 11      ( $n_{\text{neg}}$  = 6 cases where class is “Drug B”)

# A<sup>9</sup> Algorithm: Example

- Start with the empty conjunction:  $S = \{\text{true}\}$ .
- Select a positive example (seed example):  $d_{\text{pos}} = [\text{female}, 37, \text{high}, A]$
- Select a covered negative example:  $d_{\text{neg}} = [\text{male}, 61, \text{normal}, B]$
- Get conditions that cover example  $d_{\text{pos}}$ , but not example  $d_{\text{neg}}$ :

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q = m_{\text{pos}} + (n_{\text{neg}} - m_{\text{neg}})$
Sex = female	3	3	6
Age $\leq 39.5$	4	2	8
Age $\leq 45.0$	4	3	7
Age $\leq 50.0$	5	3	8
Age $\leq 53.0$	5	4	7
Age $\leq 57.5$	6	4	8
Blood pressure = high	3	<b>0</b>	<b>9</b>

- After restriction to beam size  $k = 1$  (greedy approach):  
 $S = \{ \text{Blood pressure} = \text{high} \}$

# A<sup>q</sup> Algorithm: Example

- Since the current search set (beam)  $S = \{ \text{Blood pressure} = \text{high} \}$  does not cover any negative examples, the first rule is:
  - If Blood pressure = high, then Drug = A.
- The remaining uncovered positive cases are:
  - $D_{\text{pos}}$  : cases 1, 6, 10      ( $n_{\text{pos}} = 3$  cases)
- Start over with the empty conjunction:  $S = \{\text{true}\}$ .
- Select a positive example (seed example):     $d_{\text{pos}} = [\text{male}, 20, \text{normal}, \text{A}]$
- Select a covered negative example:             $d_{\text{neg}} = [\text{female}, 52, \text{normal}, \text{B}]$
- Get conditions that cover example  $d_{\text{pos}}$ , but not example  $d_{\text{neg}}$ :

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Sex = male	2	3	5
Age $\leq 23.0$	1	0	7
Age $\leq 27.5$	1	1	6

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Age $\leq 29.5$	2	1	6
Age $\leq 31.5$	3	1	8
Age $\leq 35.0$	3	2	7

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Age $\leq 39.5$	3	2	7
Age $\leq 50.0$	3	3	6
Age $\leq 45.0$	3	3	6

# A<sup>q</sup> Algorithm: Example

- After restriction to beam size  $k = 1$  (greedy approach):  
 $S = \{ \text{Age} \leq 31.5 \}$
- Select a covered negative example:  $d_{\text{neg}} = [\text{female}, 26, \text{low}, \text{B}]$
- Get conditions that cover example  $d_{\text{pos}}$ , but not example  $d_{\text{neg}}$ :

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q = m_{\text{pos}} + (n_{\text{neg}} - m_{\text{neg}})$
Sex = male	2	0	8
Blood pressure = normal	3	0	9

- After restriction to beam size  $k = 1$  (greedy approach):  
 $S = \{ \text{Age} \leq 31.5 \wedge \text{Blood pressure} = \text{normal} \}$
- Since this does not cover any negative examples, the second rule is:
  - If  $\text{Age} \leq 31.5$  and  $\text{Blood pressure} = \text{normal}$ , then  $\text{Drug} = \text{A}$ .
- There are no uncovered positive cases left, hence the algorithm terminates.

# A<sup>q</sup> Algorithm: Example

- The other class (Drug B) may be covered by a default rule:
  - If true, then Drug = B.

Note that this introduces an order of the rules, at least grouped by the predicted class.
- Alternatively, the algorithm may be run again for the other class (Drug B):
  - $D_{\text{pos}}$  : cases 2, 4, 7, 8, 9, 11 ( $n_{\text{pos}} = 6$  cases)
  - $D_{\text{neg}}$  : cases 1, 3, 5, 6, 10, 12 ( $n_{\text{neg}} = 6$  cases)
- Start with the empty conjunction:  $S = \{\text{true}\}$ .
- Select a positive example (seed example):  $d_{\text{pos}} = [\text{male}, 42, \text{low}, \text{B}]$
- Select a covered negative example:  $d_{\text{neg}} = [\text{male}, 29, \text{normal}, \text{A}]$
- Get conditions that cover example  $d_{\text{pos}}$ , but not example  $d_{\text{neg}}$ :

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Age > 29.5	5	4	7
Age > 31.5	5	3	8

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Age > 35.0	4	3	7
Age > 39.5	4	2	8

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Blood pr. = low	3	0	9



# A<sup>9</sup> Algorithm: Example

- After restriction to beam size  $k = 1$  (greedy approach):  
 $S = \{ \text{Blood pressure} = \text{low} \}$
- Since the current search set (beam)  $S = \{ \text{Blood pressure} = \text{low} \}$  does not cover any negative examples, the first rule is:
  - If Blood pressure = low, then Drug = B.
- The remaining uncovered positive cases are:
  - $D_{\text{pos}} : \text{cases } 2, 7, 9 \quad (n_{\text{pos}} = 3 \text{ cases})$
- Start over with the empty conjunction:  $S = \{\text{true}\}$ .
- Select a positive example (seed example):  $d_{\text{pos}} = [\text{female}, 52, \text{normal}, \text{B}]$
- Select a covered negative example:  $d_{\text{neg}} = [\text{male}, 54, \text{high}, \text{A}]$

# A<sup>q</sup> Algorithm: Example

- Get conditions that cover example  $d_{\text{pos}}$ , but not example  $d_{\text{neg}}$ :

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Sex = female	2	3	5
Age $\leq 53$	1	5	2
Blood pressure = normal	3	3	6

- After restriction to beam size  $k = 1$  (greedy approach):  
 $S = \{ \text{Blood pressure} = \text{normal} \}$
- Select a covered negative example:  $d_{\text{neg}} = [\text{male}, 29, \text{normal}, A]$
- Get conditions that cover example  $d_{\text{pos}}$ , but not example  $d_{\text{neg}}$ :

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Sex = female	2	1	7

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Age $> 29.5$	3	1	8
Age $> 31.5$	3	0	9
Age $> 35.0$	3	0	9

condition	$m_{\text{pos}}$	$m_{\text{neg}}$	$q$
Age $> 39.5$	3	0	9
Age $> 45.0$	3	0	9
Age $> 50.0$	3	0	9

- Since there is no unique best condition, one is chosen arbitrarily.

# A<sup>9</sup> Algorithm: Example

- After restriction to beam size  $k = 1$  (greedy approach):  
 $S = \{ \text{Age} > 45 \wedge \text{Blood pressure} = \text{normal} \}$
- Since this does not cover any negative examples, the second rule is:
  - If  $\text{Age} > 45$  and  $\text{Blood pressure} = \text{normal}$ , then  $\text{Drug} = \text{B}$ .
- There are no uncovered positive cases left, hence the algorithm terminates.
- The **final set of rules** (for both classes together) is:
  - If  $\text{Blood pressure} = \text{high}$ , then  $\text{Drug} = \text{A}$ .
  - If  $\text{Blood pressure} = \text{normal}$  and  $\text{Age} \leq 31.5$ , then  $\text{Drug} = \text{A}$ .
  - If  $\text{Blood pressure} = \text{low}$ , then  $\text{Drug} = \text{B}$ .
  - If  $\text{Blood pressure} = \text{normal}$  and  $\text{Age} > 45.0$ , then  $\text{Drug} = \text{B}$ .
- A default rule predicting either class may be added.
- Note that this rule set corresponds very closely to the rule set that was extracted from the corresponding decision tree (cf. slide 21).

# A<sup>q</sup> Algorithm: Discussion

- Note that the A<sup>q</sup> algorithm assumes **noise-free data**.
  - If there are **contradictions**, i.e., if two sample cases differ only in their class, the algorithm (in particular, the function “star”) cannot terminate, because there is no conjunction  $s$  covering  $d$  (because the case is in  $D_{\text{pos}}$ ) and at the same time excluding  $d$  (because the case is in  $D_{\text{neg}}$ ).
  - If there are no contradictions, but **noise** (a class and its complement overlap), the A<sup>q</sup> algorithm may generate very many rules, with each rule covering only a very small number of examples (maybe even just one).

The result **overfits the data** and thus **does not generalized well**.

- The core problem is that the A<sup>q</sup> algorithm (in the version presented here) **does not allow for misclassifications in the training data**.
- Any noise-tolerant rule induction algorithm must accept some such misclassifications in order to avoid overfitting.
- The **CN2 algorithm** does this by replacing the pure (positive) class objective by an objective based on the Shannon entropy of the class distribution.

# CN2 Algorithm

- The **CN2 algorithm** does **not** require a target class for which rules are induced.
- Rather it tries to find conjunctions of conditions (rule antecedents) that lead to a class distribution with low Shannon entropy (but maybe  $> 0$ ).  
This **allows for misclassifications of the training data**.
- Rule consequents are chosen as the **majority class of the covered examples**.
- The CN2 algorithm also employs two quality criteria for rules:
  - $Q_1$  is the Shannon entropy of the class distribution of covered examples,
  - $Q_2$  is a  $p$ -value quantifying the statistical significance of a rule.
- The statistical significance of a rule (or rather its antecedent) is computed by comparing the **class distribution of all examples** (in the given data set) to the **class distribution of the covered examples** (i.e., antecedent is true).
  - Idea: If the class distribution of the covered examples is likely to occur for a random selection of examples, the rule antecedent is not significant.

## Excursion: Likelihood Ratio Statistic / G Statistic

- Suppose we have a probability model with a parameter  $\theta$  [Kalbfleisch 1979] and want to test a hypothesis about the value of  $\theta$ .

In such a case a **likelihood ratio** can often be used for the significance test.

- Recall: The **likelihood function** describes the probability of observed data  $D$  as a function of a (possibly multivariate) parameter  $\theta$  and is written as

$$\mathcal{L}(\theta \mid D) = P(D \mid \theta) \quad \text{or} \quad \mathcal{L}(\theta \mid D) = P(D; \theta).$$

- Consider, for example, that we want to test the (simple) hypothesis  $\theta = \theta_0$ . Then the **likelihood ratio** for  $\theta = \theta_0$  versus the most likely value  $\theta = \hat{\theta}$  is

$$\mathcal{R}(\theta_0; D) = \frac{\mathcal{L}(\theta_0 \mid D)}{\mathcal{L}(\hat{\theta} \mid D)} = \frac{\mathcal{L}(\theta_0 \mid D)}{\sup_{\theta \in \Theta} \mathcal{L}(\theta \mid D)} = \frac{P(D \mid \theta_0)}{\sup_{\theta \in \Theta} P(D \mid \theta)},$$

where  $\Theta$  is the **parameter space** (that is, the set of possible values for  $\theta$ ).

- Note that the denominator is the **maximum likelihood estimate** for  $\theta$ .
- $\mathcal{R}(\theta_0; D)$  is also called the **relative likelihood** of  $\theta = \theta_0$ .

## Excursion: Likelihood Ratio Statistic / G Statistic

- If we fix  $\theta$  at its hypothesized value  $\theta_0$ , we can use  $\mathcal{R}(\theta_0; D)$  to **rank possible data sets**  $D$ .
  - If  $\mathcal{R}(\theta_0; D)$  is near 1, then the data  $D$  gives the hypothesized value  $\theta_0$  a high relative likelihood, and is in good agreement with  $H: \theta = \theta_0$ .
  - If  $\mathcal{R}(\theta_0; D)$  is near 0, then the data  $D$  makes the hypothesized value  $\theta_0$  very unlikely, and hence  $D$  is in poor agreement with  $H: \theta = \theta_0$ .
- If the hypothesis is compound (that is,  $H: \theta \in \Theta_0$  with  $|\Theta_0| > 1$ ), one may use the **maximized likelihood ratio**.

$$\mathcal{R}(\Theta_0; D) = \frac{\mathcal{L}(D \mid \Theta_0)}{\mathcal{L}(D \mid \Theta)} = \frac{\sup_{\theta \in \Theta_0} P(D \mid \Theta_0)}{\sup_{\theta \in \Theta} P(D \mid \Theta)},$$

- If the hypothesis is simple (that is,  $H: \theta \in \Theta_0$  with  $|\Theta_0| = 1$ , i.e.,  $\theta = \theta_0$ ), this can be seen as a special case of the maximized likelihood ratio.
- Large values of  $\mathcal{R}(\Theta_0; D)$  correspond to data  $D$  in good agreement with  $H$ , small values of  $\mathcal{R}(\Theta_0; D)$  correspond to data  $D$  in poor agreement with  $H$ .

## Excursion: Likelihood Ratio Statistic / G Statistic

- A test of significance in which the (maximized) likelihood ratio is used to rank data is called a **likelihood ratio test**.
- Any strictly decreasing function of  $\mathcal{R}(\Theta_0; D)$  may be selected as the **test statistic** for such a likelihood test. A convenient choice is

$$\mathcal{D}(\Theta_0; D) = -2 \ln(\mathcal{R}(\Theta_0; D)).$$

- Suppose that under an unconstrained model the vector of unknown parameters can take any value in a parameter space  $\Theta$  with dimensionality  $r$ .
- Furthermore, suppose that the hypothesis  $H$  restricts  $\theta$  to a subspace  $\Theta_0$  of  $\Theta$  with dimensionality  $r_0 < r$  ( $r_0$  unknown parameters under the hypothesis  $H$ ).
- According to **Wilks' Theorem** [Wilks 1938], in large samples and under suitable regularity conditions, the test statistic  $\mathcal{D}(\Theta_0; D)$  is approximately  $\chi^2$  distributed with  $r - r_0$  degrees of freedom:

$$\mathcal{D}(\Theta_0; D) = -2 \ln(\mathcal{R}(\Theta_0; D)) \approx \chi_{r-r_0}^2.$$



## Excursion: Likelihood Ratio Statistic / G Statistic

- We now apply this approach to assessing the **significance of a rule**.
- The parameter  $\theta$  is multivariate and describes a **multinomial distribution**, namely the **distribution of the possible classes**.
- Its hypothesized value  $\theta_0$  is the **class distribution in the whole data set**. This we estimate from the whole data set as  $\theta_0 = (\theta_{01}, \dots, \theta_{0c})$  with

$$\theta_{0i} = \frac{n_i}{n} \quad \text{for } i = 1, \dots, c,$$

where  $c$  is the number of classes,  $n$  the total number of examples, and  $n_i$  the number of examples in class  $c_i$ . (obviously  $\sum_{i=1}^c n_i = n$ )

(This is the maximum likelihood estimator for the  $\theta_{0i}$  (see also below), which happens to be a consistent, unbiased, maximally efficient, and sufficient estimator.)

- In this case it is  $r_0 = 0$ , since there is only one possible value (vector)  $\theta = \theta_0$ .
- The unconstrained model has dimensionality  $r = c - 1$  (**not**  $r = c$ ), because  $\theta$  is in the probability simplex  $\Theta = \Delta_{c-1}$  due to  $\sum_{i=1}^c \theta_i = 1$ .

## Excursion: Likelihood Ratio Statistic / G Statistic

- We now consider the set of examples  $D_R$  covered by a rule  $R$ , where  $m = |D_R|$  and  $m_i$  is the number of covered examples in class  $i$ ,  $i = 1, \dots, c$ .
- The **likelihood function** for the parameter  $\theta = (\theta_1, \dots, \theta_c) \in \Theta$  is

$$\mathcal{L}(\theta \mid D_R) = \theta_1^{m_1} \cdot \theta_2^{m_2} \cdots \theta_c^{m_c} = \prod_{i=1}^c \theta_i^{m_i}.$$

- This likelihood function is maximized for (**maximum likelihood estimate**)

$$\hat{\theta}_i = \frac{m_i}{m} \quad \text{for } i = 1, \dots, c.$$

- Therefore the (**maximized**) **likelihood ratio** is

$$\mathcal{R}(\theta_0 \mid D_R) = \frac{\mathcal{L}(\theta_0 \mid D)}{\sup_{\theta \in \Theta} \mathcal{L}(\theta \mid D)} = \frac{\mathcal{L}(\theta_0 \mid D)}{\mathcal{L}(\hat{\theta} \mid D)} = \frac{\prod_{i=1}^c \theta_{0i}^{m_i}}{\prod_{i=1}^c \hat{\theta}_i^{m_i}} = \prod_{i=1}^c \left( \frac{m \theta_{0i}}{m_i} \right)^{m_i}.$$

- The **likelihood ratio statistic**, which for this case is a.k.a. the **G statistic**, is

$$\mathcal{D}(\theta_0 \mid D_R) = -2 \ln(\mathcal{R}(\theta_0 \mid D_R)) = 2 \sum_{i=1}^c m_i \ln \left( \frac{m_i}{m \theta_{0i}} \right).$$

## Excursion: Likelihood Ratio Statistic / G Statistic

- Note that the products  $e_i = m\theta_{0i}$ ,  $i = 1, \dots, c$ , are the **expected frequencies** of the classes  $c_i$  under the hypothesis  $H: \theta = \theta_0$ .

- Hence we can write the **likelihood ratio statistic / G statistic** as

$$\mathcal{D}(\theta_0 \mid D_R) = -2 \ln(\mathcal{R}(\theta_0 \mid D_R)) = 2 \sum_{i=1}^c m_i \ln \left( \frac{m_i}{e_i} \right),$$

which relates the **observed frequencies**  $m_i$  to the **expected frequencies**  $e_i$ .

- If we write the test statistic in terms of the class frequencies  $n_i$  in the whole data set and  $m_i$  in the subset  $D_R$  covered by rule  $R$ , we obtain

$$\mathcal{D}(\theta_0 \mid D_R) = 2 \sum_{i=1}^c m_i \ln \left( \frac{nm_i}{mn_i} \right) = 2 \sum_{i=1}^c m_i \ln \left( \frac{m_i}{n_i} \bigg/ \frac{m}{n} \right).$$

- If the set  $D_R$  covered by rule  $R$  is similar to a random sample of size  $m$ , the fractions  $\frac{m_i}{n_i}$  of examples covered by rule  $R$  in each class  $c_i$ ,  $i = 1, \dots, c$ , should be close to the fraction  $\frac{m}{n}$  of the whole data set that is covered by  $R$ . In this case  $\mathcal{D}(\theta_0 \mid D_R)$  will be small and the rule  $R$  not significant.

## Excursion: Likelihood Ratio Statistic / G Statistic

- The **likelihood ratio statistic / G statistic** is

$$\mathcal{D}(\theta_0 \mid D_R) = -2 \ln(\mathcal{R}(\theta_0 \mid D_R)) = 2 \sum_{i=1}^c m_i \ln \left( \frac{m_i}{n_i} \middle/ \frac{m}{n} \right).$$

- According to **Wilks' Theorem** [Wilks 1938], this test statistic is approximately  $\chi^2$  distributed with  $r - r_0 = (c - 1) - 0 = c - 1$  degrees of freedom.
- Hence we can compute a **p-value** by evaluating the **survival function** (that is, 1 minus the **cumulative distribution function**, cdf) of the  $\chi^2$  distribution with  $c-1$  degrees of freedom at  $\mathcal{D}(\theta_0 \mid D_R)$ :

$$p = S_{\chi_{c-1}^2}(\mathcal{D}(\theta_0 \mid D_R)) = 1 - F_{\chi_{c-1}^2}(\mathcal{D}(\theta_0 \mid D_R)).$$

- The **test** is conducted by comparing  $p$  to a chosen **significance level**  $\alpha$ .
  - If  $p \leq \alpha$ , then the class distribution of the examples covered by rule  $R$  differs significantly from the class distribution in the whole data set.
  - If  $p > \alpha$ , then the class distribution of the examples covered by rule  $R$  does not differ significantly from the class distribution in the whole data set.

## Alternative: Pearson's $\chi^2$ Statistic

- An alternative to the G statistic is **Pearson's  $\chi^2$  statistic**: [Pearson 1900]

$$\chi^2 = \sum_{i=1}^c \frac{(m_i - e_i)^2}{e_i} = \sum_{i=1}^c \frac{(m_i - \frac{m}{n}n_i)^2}{\frac{m}{n}n_i} = \frac{1}{n} \sum_{i=1}^c \frac{(nm_i - mn_i)^2}{mn_i},$$

where the  $e_i$  are the **expected frequencies** under the null hypothesis and the  $m_i$  are the **observed frequencies**.

Reminder: The null hypothesis is that the  $m$  examples covered by the rule  $R$  are a random sample from all  $n$  examples.

- The  $\chi^2$  statistic is approximately  $\chi^2$  distributed with  $c - 1$  degrees of freedom (in large samples and under suitable regularity conditions).
- It is usually recommended to **prefer the G statistic over Pearson's  $\chi^2$  statistic**, because the  $\chi^2$  approximation is generally better for the G statistic.
- However, in practice Pearson's  $\chi^2$  statistic is more popular, possibly because the theory of likelihood ratio statistics (in particular the special case of the G statistic) is less well known.

# CN2 Algorithm

```
function CN2 ( $D, k, \alpha$ ) : set of rules      (*  $k$ : beam size,  $\alpha$ : significance level *)
   $\mathcal{R} \leftarrow []$                         (* initialize an empty list of rules *)
  while  $D \neq \emptyset$  do                  (* while not all examples are covered *)
     $s_{\text{best}} \leftarrow \text{best}(D, k, \alpha)$       (* find the best conjunction/antecedent *)
     $D_{\text{cov}} \leftarrow \{d \in D \mid s_{\text{best}} \triangleleft d\}$   (* get majority class of covered examples *)
     $c_{\text{maj}} \leftarrow \text{argmax}_{c \in \text{dom}(\text{class})} |\{d \in D_{\text{cov}} \mid (\text{class} = c) \triangleleft d\}|$ 
     $\mathcal{R} \leftarrow \mathcal{R} \sqcup [s_{\text{best}} \rightarrow (\text{class} = c_{\text{maj}})]$   (* append found rule to rule list *)
     $D \leftarrow \{d \in D \mid s_{\text{best}} \not\triangleleft d\}$   (* remove covered examples *)
  return  $\mathcal{R}$                              (* return the found rules *)
```

- The function “best” may return an empty conjunction, i.e. “true”.  
This generates a “default” rule, covering all remaining examples.
- The symbol “ $\sqcup$ ” is used here to denote a concatenation of lists.
- Since all covered examples are removed, irrespective of their class, the order of the rules in the returned list is relevant (“decision list”), and hence we cannot use a simple *set* of rules.

# CN2 Algorithm

```
function best ( $D, k, \alpha$ ) : conjunction      (*  $k$ : beam size,  $\alpha$ : significance level *)
   $s_{\text{best}} \leftarrow \text{true}$                   (* default: empty conjunction is best *)
   $S \leftarrow \{s_{\text{best}}\}$                   (* initialize a set of conjunctions *)
   $H_{\text{best}} \leftarrow Q_1(s_{\text{best}}, D)$       (* class entropy of all examples *)
   $p_{\text{best}} \leftarrow 1$                     (* significance of empty conjunction *)
  while  $S \neq \emptyset$  do                  (* while conjunctions to specialize *)
     $S \leftarrow \{s \wedge c \mid c \in C_{\text{all}} \wedge (s \wedge c \neq \text{false})\} - S$  (* specialize all conjunctions *)
    for  $s \in S$  do                          (* traverse specialized conjunctions *)
       $D_s \leftarrow \{d \in D \mid s \triangleleft d\}$  (* get examples covered by  $s$  *)
       $H_s \leftarrow Q_1(s, D_s)$               (* class entropy of covered examples *)
       $p_s \leftarrow Q_2(s, D_s, D)$           (*  $p$ -value of conjunction  $s$  *)
      if  $p_s \leq \alpha \wedge (H_s < H_{\text{best}} \vee (H_s = H_{\text{best}} \wedge p_s < p_{\text{best}}))$  then
         $s_{\text{best}} \leftarrow s$ ;  $H_{\text{best}} \leftarrow H_s$ ;  $p_{\text{best}} \leftarrow p_s$ 
      while  $|S| > k$  do                      (* while more conj. than beam size *)
         $S \leftarrow S - \{\text{argmax}_{s \in S} Q_1(s, D_s)\}$  (* remove worst conjunction in  $S$  *)
  return  $s_{\text{best}}$                           (* return found best conjunction *)
```

- When specializing  $S$ , redundant conditions may have to be removed as well.

# CN2 Algorithm: Example

- **Zoo Data Set**

(from UCI Machine Learning Repository)

- 101 example cases (each example case describes an animal)
- 15 Boolean attributes:  
hair, feathers, eggs, milk, airborne, aquatic, predator, toothed, backbone,  
breathes, venomous, fins, tail, domestic, catsize
- 1 numeric attribute:  
legs (with values 0, 2, 4, 5, 6, 8) (5 legs: starfish)
- 7 classes: amphibian, bird, fish, insect, invertebrate, mammal, reptile

- **Get all possible conditions** that may be used in rule antecedents:

- Boolean attributes: “ $\langle \text{attribute} \rangle = \text{true}$ ” and “ $\langle \text{attribute} \rangle = \text{false}$ ”
- Numeric attribute:

“legs $\leq 1$ ”	“legs $\leq 5.5$ ”	“legs $> 1$ ”	“legs $> 5.5$ ”
“legs $\leq 3$ ”	“legs $\leq 7$ ”	“legs $> 3$ ”	“legs $> 7$ ”
“legs $\leq 4.5$ ”			“legs $> 4.5$ ”



## CN2 Algorithm: Example

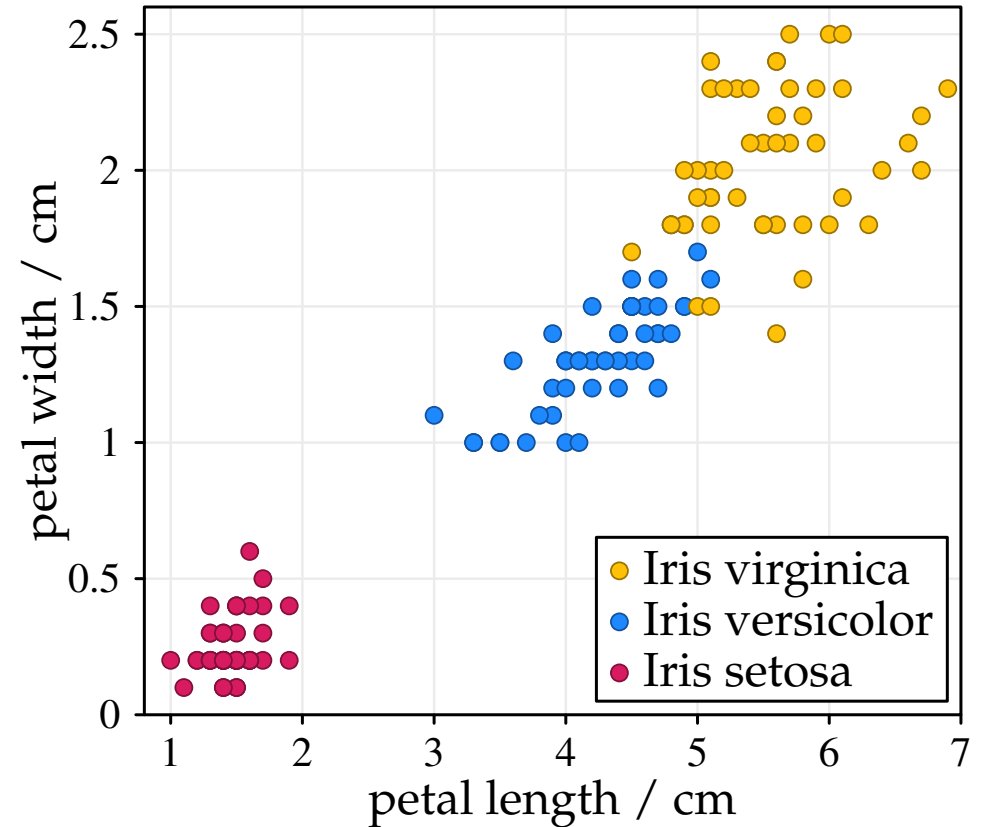
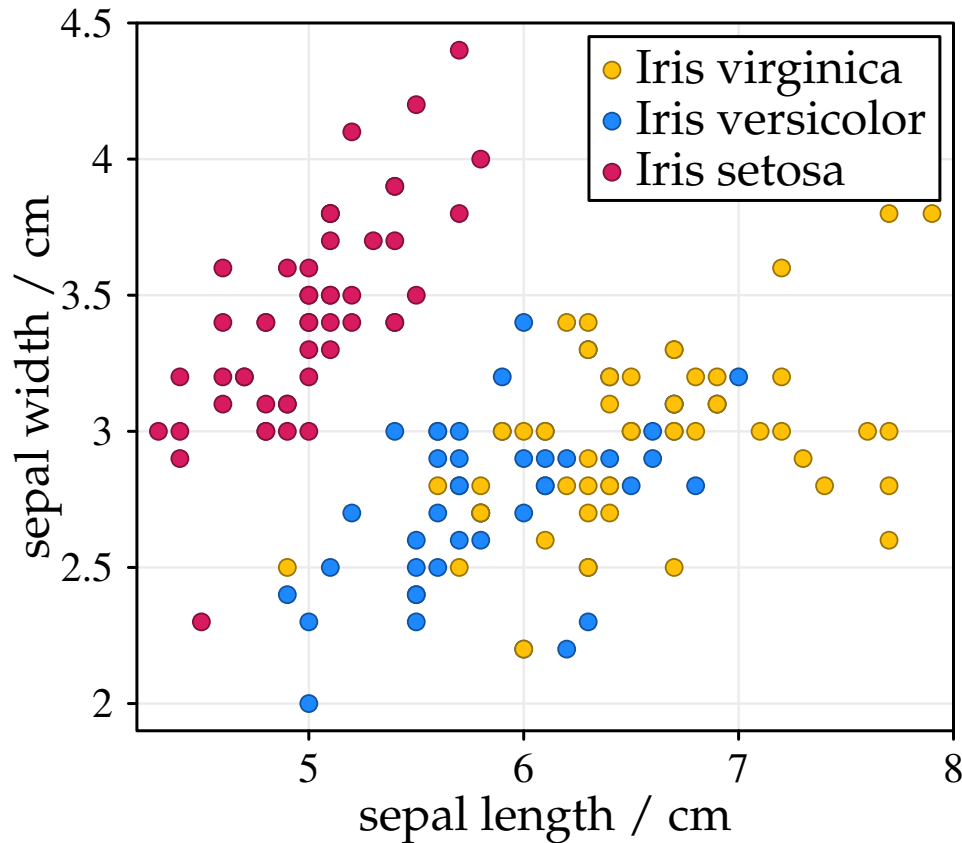
- Rules generated by the CN2 algorithm (parameters:  $k = 5$ ,  $\alpha = 0.1$ ):
  - If milk = true, then class = mammal. mammal:41
  - If feathers = true, then class = bird. bird:20
  - If fins = true, then class = fish. fish:13
  - If airborne = true, then class = insect. insect:6
  - If predator = true and backbone = false, then class = invertebrate. invertebrate:8
  - If legs > 5.5, then class = insect. insect:2
  - If backbone = false, then class = invertebrate. invertebrate:2
  - If tail = false, then class = amphibian. amphibian:3
  - If true, then class = reptile. amphibian:1, reptile:5
- Only one error on the training data. An additional rule to achieve perfect classification does not pass the rule significance test.

# 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.

# 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).

## CN2 Algorithm: Example

- Rules generated by the CN2 algorithm (parameters:  $k = 5$ ,  $\alpha = 0.1$ ):
  - If petal width  $\leq 0.8$ , then iris type = Iris setosa. Iris setosa:50
  - If petal width  $> 1.85$ , then iris type = Iris virginica. Iris virginica:34
  - If petal length  $> 5.35$ , then iris type = Iris virginica. Iris virginica:8
  - If petal width  $\leq 1.45$ , then iris type = Iris versicolor. Iris versicolor:35
  - If sepal width  $> 3.05$ , then iris type = Iris versicolor. Iris versicolor:6
  - If petal width  $> 1.75$ , then iris type = Iris virginica. Iris virginica:5
  - If sepal width  $> 2.85$ , then iris type = Iris versicolor. Iris versicolor:5
  - If true, then iris type = Iris versicolor. Iris versicolor:4 Iris virginica:3
- Only three errors on the training data. Additional rules to achieve perfect classification does not pass the rule significance test.

# CN2 Algorithm: Discussion

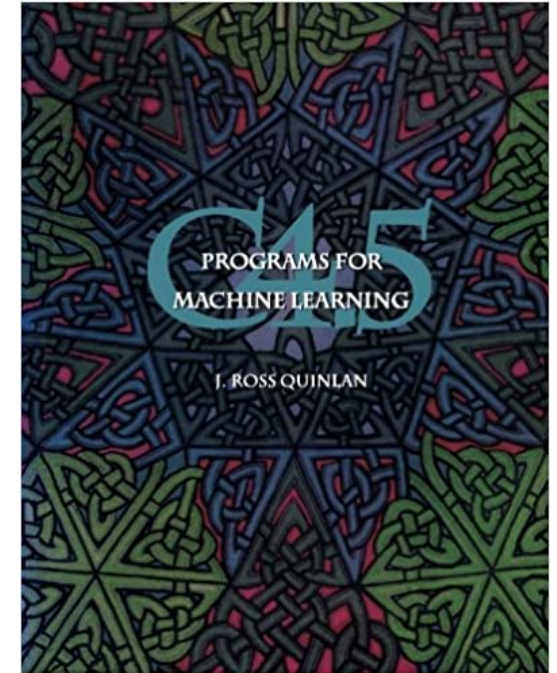
- The CN2 algorithm **allows for misclassifications** in the training data, that is, it **does not assume noise-free data**.
- Since there is **no target class** (rule consequents chosen as majority class), and **no seed examples** (positive and negative) are chosen during the search, the set of conditions used for specializations is (usually) less restricted.
- As a consequence, the CN2 algorithm is **usually slower** than the  $A^q$  (because more combinations of conditions / rule antecedents are checked, even if the same beam size  $k$  is used).
- It is usually also slower than a decision tree based approach, because decision tree induction relies on a greedy test selection.
- Clark & Niblett claim: “Indeed, with a beam width of one the CN2 algorithm behaves equivalently to ID3 [precursor of C4.5] growing a single tree branch.”

This is incorrect, since ID3 computes information gain to assess tests, not just the entropy for a single outcome of a potential test, and hence may choose a different test (attribute) than CN2.

# Summary Rule Learning

- **Rule Extraction from Decision Trees**

- Each path from the root to a leaf yields a rule.
- Rule conditions are (greedily) simplified.  
(errors increased to upper bound of confidence interval)
- Rules predicting the same class are grouped.
- Class rule subsets are simplified and ordered.
- More on decision trees and rule extraction  $\Rightarrow$



- **Direct Rule Induction**

- Successive specialization of a rule with an initially empty antecedent.
- Usually employs beam search to limit the search complexity, since the number of rules grows exponentially with the number of attributes.
- $A^q$  algorithm: assumes noise-free data, induces rules for a target class.
- CN2 algorithm: allows for noise, chooses class from covered examples.

# Support Vector Machines

# Overview Support Vector Machines

- **Linear Regression**

- Ordinary least squares (primal and dual form)
- Ridge regression / Tikhonov regularization (primal and dual form)

- **Linear Classification**

- Geometric interpretation and linear separability
- Maximum margin separation and support vectors
- Slack variables to allow for some misclassifications
- Limitations of linear separability

- **Non-Linear Regression and Non-Linear Classification**

- Idea: Map data to a different space to make it linear(ly separable)
- Kernel functions as means for implicit mapping ("*kernel trick*")
- Some commonly used kernel functions

- **Summary**



# Reminder: Regression

Regression is also known as the **Method of Least Squares**. [Carl Friedrich Gauß]  
(or more specifically Ordinary Least Squares, abbreviated OLS)

Given:

- A data set  $((\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n))$  of  $n$  data tuples (one or more input values and one output value) and
- a hypothesis about the functional relationship between response and predictor values, e.g.  $Y = f(X) = a + bX + \varepsilon$  (where  $\varepsilon$  is a noise term).

Desired:

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

Depending on

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

different types of regression are distinguished.

# Reminder: Function Optimization

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

**Often feasible approach:**

- A necessary condition for a (local) optimum (minimum/maximum) is that the partial derivatives w.r.t. the parameters vanish [Pierre de Fermat, 1607–1665].
- Therefore: (Try to) Solve the equation system that results from setting all partial derivatives w.r.t. the parameters equal to zero.

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

**Solution procedure:**

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

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

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

# Reminder: Univariate Linear Regression

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

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

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

Necessary conditions for a minimum

(a.k.a. Fermat's theorem, after Pierre de Fermat, 1607–1665):

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

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

(Note: Not to be confused with Fermat's *Last* Theorem!)

# Reminder: Univariate Linear Regression

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

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

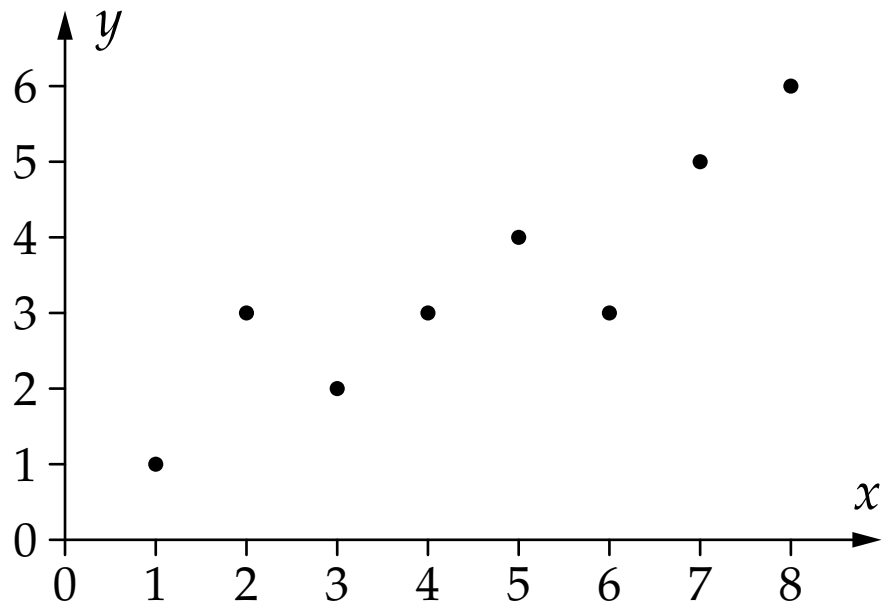
- Two linear equations for two unknowns  $a$  and  $b$ .
- System can be solved with standard methods from linear algebra.
- Solution is unique unless all  $x$ -values are identical.
  - If all  $x$ -values are equal, it is  $x_i = x_0, i = 1, \dots, n$ .
  - The first equation simplifies to  $na + nx_0b = \sum_{i=1}^n y_i$ , the second to  $nx_0a + nx_0^2b = x_0 \sum_{i=1}^n y_i$ . These two equations are linearly dependent, hence there is no unique solution.
- The resulting line is called a **regression line**.

# Univariate Linear Regression: Example

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

Assumption:

$$y = a + bx$$

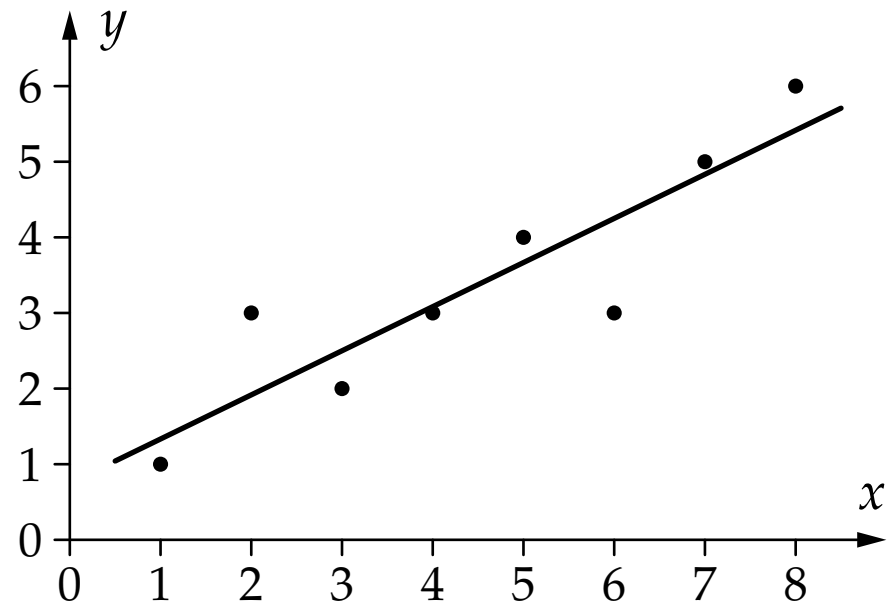


Normal equations:

$$\begin{aligned} 8a + 36b &= 27, \\ 36a + 204b &= 146. \end{aligned}$$

Solution:

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



# Reminder: Bivariate Linear Regression

**Generalization to more than one argument / regressor:**

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

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

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

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

$$\frac{\partial F}{\partial a} = \sum_{i=1}^n 2(a + bx_i + cy_i - z_i) = 0,$$

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

$$\frac{\partial F}{\partial c} = \sum_{i=1}^n 2(a + bx_i + cy_i - z_i)y_i = 0.$$

# Reminder: Bivariate Linear Regression

**System of normal equations for two arguments / regressors:**

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

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

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

- 3 linear equations for 3 unknowns  $a$ ,  $b$ , and  $c$ .
- System can be solved with standard methods from linear algebra.
- Solution is unique unless all data points  $(x_i, y_i)$  lie on a straight line.
  - Assuming  $y = px + q$ , one can show that the equations are linearly dependent.

# Reminder: Multivariate Linear Regression

**General multivariate linear case:**

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

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

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

where (leading 1s in  $\mathbf{X}^\circ$  capture the constant  $a_0$  via a pseudo-argument  $x_0 \equiv 1$ )

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

$\underbrace{\hspace{10em}}_{= \mathbf{X}}$

Necessary conditions for a minimum: ( $\nabla$  is a differential operator called “nabla” or “del”.)

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



# Reminder: Multivariate Linear Regression

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

$$\nabla_{\vec{a}^\circ} = \left( \frac{\partial}{\partial a_0}, \dots, \frac{\partial}{\partial a_m} \right)$$

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

- Alternatively, one may write out the differentiation component-wise.

With the former method we obtain for the derivative:

$$\begin{aligned} \nabla_{\vec{a}^\circ} F(\vec{a}^\circ) &= \nabla_{\vec{a}^\circ} \left( (\mathbf{X}^\circ \vec{a}^\circ - \vec{y})^\top (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}) \right) \\ &= (\nabla_{\vec{a}^\circ} (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}))^\top (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}) + ((\mathbf{X}^\circ \vec{a}^\circ - \vec{y})^\top (\nabla_{\vec{a}^\circ} (\mathbf{X}^\circ \vec{a}^\circ - \vec{y})))^\top \\ &= (\nabla_{\vec{a}^\circ} (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}))^\top (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}) + (\nabla_{\vec{a}^\circ} (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}))^\top (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}) \\ &= 2\mathbf{X}^{\circ\top} (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}) \\ &= 2\mathbf{X}^{\circ\top} \mathbf{X}^\circ \vec{a}^\circ - 2\mathbf{X}^{\circ\top} \vec{y} = \vec{0} \end{aligned}$$

# Reminder: Multivariate Linear Regression

Necessary conditions for a minimum are therefore:

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

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

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

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

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

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

The **prediction function** (i.e. the function to compute  $y$  from given  $\vec{x}$ ) is

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

where  $\vec{x}^\circ = \begin{bmatrix} 1 \\ \vec{x} \end{bmatrix} = (1, x_1, \dots, x_m)^\top$  is the extended argument vector.

# Multivariate Linear Regression: Primal and Dual Form

- Writing the solution vector as  $\vec{a}^\circ = (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-1} \mathbf{X}^{\circ\top} \vec{y}$  is called the **primal form** of the solution.
- To obtain the **dual form** of the solution, we rewrite the primal form as

$$\begin{aligned}\vec{a}^\circ &= (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-1} \mathbf{X}^{\circ\top} \vec{y} \\ &= \mathbf{I}_{m+1} (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-1} \mathbf{X}^{\circ\top} \vec{y} && (\mathbf{I}_{m+1}: (m+1) \times (m+1) \text{ unit matrix}) \\ &= \overbrace{\mathbf{X}^{\circ\top} \mathbf{X}^\circ (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-1}} (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-1} \mathbf{X}^{\circ\top} \vec{y} \\ &= \mathbf{X}^{\circ\top} \mathbf{X}^\circ (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-2} \mathbf{X}^{\circ\top} \vec{y} \\ &= \mathbf{X}^{\circ\top} \vec{b} \quad \text{where } \vec{b} = \mathbf{X}^\circ (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-2} \mathbf{X}^{\circ\top} \vec{y}.\end{aligned}$$

- Writing this out as a sum over the extended data points yields

$$\vec{a}^\circ = \sum_{i=1}^n b_i \vec{x}_i^\circ, \quad \text{where} \quad \vec{x}_i^\circ = \begin{bmatrix} 1 \\ \vec{x} \end{bmatrix} = (1, x_{i1}, \dots, x_{im})^\top.$$

That is, the solution  $\vec{a}^\circ$  can be written as a **weighted sum of the data points**.

# Multivariate Linear Regression: Primal and Dual Form

- Writing the solution vector as  $\vec{a}^\circ = (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-1} \mathbf{X}^{\circ\top} \vec{y}$  is called the **primal form** of the solution.
- Writing the solution vector as  $\vec{a}^\circ = \mathbf{X}^{\circ\top} \vec{b}$  with  $\vec{b} = \mathbf{X}^\circ (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-2} \mathbf{X}^{\circ\top} \vec{y}$  is called the **dual form** of the solution.

- The **prediction function** (i.e. the function to compute  $y$  from given  $\vec{x}$ ) resulting from this dual form is

$$y = f(\vec{x}) = \vec{a}^{\circ\top} \vec{x}^\circ = (\mathbf{X}^{\circ\top} \vec{b})^\top \vec{x}^\circ = \vec{b}^\top \mathbf{X}^\circ \vec{x}^\circ = \vec{b}^\top \vec{z}^\circ,$$

where  $\vec{x}^\circ = \begin{bmatrix} 1 \\ \vec{x} \end{bmatrix} = (1, x_1, \dots, x_m)^\top$  is the extended argument vector and  $z_i^\circ = \vec{x}_i^{\circ\top} \vec{x}^\circ, i = 1, \dots, n$ , are its scalar products with the extended data points.

- Note that this prediction function does not refer to the **primal parameters**  $\vec{a}^\circ$ , but rather to the **dual parameters**  $\vec{b}$  and a vector of scalar products.
- The concept of **reparameterizing the prediction function** is central to the theory of support vector machines.

## Excursion: Properties of the Matrix $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$

The matrix  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is square, symmetric, and always positive semi-definite.

- A  $q \times q$  matrix  $\mathbf{M}$  is called  
positive semi-definite (or non-negative definite):  $\forall \vec{v} \in \mathbb{R}^q : \vec{v}^{\top} \mathbf{M} \vec{v} \geq 0$ ,  
negative semi-definite (or non-positive definite):  $\forall \vec{v} \in \mathbb{R}^q : \vec{v}^{\top} \mathbf{M} \vec{v} \leq 0$ ,
- For any  $\vec{x} \in \mathbb{R}^q$  the outer product  $\vec{x} \vec{x}^{\top}$  yields a positive semi-definite matrix:

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

- 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.

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

- The matrix  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is positive semi-definite as it can be written as

$$\mathbf{X}^{\circ\top} \mathbf{X}^{\circ} = \sum_{i=1}^n \vec{x}_i^{\circ} \vec{x}_i^{\circ\top} \quad \text{where} \quad \vec{x}_i^{\circ} = \begin{bmatrix} 1 \\ \vec{x} \end{bmatrix} = (1, x_{i1}, \dots, x_{im})^{\top}.$$

## Excursion: Properties of the Matrix $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$

The matrix  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is generally positive definite,  
unless the rows of  $\mathbf{X}^{\circ}$  are linearly dependent. (rows of  $\mathbf{X}^{\circ}$ : extended data points)

- positive definite:  $\forall \vec{v} \in \mathbb{R}^q - \{\vec{0}\} : \vec{v}^{\top} \mathbf{M} \vec{v} > 0,$   
negative definite:  $\forall \vec{v} \in \mathbb{R}^q - \{\vec{0}\} : \vec{v}^{\top} \mathbf{M} \vec{v} < 0,$
- The matrix  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  can be written as  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ} = \sum_{i=1}^n \vec{x}_i^{\circ} \vec{x}_i^{\circ\top}.$
- Suppose that  $\exists \vec{v} \in \mathbb{R}^{m+1} - \{\vec{0}\} : \forall i; 1 \leq i \leq n : \vec{v}^{\top} \vec{x}_i^{\circ} = 0.$   
(implying  $\vec{v}^{\top} \vec{x}_i^{\circ} \vec{x}_i^{\circ\top} \vec{v} = (\vec{v}^{\top} \vec{x}_i^{\circ})^2 = 0$ )

Furthermore, suppose that the set  $\{\vec{x}_1^{\circ}, \dots, \vec{x}_n^{\circ}\}$  spans  $\mathbb{R}^{m+1}.$

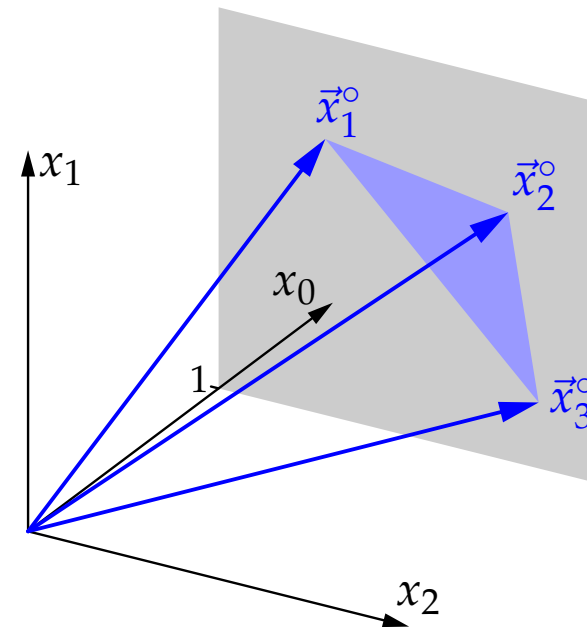
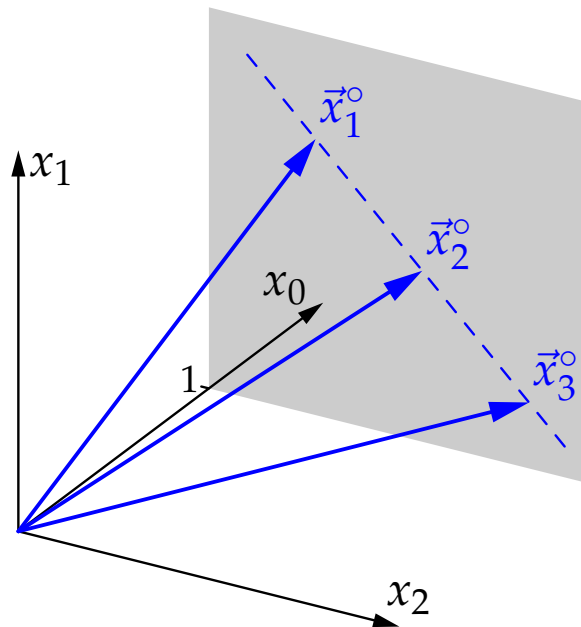
Then there exist  $\alpha_1, \dots, \alpha_n \in \mathbb{R}$  such that  $\vec{v} = \alpha_1 \vec{x}_1^{\circ} + \dots + \alpha_n \vec{x}_n^{\circ}.$

Hence  $\vec{v}^{\top} \vec{v} = \underbrace{\vec{v}^{\top} \vec{x}_1^{\circ}}_{=0} \alpha_1 + \dots + \underbrace{\vec{v}^{\top} \vec{x}_n^{\circ}}_{=0} \alpha_n = 0,$  implying  $\vec{v} = \vec{0},$  contradicting  $\vec{v} \neq \vec{0}.$   
(by assumption)

- Therefore, if the  $\vec{x}_i^{\circ}, i = 1, \dots, n,$  span  $\mathbb{R}^{m+1},$  then  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is positive definite.  
Only if the  $\vec{x}_i^{\circ}, i = 1, \dots, n,$  do not span  $\mathbb{R}^{m+1},$  that is, if the data points  $\vec{x}_i$  lie in a lower-dimensional (linear) subspace,  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is only positive semi-definite.

## Excursion: Properties of the Matrix $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$

- For the matrix  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  to be positive definite, only the data points  $\vec{x}_i$  must not lie in a lower-dimensional (linear) subspace.
- The extended data points  $\vec{x}_i^{\circ}$  always lie in a lower-dimensional subspace, simply because it is  $x_{i0} = 1$  for  $i = 1, \dots, n$ .
- However, as (location) vectors from the origin, the  $\vec{x}_i^{\circ}$  still span  $\mathbb{R}^{m+1}$ , unless the data points  $\vec{x}_i$  lie in a lower-dimensional (linear) subspace.



## Excursion: Properties of the Matrix $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$

- A square, symmetric, positive definite matrix is regular, that is, invertible. Thus, the matrix  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is invertible unless the  $\vec{x}_i, i = 1, \dots, n$ , do not span  $\mathbb{R}^m$ .
- However, even if  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is invertible, computing the inverse can be numerically unstable due to the finite precision of computer arithmetic. This is particularly the case if the matrix to invert is **ill-conditioned**.
- Generally, the **condition number** of a function measures how much its value can change for a small change of its argument.
- Solving the linear equation system  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ} \vec{a}^{\circ} = \mathbf{X}^{\circ\top} \vec{y}$  (a.k.a. normal equations) can also be seen as evaluating a function  $\vec{a}^{\circ} = f(\mathbf{X}^{\circ}, \vec{y})$ . Its stability is captured by the **condition number** of the matrix  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$ .
- If  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is positive definite, it allows for an eigenvalue decomposition where all eigenvalues  $\lambda_i, i = 1, \dots, m + 1$ , are real and positive. Then the condition number of  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is  $\lambda_{\max} / \lambda_{\min}$ , where  $\lambda_{\max}$  and  $\lambda_{\min}$  are the largest and smallest eigenvalue, respectively, of  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$ .



# Ridge Regression / Tikhonov Regularization

- Suppose that  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is either **singular** or **ill-conditioned**. (large condition number)  
Solution: Extend the functional to optimize by a **regularization** term, e.g.,

$$F_{\lambda}(\vec{a}^{\circ}) = (\mathbf{X}^{\circ} \vec{a}^{\circ} - \vec{y})^{\top} (\mathbf{X}^{\circ} \vec{a}^{\circ} - \vec{y}) + \lambda \vec{a}^{\circ\top} \vec{a}^{\circ}$$

where  $\lambda > 0$  is a small real-valued number.

- This particular form is known as **ridge regression** or **Tikhonov regularization**.  
The minimum of  $F_{\lambda}(\vec{a}^{\circ})$  can be computed as [Andrey Tikhonov 1943]

$$\vec{a}^{\circ} = (\mathbf{X}^{\circ\top} \mathbf{X}^{\circ} + \lambda \mathbf{I}_{m+1})^{-1} \mathbf{X}^{\circ\top} \vec{y}. \quad (\mathbf{I}_q \text{ denotes a } q \times q \text{ unit matrix})$$

- Note that with  $\lambda = 0$ , ridge regression reduces to ordinary least squares.
- The matrix  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ} + \lambda \mathbf{I}_{m+1}$  is always regular (invertible) for  $\lambda > 0$ ,  
hence a solution always exists. (at least in principle, barring numerical problems)
- Adding  $\lambda \mathbf{I}_{m+1}$  to  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  “shifts (i.e. increases) the eigenvalues” of  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  and thus improves (i.e. lowers) the condition number of  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  (provided  $\lambda > 0$ ); hence this is often done for convenience, even if  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  is not ill-conditioned.

# Ridge Regression: Primal and Dual Form

- Writing the solution vector as  $\vec{a}^\circ = (\mathbf{X}^{\circ\top} \mathbf{X}^\circ + \lambda \mathbf{I}_{m+1})^{-1} \mathbf{X}^{\circ\top} \vec{y}$  is called the **primal form** of the solution of multivariate linear ridge regression.
- To obtain the **dual form**, we assume that  $\vec{a}^\circ$  can be written as  $\vec{a}^\circ = \mathbf{X}^{\circ\top} \vec{b}$ .  
We may obtain this from the normal equations of ridge regression:

$$\begin{aligned} (\mathbf{X}^{\circ\top} \mathbf{X}^\circ + \lambda \mathbf{I}_{m+1}) \vec{a}^\circ &= \mathbf{X}^{\circ\top} \vec{y} \Leftrightarrow \mathbf{X}^{\circ\top} \mathbf{X}^\circ \vec{a}^\circ + \lambda \vec{a}^\circ = \mathbf{X}^{\circ\top} \vec{y} \\ &\Leftrightarrow \lambda \vec{a}^\circ = \mathbf{X}^{\circ\top} \vec{y} - \mathbf{X}^{\circ\top} \mathbf{X}^\circ \vec{a}^\circ \\ &\Leftrightarrow \vec{a}^\circ = \mathbf{X}^{\circ\top} \underbrace{(\vec{y} - \mathbf{X}^\circ \vec{a}^\circ) \lambda^{-1}}_{=\vec{b}} \end{aligned}$$

- Note, however, that in this form  $\vec{b}$  contains  $\vec{a}^\circ$ . To eliminate  $\vec{a}^\circ$ , we exploit

$$\begin{aligned} \vec{b} &= (\vec{y} - \mathbf{X}^\circ \vec{a}^\circ) \lambda^{-1} \Leftrightarrow \lambda \vec{b} = (\vec{y} - \mathbf{X}^\circ \vec{a}^\circ) \\ &\Leftrightarrow \lambda \vec{b} = (\vec{y} - \mathbf{X}^\circ \mathbf{X}^{\circ\top} \vec{b}) \\ &\Leftrightarrow \mathbf{X}^\circ \mathbf{X}^{\circ\top} \vec{b} + \lambda \vec{b} = \vec{y} \\ &\Leftrightarrow (\mathbf{X}^\circ \mathbf{X}^{\circ\top} + \lambda \mathbf{I}_n) \vec{b} = \vec{y} \\ &\Leftrightarrow \vec{b} = (\mathbf{X}^\circ \mathbf{X}^{\circ\top} + \lambda \mathbf{I}_n)^{-1} \vec{y} \end{aligned}$$

# Ridge Regression: Primal and Dual Form

- Writing the solution vector as  $\vec{a}^\circ = (\mathbf{X}^{\circ\top} \mathbf{X}^\circ + \lambda \mathbf{I}_{m+1})^{-1} \mathbf{X}^{\circ\top} \vec{y}$  is called the **primal form** of the solution of multivariate linear ridge regression.
- Writing the solution vector as  $\vec{a}^\circ = \mathbf{X}^{\circ\top} \vec{b} = \mathbf{X}^{\circ\top} (\mathbf{X}^\circ \mathbf{X}^{\circ\top} + \lambda \mathbf{I}_n)^{-1} \vec{y}$  is called the **dual form** of the solution of multivariate linear ridge regression.
- The **prediction function** (i.e. the function to compute  $y$  from given  $\vec{x}$ ) resulting from this dual form is

$$y = f(\vec{x}) = \vec{a}^{\circ\top} \vec{x}^\circ = (\mathbf{X}^{\circ\top} \vec{b})^\top \vec{x}^\circ = \vec{b}^\top \mathbf{X}^\circ \vec{x}^\circ = \vec{b}^\top \vec{z}^\circ,$$

where  $\vec{x}^\circ = \begin{bmatrix} 1 \\ \vec{x} \end{bmatrix} = (1, x_1, \dots, x_m)^\top$  is the extended argument vector and  $z_i^\circ = \vec{x}_i^{\circ\top} \vec{x}^\circ, i = 1, \dots, n$ , are its scalar products with the extended data points.

- Note that this prediction function does not refer to the **primal parameters**  $\vec{a}^\circ$ , but rather to the **dual parameters**  $\vec{b}$  and a vector of scalar products.
- Note: With the dual parameters  $\vec{b}$ , we no longer need the data points itself.

**We only need means to compute scalar products of data points.**

# Ridge Regression: Primal and Dual Form

- Recalling that the dual parameters are computed as  $\vec{b} = (\mathbf{X}^\circ \mathbf{X}^{\circ\top} + \lambda \mathbf{I}_n)^{-1} \vec{y}$  and realizing that the elements of the matrix  $\mathbf{G}^\circ = \mathbf{X}^\circ \mathbf{X}^{\circ\top}$  are  $G_{ij}^\circ = \vec{x}_i^{\circ\top} \vec{x}_j^\circ$ , we note that we neither need the data points to compute the dual parameters.

**We only need means to compute scalar products of data points.**

- This insight does not seem particularly useful at the moment (after all: how can we compute scalar products without data points?), but it will be decisive for the so-called **kernel trick** discussed later.
- The matrix  $\mathbf{G}^\circ = \mathbf{X}^\circ \mathbf{X}^{\circ\top}$  of pairwise scalar products of the (extd.) data points is known as the **Gram matrix** or **Gramian matrix**. [Jørgen Pedersen Gram]
- In linear algebra, the Gram matrix  $\mathbf{G}$  of a set of (real-valued) vectors  $\vec{v}_1, \dots, \vec{v}_n$  is the matrix of inner products of these vectors, that is,  $G_{ij} = \vec{v}_i^\top \vec{v}_j$ .
- If the vectors  $\vec{v}_i$  are the columns of a matrix  $\mathbf{V}$ , the Gram matrix is  $\mathbf{V}^\top \mathbf{V}$ .  
If the vectors  $\vec{v}_i$  are the rows of a matrix  $\mathbf{V}$ , the Gram matrix is  $\mathbf{V} \mathbf{V}^\top$ .
- In this sense  $\mathbf{X}^{\circ\top} \mathbf{X}^\circ$  is also a Gram matrix, but for a different set of vectors.

# Efficiency Considerations

- Ordinary least squares (primal):  $\vec{a}^\circ = (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-1} \mathbf{X}^{\circ\top} \vec{y}$   
Ordinary least squares (dual):  $\vec{b} = \mathbf{X}^\circ (\mathbf{X}^{\circ\top} \mathbf{X}^\circ)^{-2} \mathbf{X}^{\circ\top} \vec{y}$   
Ridge regression (primal):  $\vec{a}^\circ = (\mathbf{X}^{\circ\top} \mathbf{X}^\circ + \lambda \mathbf{I}_{m+1})^{-1} \mathbf{X}^{\circ\top} \vec{y}$   
Ridge regression (dual):  $\vec{b} = (\mathbf{X}^\circ \mathbf{X}^{\circ\top} + \lambda \mathbf{I}_n)^{-1} \vec{y}$
- The cost of computing the (primal or dual) parameters is (mainly) governed by the cost of the matrix inversion, as this is the most expensive operation.
- Generally, inverting a  $q \times q$  matrix has time complexity  $O(q^3)$ .
- We can exploit that the matrices to invert are symmetric and positive definite.  
This allows us to use **Cholesky decomposition** for the inversion, which is faster than, e.g., Gaussian elimination or LU-decomposition.  
(Although only by a constant factor, so the time complexity is still  $O(q^3)$ .)
- Note that ordinary least squares (primal & dual) and ridge regression (primal) require to invert a  $(m + 1) \times (m + 1)$  matrix ( $m$ : dimensions of the data space), ridge regression (dual) to invert an  $n \times n$  matrix ( $n$ : number of data points).

# Efficiency Considerations

- Whether computing the parameters is cheaper for the primal or the dual form depends on the number  $m$  of dimensions of the data space and the number  $n$  of data points (and their relationship).
- If  $m < n$  (which is usually the case), the primal form is cheaper.  
If  $m > n$  (which requires ridge regression), the dual form is cheaper.
- Computing predictions (primal):  $y = \vec{a}^{\circ\top} \vec{x}^{\circ}$   
Computing predictions (dual):  $y = \vec{b}^{\top} \vec{z}^{\circ} = \vec{b}^{\top} \mathbf{X}^{\circ} \vec{x}^{\circ}$
- Computing the prediction via the scalar product (either  $\vec{a}^{\circ\top} \vec{x}^{\circ}$  or  $\vec{b}^{\top} \vec{z}^{\circ}$ ) has complexity  $O(m)$  for the primal and  $O(n)$  for the dual form.
- Computing predictions with the dual form requires first computing  $\vec{z}^{\circ}$ , the vector of scalar products of  $\vec{x}^{\circ}$  with the data points.  
This computation has complexity  $O(nm)$ .
- Therefore computing predictions with the dual form is always more costly.  
However, we will see later that it has compensating advantages.

# Reminder: Cholesky Decomposition

- Let  $\mathbf{M}$  be a symmetric, positive definite  $q \times q$  matrix (e.g. the matrix  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$ ).
  - symmetric:  $\forall 1 \leq i, j \leq q : \mathbf{M}_{ij} = \mathbf{M}_{ji}$  or  $\mathbf{M}^{\top} = \mathbf{M}$ .
  - positive definite:  $\forall \vec{v} \in \mathbb{R}^q - \{0\} : \vec{v}^{\top} \mathbf{M} \vec{v} > 0$
- Cholesky decomposition serves the purpose to compute a “square root” of  $\mathbf{M}$ , that is, a matrix  $\mathbf{L}$  such that  $\mathbf{L}\mathbf{L}^{\top} = \mathbf{M}$  (that is, the “square” of  $\mathbf{L}$  is  $\mathbf{M}$ ).
- Problem: Generally, the equation  $\mathbf{L}\mathbf{L}^{\top} = \mathbf{M}$  does not have a unique solution. One way to make it unique is to require  $\mathbf{L}$  to be left/lower triangular.
- Hence: Compute a left/lower triangular matrix  $\mathbf{L}$  such that  $\mathbf{L}\mathbf{L}^{\top} = \mathbf{M}$ .

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

# Matrix Inversion from Cholesky Decomposition

- The inverse  $\mathbf{K} = \mathbf{L}^{-1}$  of a left/lower triangular matrix  $\mathbf{L}$  is left/lower triangular.
- It can be computed with a forward substitution with  $\mathbf{L}$ : (find originals of  $\mathbf{I}_{:i}$ )

$$\begin{aligned}\mathbf{K}_{ii} &= \frac{1}{\mathbf{L}_{ii}} \\ \mathbf{K}_{ji} &= -\frac{1}{\mathbf{L}_{jj}} \left( \sum_{k=i}^{j-1} \mathbf{L}_{jk} \mathbf{K}_{ki} \right), \quad j = i+1, i+2, \dots, q.\end{aligned}$$

- Since  $\mathbf{M} = \mathbf{L}\mathbf{L}^\top$  by construction, the inverse  $\mathbf{N} = \mathbf{M}^{-1}$  can be computed as

$$\mathbf{M}^{-1} = (\mathbf{L}\mathbf{L}^\top)^{-1} = \mathbf{L}^{-1}(\mathbf{L}^\top)^{-1} = \mathbf{L}^{-1}(\mathbf{L}^{-1})^\top = \mathbf{K}\mathbf{K}^\top.$$

- This is equivalent to a backward substitution with  $\mathbf{L}^\top$ : (find originals of  $\mathbf{K}_{:i}$ )

$$\mathbf{N}_{ji} = \frac{1}{\mathbf{L}_{jj}} \left( \mathbf{K}_{ji} - \sum_{k=j+1}^q \mathbf{L}_{kj} \mathbf{K}_{ki} \right), \quad j = i, i+1, \dots, q.$$

Since  $\mathbf{M}$  is symmetric,  $\mathbf{N} = \mathbf{M}^{-1}$  must be symmetric. Hence it suffices to compute the left/lower triangle of  $\mathbf{N}$  and fill the right/upper triangle of  $\mathbf{N}$  with its transpose.



# Equation System Solution from Cholesky Decomposition

- Actually we have to solve the equation systems  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ} \vec{a}^{\circ} = \vec{r}^{\circ}$  (normal) or  $(\mathbf{X}^{\circ\top} \mathbf{X}^{\circ} + \lambda \mathbf{I}_i) \vec{a}^{\circ} = \vec{r}^{\circ}$  (regularized) where the right hand side is  $\vec{r}^{\circ} = \mathbf{X}^{\circ\top} \vec{y}$ .
- Given a Cholesky decomposition  $\mathbf{L}$  of  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ}$  or  $\mathbf{X}^{\circ\top} \mathbf{X}^{\circ} + \lambda \mathbf{I}_{m+1}$ , respectively, the result  $\vec{a}^{\circ}$  can be computed by forward / backward substitution directly. (This bypasses the explicit computation of the inverse matrix, which is not really needed.)
- For  $i = 1, \dots, m + 1$  (traversal in ascending order, forward substitution)

$$t_i = \frac{1}{\mathbf{L}_{ii}} \left( r_i - \sum_{k=1}^{i-1} \mathbf{L}_{ik} r_k \right)$$

(Find original of  $\vec{r}$  w.r.t. a mapping with  $\mathbf{L}$ .)

- For  $i = m + 1, \dots, 1$  (traversal in descending order, backward substitution)

$$a_i = \frac{1}{\mathbf{L}_{ii}} \left( t_i - \sum_{k=i+1}^{m+1} \mathbf{L}_{ki} t_k \right)$$

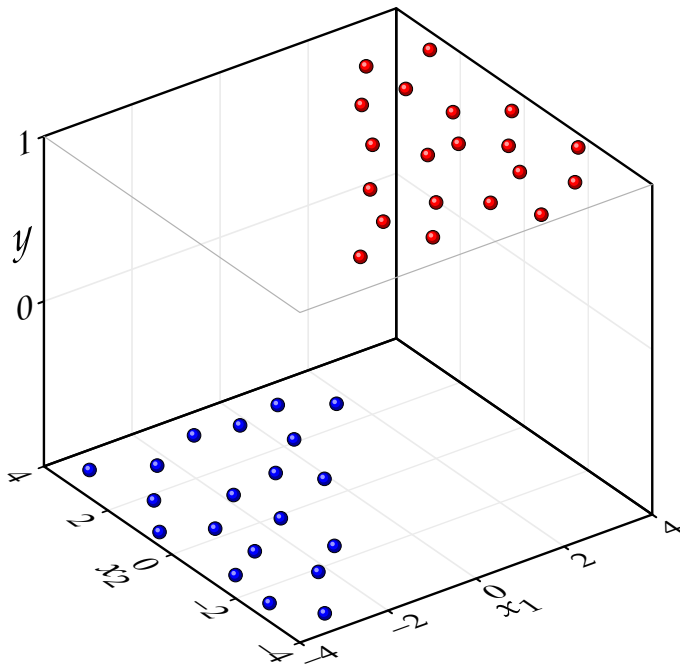
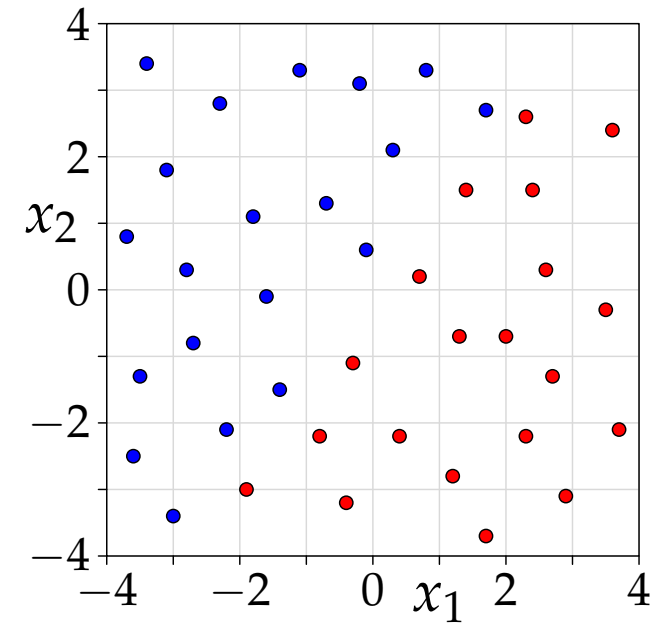
(Find original of  $\vec{t}$  w.r.t. a mapping with  $\mathbf{L}^{\top}$ .)

# Linear Regression: Limitations

- **Linear regression is very popular** in economics, psychology, sociology etc.
- Linear regression has the advantage that it is a fairly robust method.  
A data analysis method is called **robust** or **stable** if its result changes little
  - if the values of the data points (i.e. their coordinates) change slightly or
  - if few data points are added or removed.
- If the attributes / regressors are normalized (usually z-normalized), the coefficients can be interpreted as “importance weights” that quantify the strength of influence the different attributes have.  
(This interpretation is very common, but should be taken with a grain of salt.)
- However, **not all relationships between variables are linear**.
  - At time  $t$  a dropped object has fallen the distance  $s = \frac{1}{2}gt^2$ ,  $g \approx 9.81\text{m/s}^2$ .
  - The decay of a radioactive substance can be described by  $n(t) = n_0e^{-\beta t}$ .
  - The orbit radius  $r$  of a planet at angle  $\theta$  can be described as  $r = \frac{a(1-e^2)}{1+e \cos \theta}$ .

# Linear Classification

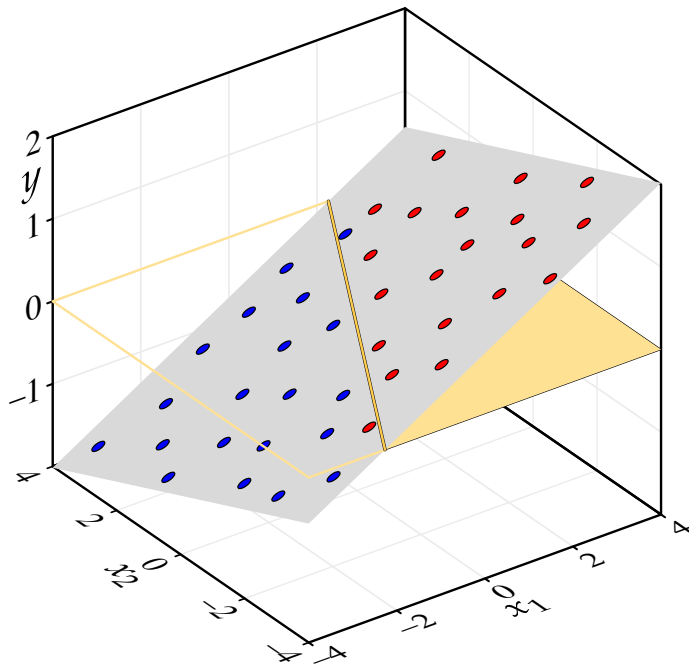
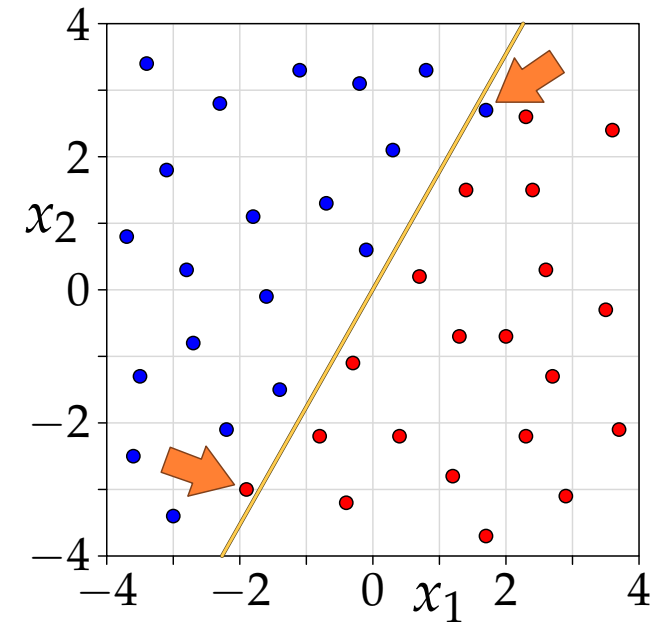
- Up to now: **Regression**  
Now: **Classification**  
(but only two classes)
- Simple data set with two classes:
  - 20 data points belong to the **blue** class.
  - 20 data points belong to the **red** class.



- In principle, two-class classification can be reduced to regression:
- Map one class to  $y = -1$  (here: **blue**), the other class to  $y = +1$  (here: **red**).
- Find a linear regression function  $y = f(\vec{x})$  as described before.

# Linear Classification

- Simple data set with two classes:
  - 20 data points belong to the **blue** class.
  - 20 data points belong to the **red** class.
- Map **blue** class to  $y = -1$ , map **red** class to  $y = +1$ , and compute a bivariate linear regression.



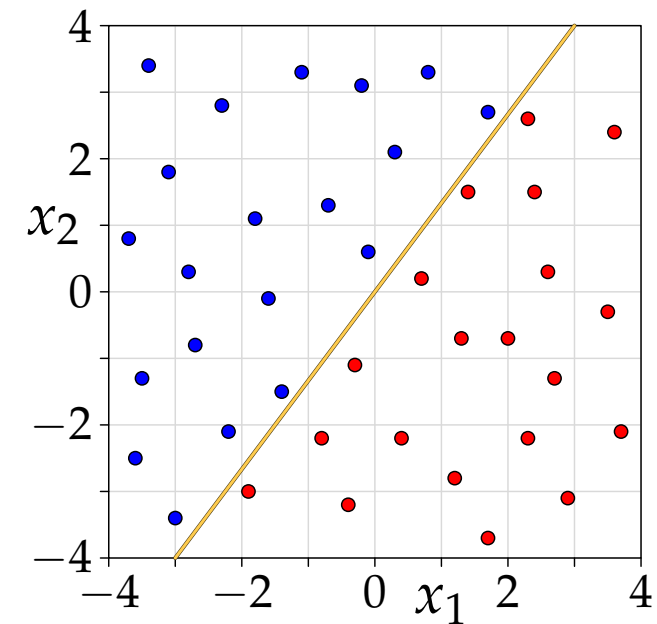
- As a prediction function one may classify the data according to:  
If  $f(\vec{x}) \geq 0$ , assign the **red** class,  
otherwise assign the **blue** class.  
(yellow plane / line)
- But this misclassifies two data points (orange arrows).

# Linear Classification

- Simple data set with two classes:
  - 20 data points belong to the **blue** class,
  - 20 data points belong to the **red** class.

- These classes are **linearly separable**:

There is a straight line such that all points of class **red** are on one side of the line and all points of class **blue** are on the other side.



- This allows for a simple classification rule.
- The separating line (shown in yellow) has the equation

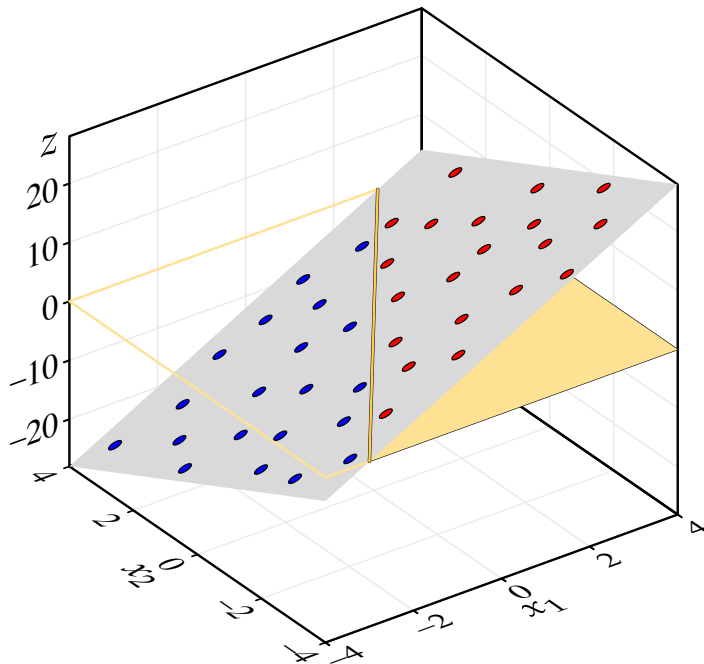
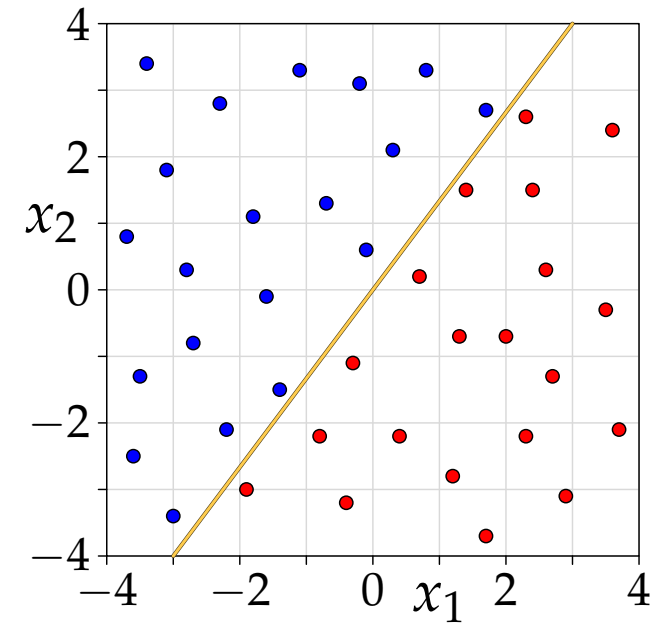
$$g \equiv \begin{pmatrix} 4 \\ -3 \end{pmatrix}^\top \cdot \left( \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right) = 0 \equiv 4x_1 - 3x_2 = 0.$$

- Classify the data according to

If  $4x_1 - 3x_2 \geq 0$ , assign the **red** class, otherwise assign the **blue** class.

# Linear Classification

- Simple data set with two classes:
  - 20 data points belong to the **blue** class,
  - 20 data points belong to the **red** class.
- These classes are **linearly separable**:  
 $4x_1 - 3x_2 > 0$ : class **red**  
 $4x_1 - 3x_2 < 0$ : class **blue**



- In this case the used linear function is
$$z = g(\vec{x}) = 4x_1 - 3x_2.$$
(Any positive multiple may be used as well.)
- With this prediction function all data points are classified correctly. But how does one obtain it?

# Linear Classification: Geometric Interpretation

## Review of line representations

Straight lines are usually represented in one of the following forms:

Explicit Form:  $g \equiv x_2 = bx_1 + c$

Implicit Form:  $g \equiv a_1x_1 + a_2x_2 + d = 0$

Point-Direction Form:  $g \equiv \vec{x} = \vec{p} + k\vec{r}, \quad k \in \mathbb{R}$

Normal Form:  $g \equiv (\vec{x} - \vec{p})^\top \vec{n} = 0$

with the parameters:

$b$  : slope of the line

$c$  : section of the  $x_2$  axis (intercept)

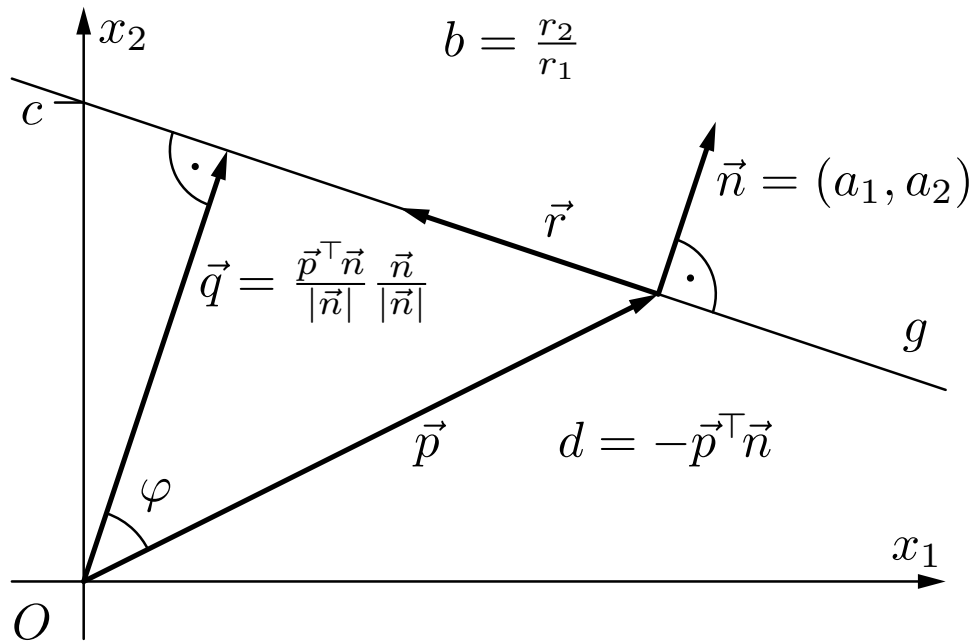
$\vec{p}$  : location vector of a point of the line (base vector)

$\vec{r}$  : direction vector of the line

$\vec{n}$  : normal vector of the line, may be chosen as  $(a_1, a_2)^\top$

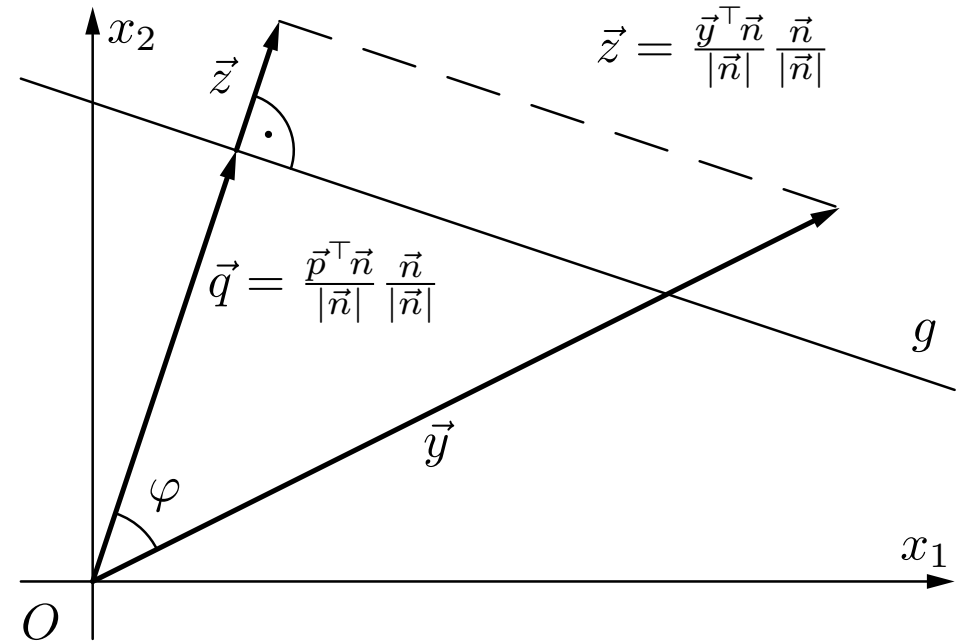
$d$  : “distance” of the line to the origin in units of  $(a_1^2 + a_2^2)^{-\frac{1}{2}}$

# Linear Classification: Geometric Interpretation



## A straight line and its defining parameters:

- $\vec{p}$ : arbitrary point on the line
- $\vec{r}$ : direction vector of the line
- $\vec{n}$ : normal vector of the line
- $\vec{q}$ : plummet from origin to line (projection of  $\vec{p}$  to  $\vec{n}$ )



## How to determine the side on which a point $\vec{y}$ lies:

- $d$ : “length” of  $\vec{q}$  in units of  $\frac{1}{|\vec{n}|}$
- $\vec{z}$ : projection of  $\vec{y}$  to  $\vec{n}$  (to be compared to  $\vec{q}$ )
- If  $\vec{y}^T \vec{n} + d > 0$ , then  $\vec{y}$  lies on the side to which  $\vec{n}$  points.



# Linear Separability

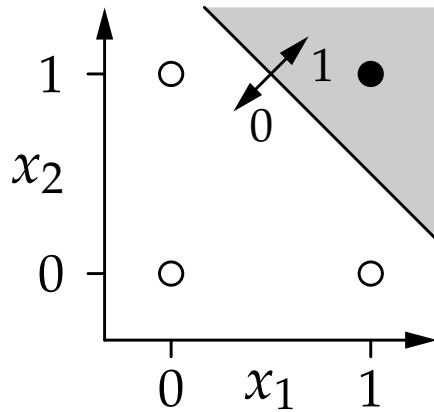
**Definition:** Two sets of points in a Euclidean space are called **linearly separable** iff there exists at least one point, line, plane or hyperplane (depending on the dimension of the space), such that all points of the one set lie on one side and all points of the other set lie on the other side of this point, line, plane or hyperplane (or on it). That is, the point sets can be separated by a **linear decision function**. Formally: Two sets  $X_1, X_2 \subset \mathbb{R}^m$  are linearly separable iff  $\vec{a} \in \mathbb{R}^m$  and  $a_0 \in \mathbb{R}$  exist such that

$$\forall \vec{x} \in X_1 : \quad \vec{a}^\top \vec{x} + a_0 \geq 0 \quad \text{and} \quad \forall \vec{x} \in X_2 : \quad \vec{a}^\top \vec{x} + a_0 < 0.$$

- **Boolean functions** define two points sets, namely the set of points that are mapped to the function value 0 and the set of points that are mapped to 1.  
 $\Rightarrow$  The term “linearly separable” can be transferred to Boolean functions.
- For example, **conjunction**, **disjunction**, and **implication** are **linearly separable** (as are NAND, NOR etc.).
- Only the **exclusive or** (XOR) and the **biimplication** (XNOR) are **not linearly separable** (among binary Boolean functions).

# Linear Separability: Boolean Functions

**The conjunction  $x_1 \wedge x_2$  is linearly separable.**

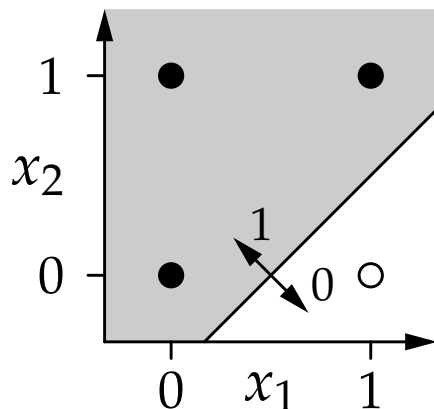


- The separating line has the equation

$$\left( \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} 1 \\ \frac{1}{2} \end{pmatrix} \right)^\top \begin{pmatrix} 2 \\ 2 \end{pmatrix} = 0 \Leftrightarrow 2x_1 + 2x_2 - 3 = 0$$

- The normal vector  $(2, 2)^\top$  points to where the conjunction  $x_1 \wedge x_2$  is 1.

**The implication  $x_1 \rightarrow x_2$  is linearly separable.**



- The separating line has the equation

$$\left( \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} \frac{1}{2} \\ 0 \end{pmatrix} \right)^\top \begin{pmatrix} -2 \\ 2 \end{pmatrix} = 0 \Leftrightarrow 2x_2 - 2x_1 + 1 = 0$$

- The normal vector  $(-2, 2)^\top$  points to where the implication  $x_1 \rightarrow x_2$  is 1.

# Linear Classification: Delta Rule / Perceptron Algorithm

- Suppose a data set with two classes is linearly separable.  
**How can we find a separating line / linear prediction function?**
- We already saw that mapping the problem to linear regression does not work.  
(We saw an example in which the result of linear regression leads to misclassifications.)
- A proper result can be guaranteed by the **Delta rule, perceptron algorithm** or **error correction procedure**. [Widrow and Hoff 1960]
- Here we consider a simplified version: (no “learning rate”  $\eta$  or fixed  $\eta = 1$ )
  - Given: training patterns as  $\mathbf{X}^\circ$  and  $\vec{y}$ . ( $\mathbf{X}^\circ$ :  $n \times (m+1)$  matrix,  $\vec{y}$ :  $n$ -dim. vector)
  - Initialize the parameters  $\vec{a}^\circ = \vec{0}$ . ( $\vec{a}^\circ$ :  $(m+1)$ -dimensional vector)
  - For each training pattern  $(\vec{x}_i^\circ, y_i)$  where  $\vec{x}_i^\circ = \mathbf{X}_{i:}^{\circ\top}$ : ( $\mathbf{X}_{i:}^\circ$ :  $i$ -th row of  $\mathbf{X}^\circ$ )  
if  $y_i \vec{a}^{\circ\top} \vec{x}_i^\circ \leq 0$ , then  $\vec{a}^\circ \leftarrow \vec{a}^\circ + y_i \vec{x}_i^\circ$ . (on error update parameters  $\vec{a}^\circ$ )
  - Repeat until no more errors occur, that is, until  
 $\forall i; 1 \leq i \leq n : y_i \vec{a}^{\circ\top} \vec{x}_i^\circ > 0$ . (all patterns are classified correctly)

# Linear Classification: Delta Rule / Perceptron Algorithm

- The procedure on the preceding slide is the **primal form** of the Delta rule, because it updates the primal parameters  $\vec{a}^\circ$ .
- The **dual form** of the Delta rule updates the dual parameters  $\vec{b}$ :
  - Given: training patterns as  $\mathbf{X}^\circ$  and  $\vec{y}$ . ( $\mathbf{X}^\circ$ :  $n \times (m+1)$  matrix,  $\vec{y}$ :  $n$ -dim. vector)
  - Compute the Gram matrix  $\mathbf{G}^\circ = \mathbf{X}^\circ \mathbf{X}^{\circ\top}$ . ( $\mathbf{G}^\circ$ :  $n \times n$  matrix)
  - Initialize the parameters  $\vec{b} = \vec{0}$ . ( $\vec{b}$ :  $n$ -dimensional vector)
  - For each training pattern  $(\vec{g}_i^\circ, y_i)$  where  $\vec{g}_i^\circ = \mathbf{G}_{:i}^\circ$ : ( $\mathbf{G}_{:i}^\circ$ :  $i$ -th column of  $\mathbf{G}^\circ$ )  
if  $y_i \vec{b}^\top \vec{g}_i^\circ \leq 0$ , then  $b_i \leftarrow b_i + y_i$ . (on error update parameters  $\vec{b}$ )
  - Repeat until no more errors occur, that is, until  
 $\forall i; 1 \leq i \leq n : y_i \vec{b}^\top \vec{g}_i^\circ > 0$ . (all patterns are classified correctly)
- Note that this dual algorithm works only with the Gram matrix  $\mathbf{G}^\circ$ .  
Once the Gram matrix is computed, the data points are no longer needed.  
**Again: We only need means to compute scalar products of data points.**

# Linear Classification: Delta Rule / Perceptron Algorithm

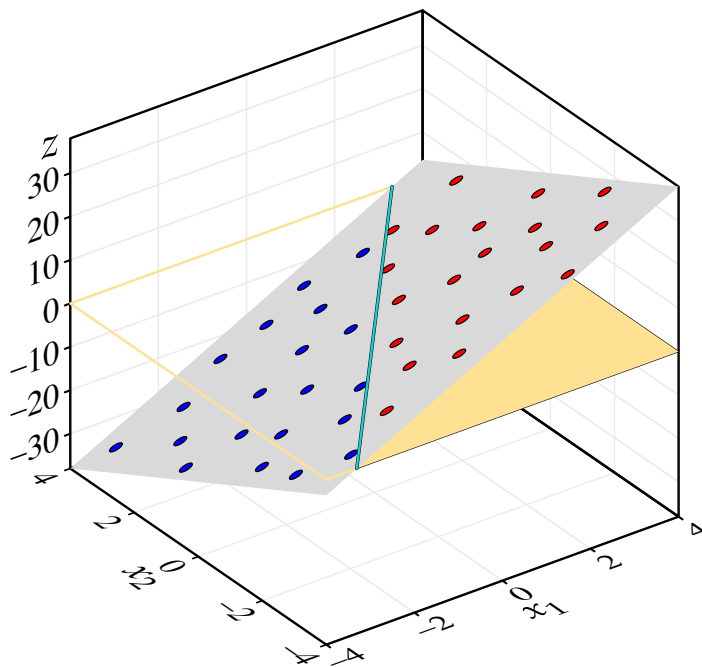
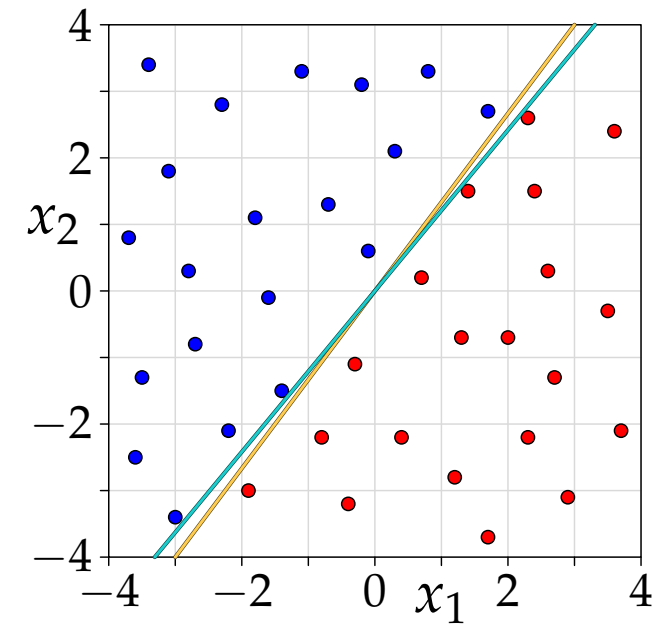
- Note that the obtained result may depend on the **order of the training patterns**.
- But the **primal and the dual form** of the Delta rule / perceptron algorithm **are entirely equivalent**, that is, they yield the same result for the same order.
- To see this, recall the relationship of  $\vec{a}^\circ$  and  $\vec{b}$ , that is,  $\vec{a}^\circ = \mathbf{X}^{\circ\top} \vec{b} = \sum_{k=1}^n b_k \vec{x}_k^\circ$ , and consider the update rule (for the  $i$ -th training pattern):

$$\begin{aligned}\vec{a}^{\circ(\text{new})} &= \vec{a}^{\circ(\text{old})} + y_i \vec{x}_i^\circ \Leftrightarrow \vec{a}^{\circ(\text{new})} = \sum_{k=1}^n b_k^{(\text{old})} \vec{x}_k^\circ + y_i \vec{x}_i^\circ \\ &\Leftrightarrow \vec{a}^{\circ(\text{new})} = \sum_{k=1; k \neq i}^n b_k^{(\text{old})} \vec{x}_k^\circ + (b_i^{(\text{old})} + y_i) \vec{x}_i^\circ \\ &\Leftrightarrow \vec{a}^{\circ(\text{new})} = \sum_{k=1}^n b_k^{(\text{new})} \vec{x}_k^\circ \\ &\text{where } b_k^{(\text{new})} = \begin{cases} b_k^{(\text{old})} + y_i & \text{if } k = i, \\ b_k^{(\text{old})} & \text{otherwise.} \end{cases}\end{aligned}$$

- Note also that in the **dual form** one may use the **regularized matrix**  $\mathbf{G}^\circ + \lambda \mathbf{I}_n$ .

# Linear Classification: Delta Rule / Perceptron Algorithm

- Simple data set with two classes:
  - 20 data points belong to the **blue** class,
  - 20 data points belong to the **red** class.
- Result of training with the Delta rule.  
(Note that the result depends on the order in which the data points are processed.)

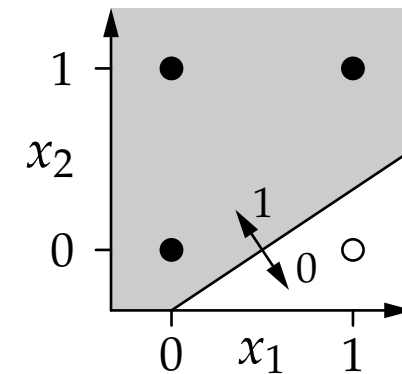


- Although training with the Delta rule always yields a solution (if one exists), the result can exhibit undesirable properties, e.g., the decision border lies unnecessarily close to data points.
- This is the case here (cyan line), esp. compared to the truth (yellow line).

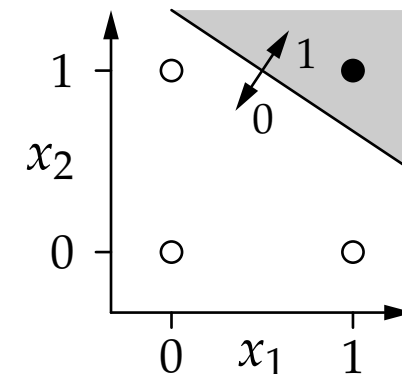
# Linear Classification: Delta Rule / Perceptron Algorithm

$x_1$	$x_2$	$y$	$\vec{a}^{\circ\top} \vec{x}^{\circ}$	$z$	$a_0$	$a_1$	$a_2$	$b_1$	$b_2$	$b_3$	$b_4$
					0	0	0	0	0	0	0
0	0	1	0	-1	1	0	0	1	0	0	0
1	0	-1	-1	-1	0	-1	0	1	-1	0	0
0	1	1	0	-1	1	-1	1	1	-1	1	0
1	1	1	1	1	1	-1	1	1	-1	1	0
0	0	1	1	1	1	-1	1	1	-1	1	0
1	0	-1	-0	-1	0	-2	1	1	-2	1	0
0	1	1	1	1	0	-2	1	1	-2	1	0
1	1	1	-1	-1	1	-1	2	1	-2	1	1
0	0	1	1	1	1	-1	2	1	-2	1	1
1	0	-1	-0	-1	0	-2	2	1	-3	1	1
0	1	1	2	1	0	-2	2	1	-3	1	1
1	1	1	0	-1	1	-1	3	1	-3	1	2
0	0	1	1	1	1	-1	3	1	-3	1	2
1	0	-1	-0	-1	0	-2	3	1	-4	1	2
0	1	1	3	1	0	-2	3	1	-4	1	2
1	1	1	1	1	0	-2	3	1	-4	1	2
0	0	1	0	-1	1	-2	3	2	-4	1	2
1	0	-1	1	1	1	-2	3	2	-4	1	2
0	1	1	4	1	1	-2	3	2	-4	1	2
1	1	1	2	1	1	-2	3	2	-4	1	2
0	0	1	1	1	1	-2	3	2	-4	1	2
1	0	-1	1	1	1	-2	3	2	-4	1	2
0	1	1	4	1	1	-2	3	2	-4	1	2
1	1	1	2	1	1	-2	3	2	-4	1	2

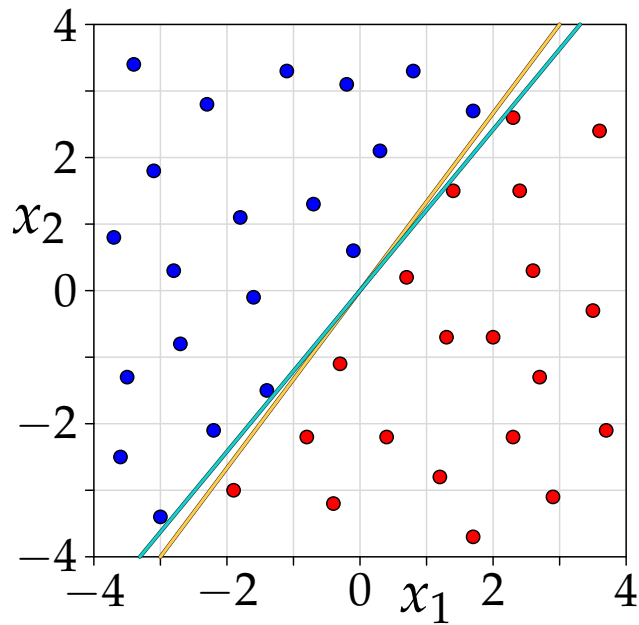
- Training process for the Boolean function  $x_1 \rightarrow x_2$ .
- Result solves the classification:



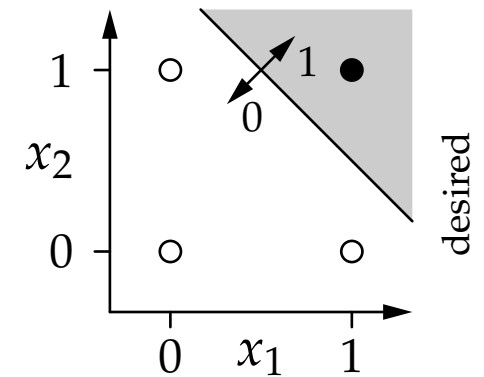
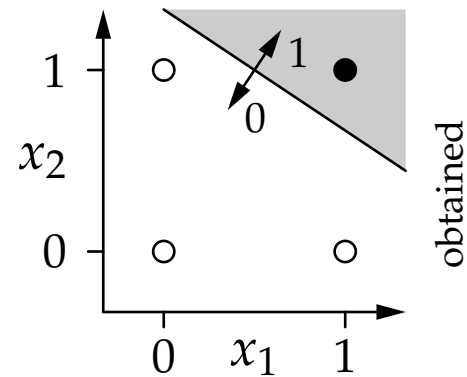
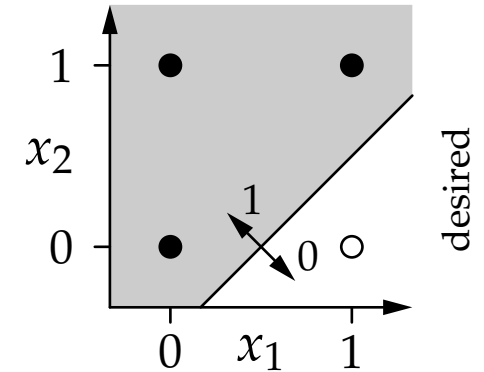
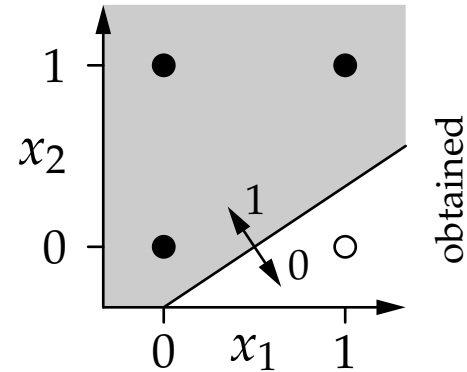
- Analog result for  $x_1 \wedge x_2$ :



# Linear Classification: Suboptimal Results



yellow: ground truth  
cyan : Delta rule result

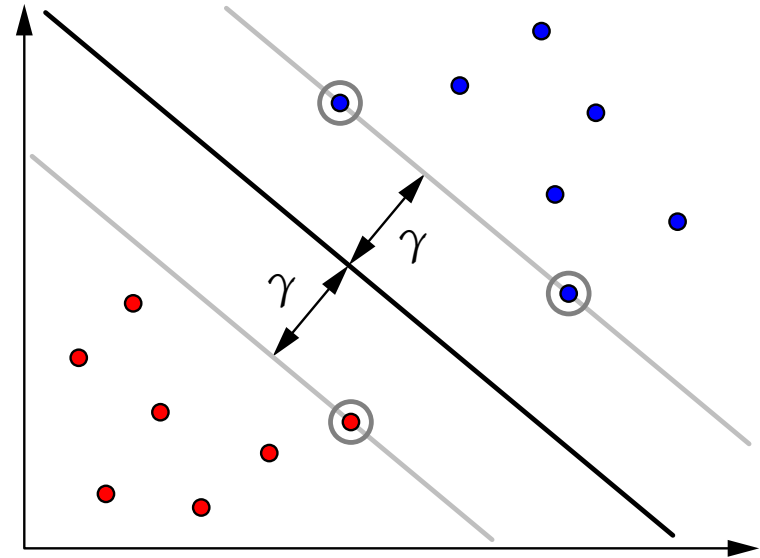


- If a (two-class) classification problem is linearly separable, then the Delta rule / perceptron algorithm finds a solution.
- However, these solutions are often (not always, though) not quite the solutions we would like to get (see examples above).



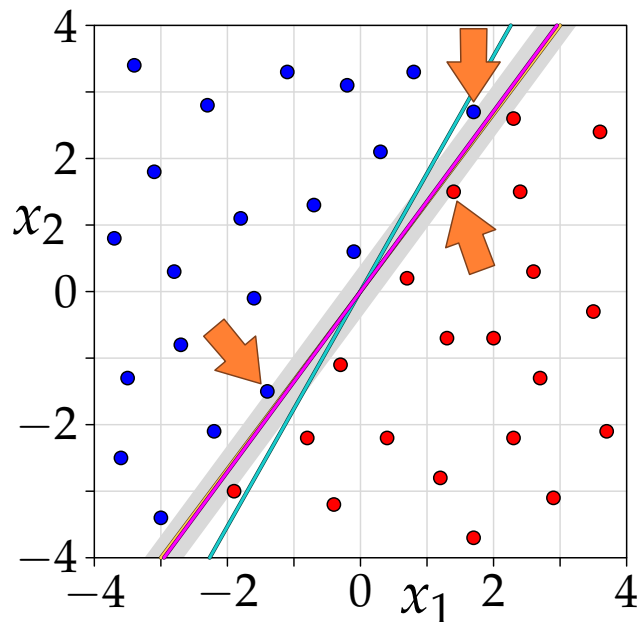
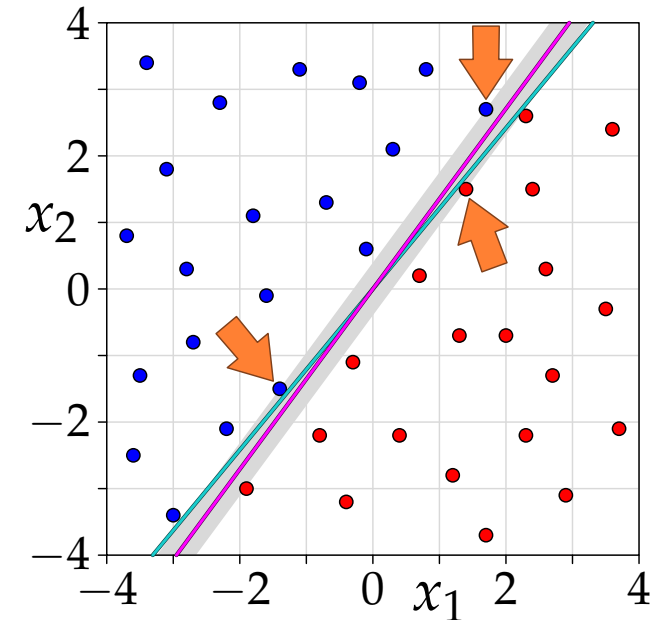
# Maximum Margin Classifiers

- In the examples just presented, the obtained result is “suboptimal” at least in an intuitive sense: the found separating line gets too close to some data points.
- It would be preferable that the line has maximum distance to the data points that are closest to it.
- The **margin**  $\gamma$  of a linear classifier is the distance of the separating line (or plane or hyperplane) to the closest (training) data point(s).
- An (intuitively) “optimal” linear classifier should maximize the margin  $\gamma$ , that is, it should be a **maximum margin classifier**.  
(Although this is intuitively plausible, there are also good theoretical arguments for this choice, which, however, we do not dive into here; see the books mentioned later.)
- The points closest to the separating line, that is, the points defining the margin, are called the **support vectors**, since they alone already fix the separating line.



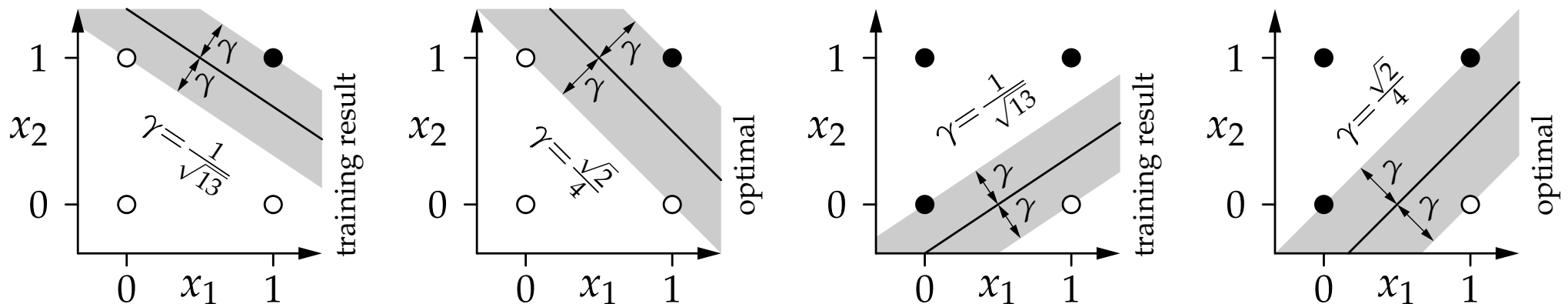
# Maximum Margin Classifiers

- Simple data set with two classes:
  - 20 data points belong to the **blue** class,
  - 20 data points belong to the **red** class.
- Result of training with the Delta rule (cyan) and maximum margin classifier (magenta). Orange arrows show the support vectors.



- Truth (yellow), lin. regression (cyan) and maximum margin classifier (magenta).
- For this (artificially constructed) example, the maximum margin classifier is very close to the ground truth.
- Of course, this depends on the data.

# Maximum Margin Classifiers



- The obtained training results are not maximum margin classifiers.  
The margin is  $\gamma = \frac{1}{\sqrt{13}} \approx 0.277$ , whereas the maximum is  $\gamma_{\text{opt}} = \frac{\sqrt{2}}{4} \approx 0.354$ .
- Up to now we considered only the sign of  $\vec{a}^{\circ\top} \vec{x}_i^{\circ}$  (or  $\vec{b}^{\top} \vec{g}_i^{\circ}$ ),  $i = 1, \dots, n$ , that is, on which side of the separating line, plane or hyperplane the data points lie.
- However,  $\vec{a}^{\circ\top} \vec{x}_i^{\circ}$  (or  $\vec{b}^{\top} \vec{g}_i^{\circ}$ ) also provides information about the distance.
- **First idea:** Do not require merely  $y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} > 0$  (or  $y_i \vec{b}^{\top} \vec{g}_i^{\circ} > 0$ ), that is, do not merely ask for a correct sign (i.e. a correct classification), but require  $y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq \gamma$  (or  $y_i \vec{b}^{\top} \vec{g}_i^{\circ} \geq \gamma$ ),  $\gamma > 0$ , and maximize  $\gamma$ .

# Maximum Margin Classifiers

- **First idea:** Require  $y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq \gamma$  (or  $y_i \vec{b}^{\top} \vec{g}_i^{\circ} \geq \gamma$ ) and maximize  $\gamma$ .
- **Problem:** Requiring merely  $y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} > 0$  (or  $y_i \vec{b}^{\top} \vec{g}_i^{\circ} > 0$ ) always yields a result that satisfies  $y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq \varepsilon$  (or  $y_i \vec{b}^{\top} \vec{g}_i^{\circ} \geq \varepsilon$ ) for some  $\varepsilon > 0$ .
- This can be turned into a solution that satisfies  $y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq \gamma$  (or  $y_i \vec{b}^{\top} \vec{g}_i^{\circ} \geq \gamma$ ), regardless of the values of  $\varepsilon$  and  $\gamma$ , by simply **scaling the parameter vectors**:

$$\text{If } y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq \varepsilon, \quad \text{then } y_i \left(\frac{\gamma}{\varepsilon} \vec{a}^{\circ}\right)^{\top} \vec{x}_i^{\circ} \geq \gamma.$$

$$\text{If } y_i \vec{b}^{\top} \vec{g}_i^{\circ} \geq \varepsilon, \quad \text{then } y_i \left(\frac{\gamma}{\varepsilon} \vec{b}\right)^{\top} \vec{g}_i^{\circ} \geq \gamma.$$

This scaling does not change the separating line, plane or hyperplane.

- Alternatively, we may say that  $\vec{a}^{\circ\top} \vec{x}_i^{\circ}$  measures the distance of a data point  $\vec{x}_i$  from the separating line, plane or hyperplane in units of  $|\vec{a}|^{-1}$  or that the distance from the line, plane or hyperplane is actually  $|\vec{a}^{\circ\top} \vec{x}_i^{\circ}| \cdot |\vec{a}|^{-1}$ .
- That is, we have to **maximize the ratio of  $\gamma$  to  $|\vec{a}|$**  to obtain a maximum margin. To achieve this, we can maximize  $\gamma$  for fixed  $|\vec{a}|$  or minimize  $|\vec{a}|$  for fixed  $\gamma$ .

# Maximum Margin Classifiers

- Therefore, in order to find a maximum margin classifier, we have to

$$\begin{array}{ll} \text{either} & \text{maximize } \gamma \\ & \text{subject to } |\vec{a}| = 1 \quad \text{and} \quad \forall i; 1 \leq i \leq n : \quad y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq \gamma \end{array}$$

$$\begin{array}{ll} \text{or} & \text{minimize } |\vec{a}| \\ & \text{subject to } \gamma = 1 \quad \text{and} \quad \forall i; 1 \leq i \leq n : \quad y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq \gamma. \end{array}$$

- It is more common to use the second form and to set  $\gamma = 1$  directly:

$$\begin{array}{ll} \text{minimize} & |\vec{a}| \\ \text{subject to} & \forall i; 1 \leq i \leq n : \quad y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq 1. \end{array}$$

- Furthermore, we modify this slightly for mathematical convenience:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2} |\vec{a}|^2 = \frac{1}{2} \vec{a}^{\top} \vec{a} \\ \text{subject to} & \forall i; 1 \leq i \leq n : \quad y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq 1. \end{array}$$

Squaring  $|\vec{a}|$  does not change the solution, because it is a monotone operation; and adding the factor  $\frac{1}{2}$  comes in handy when taking derivatives later.

# Reminder: Function Optimization

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

**Often feasible approach:**

- A necessary condition for a (local) optimum (minimum/maximum) is that the partial derivatives w.r.t. the parameters vanish [Pierre de Fermat, 1607–1665].
- Therefore: (Try to) solve the equation system that results from setting all partial derivatives w.r.t. the parameters equal to zero.

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

**Solution procedure:**

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

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

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

# Function Optimization with Constraints

Often a function has to be optimized subject to certain **constraints**.

**First:** restriction to  $k$  **equality constraints**  $C_i(\vec{x}) = 0, i = 1, \dots, k$ .

**Note:** the equality constraints describe a subspace of the domain of the function.

**Problem** of optimization with constraints:

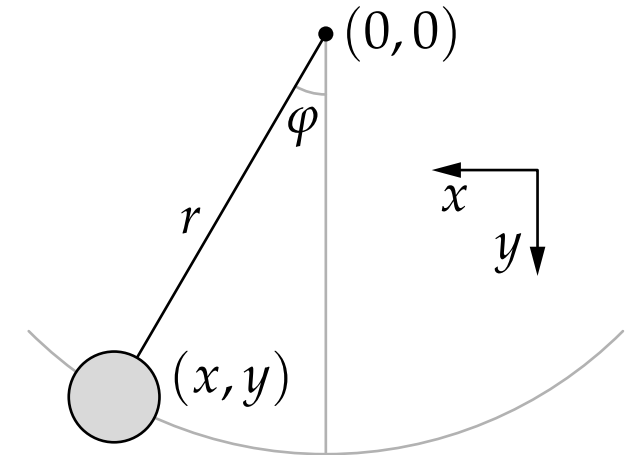
- The gradient of the objective function  $f$  may vanish outside the constrained subspace, leading to an unacceptable solution (violating the constraints).
- At an optimum *in the constrained subspace* the derivatives need not vanish.

One way to handle this problem are **generalized coordinates**:

- Exploit the dependence between the parameters specified in the constraints to express some parameters in terms of the others and thus reduce the set  $\vec{x}$  to a set  $\vec{x}'$  of independent parameters (*generalized coordinates*).
- Problem: Can be clumsy and cumbersome, if possible at all, because the form of the constraints may not allow for expressing some parameters as proper functions of the others.

# Generalized Coordinates: Example

- Consider a simple **stick pendulum**, that is, a mass at the end of a stick suspended from a hub at the other end (like in the diagram on the right).



- Let the location of the mass be described by **Cartesian coordinates**  $x$  and  $y$  (coordinate origin is at the hub).

- Any optimization problem involving the location  $(x, y)$  of the mass is constrained by  $x^2 + y^2 = r^2$  (because the stick is rigid, its length is constant).

$$\Rightarrow y = \sqrt{r^2 - x^2}$$

- This constraint may also be eliminated by switching to **polar coordinates**:

$$r = \text{hypot}(x, y) = \sqrt{x^2 + y^2} \quad \text{and} \quad \varphi = \text{atan2}(x, y).$$

- Because  $r$  is constant (since the stick length is fixed), one only needs to consider the angle  $\varphi$  (which may be varied freely). Hence the angle  $\varphi$  may be used as a **generalized coordinate**.



# Function Optimization with Constraints

A much more elegant approach is based on the following nice insights:

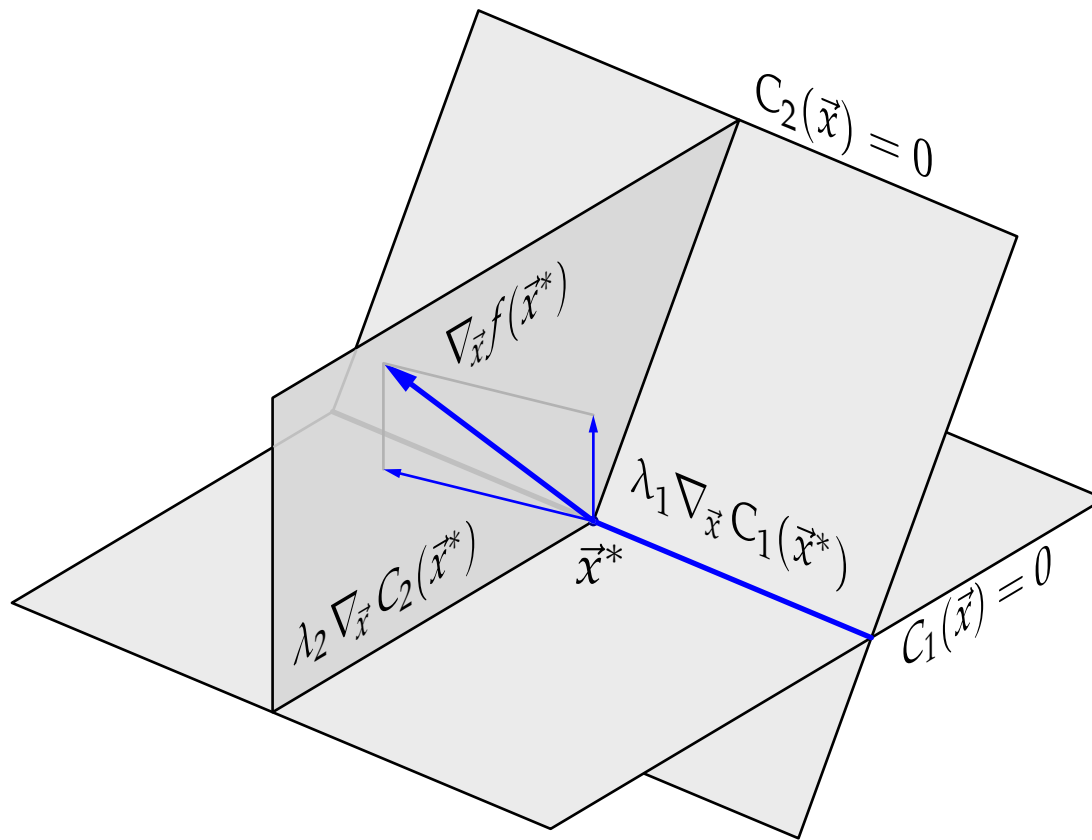
Let  $\vec{x}^*$  be a (local) optimum of  $f(\vec{x})$  in the constrained subspace. Then:

- The gradient  $\nabla_{\vec{x}} f(\vec{x}^*)$ , if it does not vanish, must be **perpendicular** to the constrained subspace. (If  $\nabla_{\vec{x}} f(\vec{x}^*)$  had a component in the constrained subspace,  $\vec{x}^*$  would not be a (local) optimum in this subspace.)
- The gradients  $\nabla_{\vec{x}} C_j(\vec{x}^*)$ ,  $1 \leq j \leq k$ , must all be **perpendicular** to the constrained subspace, because they are constant, namely 0, in this subspace. Together they span the subspace perpendicular to the constrained subspace.
- Therefore it must be possible to find values  $\lambda_j$ ,  $1 \leq j \leq k$ , such that

$$\nabla_{\vec{x}} f(\vec{x}^*) - \sum_{j=1}^k \lambda_j \nabla_{\vec{x}} C_j(\vec{x}^*) = \vec{0}.$$

If the constraints (and thus their gradients) are linearly independent, the values  $\lambda_j$  are uniquely determined. This equation can be used to **compensate the gradient** of  $f(\vec{x}^*)$  so that it vanishes at  $\vec{x}^*$ .

# Function Optimization with Constraints



$$\nabla_{\vec{x}} f(\vec{x}^*) - \lambda_1 \nabla_{\vec{x}} C_1(\vec{x}^*) - \lambda_2 \nabla_{\vec{x}} C_2(\vec{x}^*) = \vec{0}$$

- The gradient  $\nabla_{\vec{x}} f(\vec{x}^*)$  can be compensated with the gradients  $\nabla_{\vec{x}} C_1(\vec{x}^*)$  and  $\nabla_{\vec{x}} C_2(\vec{x}^*)$  using suitable factors  $\lambda_1$  and  $\lambda_2$ .
- These factors are called **Lagrange multipliers**.
- Due to these factors, the gradient of an enhanced function vanishes at the (local) optimum  $\vec{x}^*$  in the constrained subspace.

- The planes  $C_1(\vec{x}) = 0$  and  $C_2(\vec{x}) = 0$  represent two (here: linear) constraints.
- Their intersection (both conditions hold, blue line) is the constrained subspace.

# Function Optimization: Lagrange Theory

As a consequence of these insights we obtain the

## Method of Lagrange Multipliers:

[Joseph-Louis Lagrange, 1736–1813]

**Given:**

- a function  $f(\vec{x})$ , which is to be optimized,
- $k$  equality constraints  $C_j(\vec{x}) = 0, 1 \leq j \leq k$ .

## Procedure:

1. Construct the so-called **Lagrange function** by incorporating the equality constraints  $C_i, i = 1, \dots, k$ , with (unknown) **Lagrange multipliers**  $\lambda_i$ :

$$\mathcal{L}(\vec{x}, \lambda_1, \dots, \lambda_k) = f(\vec{x}) - \sum_{i=1}^k \lambda_i C_i(\vec{x}).$$

2. Set the partial derivatives of the Lagrange function equal to zero:

$$\frac{\partial \mathcal{L}}{\partial x_1} = 0, \quad \dots, \quad \frac{\partial \mathcal{L}}{\partial x_m} = 0, \quad \frac{\partial \mathcal{L}}{\partial \lambda_1} = 0, \quad \dots, \quad \frac{\partial \mathcal{L}}{\partial \lambda_k} = 0.$$

3. (Try to) Solve the resulting equation system.

# Function Optimization: Lagrange Theory

## Observations:

- Due to the representation of the gradient of  $f(\vec{x})$  at a local optimum  $\vec{x}^*$  in the constrained subspace (see above) the gradient of  $\mathcal{L}$  w.r.t.  $\vec{x}$  vanishes at  $\vec{x}^*$ .  
→ The standard approach works again!
- If the constraints are satisfied, the additional terms have no influence.  
→ The original task is not modified (same objective function).
- Taking the partial derivative w.r.t. a Lagrange multiplier reproduces the corresponding equality constraint (though negated):

$$\forall j; 1 \leq j \leq k : \quad \frac{\partial}{\partial \lambda_j} \mathcal{L}(\vec{x}, \lambda_1, \dots, \lambda_k) = -C_j(\vec{x}) \stackrel{!}{=} 0,$$

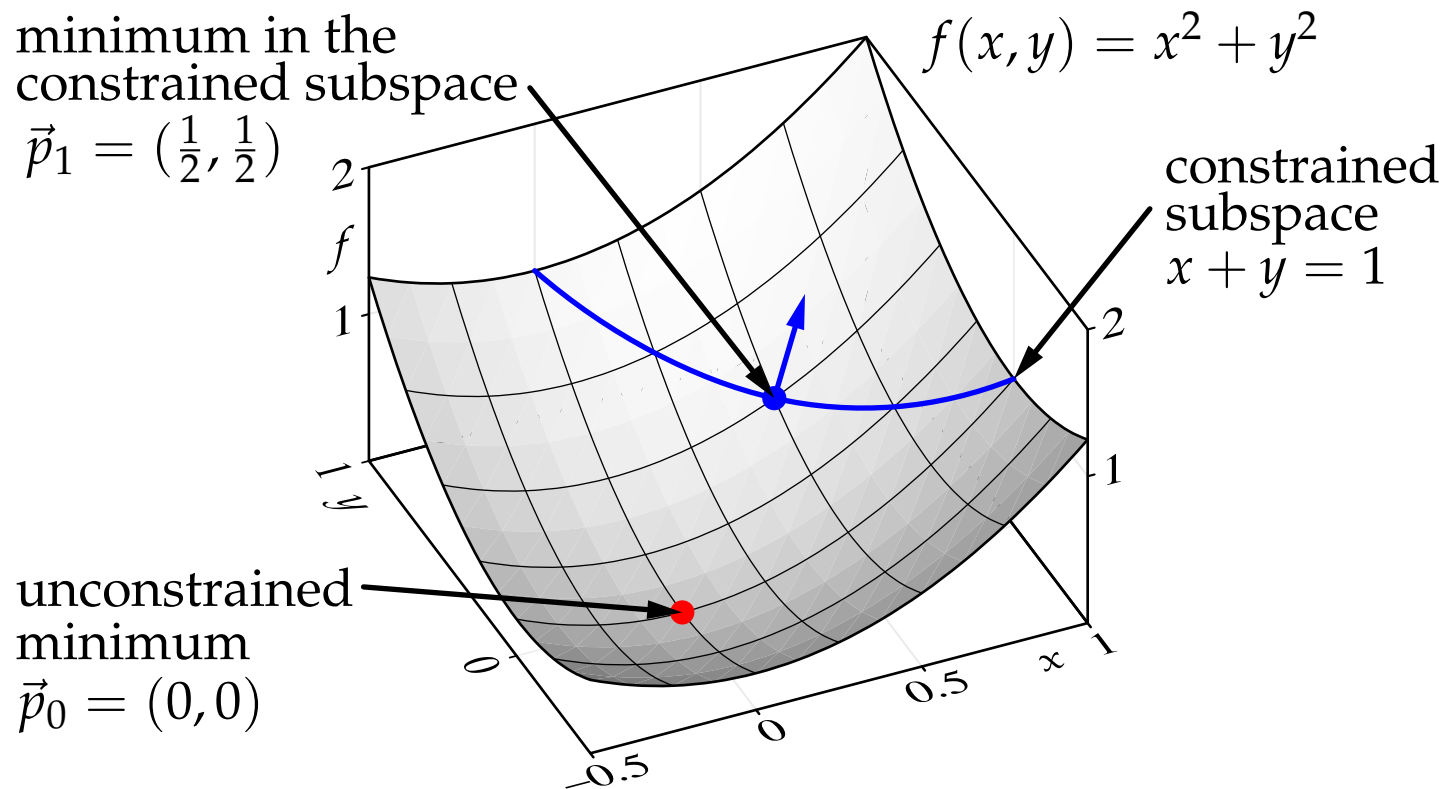
→ Constraints enter the equation system to solve in a natural way.

**Remark:** Up to now we considered only equality constraints.

- **Inequality constraints** can be handled with **Karush–Kuhn–Tucker theory**.

# Lagrange Theory: Example 1

**Example task:** Minimize  $f(x, y) = x^2 + y^2$  subject to  $x + y = 1$ .



The unconstrained minimum is not in the constrained subspace.  
At the minimum in the constrained subspace the gradient of  $f$  does not vanish.

# Lagrange Theory: Example 1

**Example task:** Minimize  $f(x, y) = x^2 + y^2$  subject to  $x + y = 1$ .

**Solution procedure:**

1. Rewrite the constraint, so that one side becomes zero:  $x + y - 1 = 0$ .
2. Construct the Lagrange function by incorporating the constraint into the objective function with a Lagrange multiplier  $\lambda$ :

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

3. Take the partial derivatives of the Lagrange function and set them equal to zero (necessary conditions for a minimum):

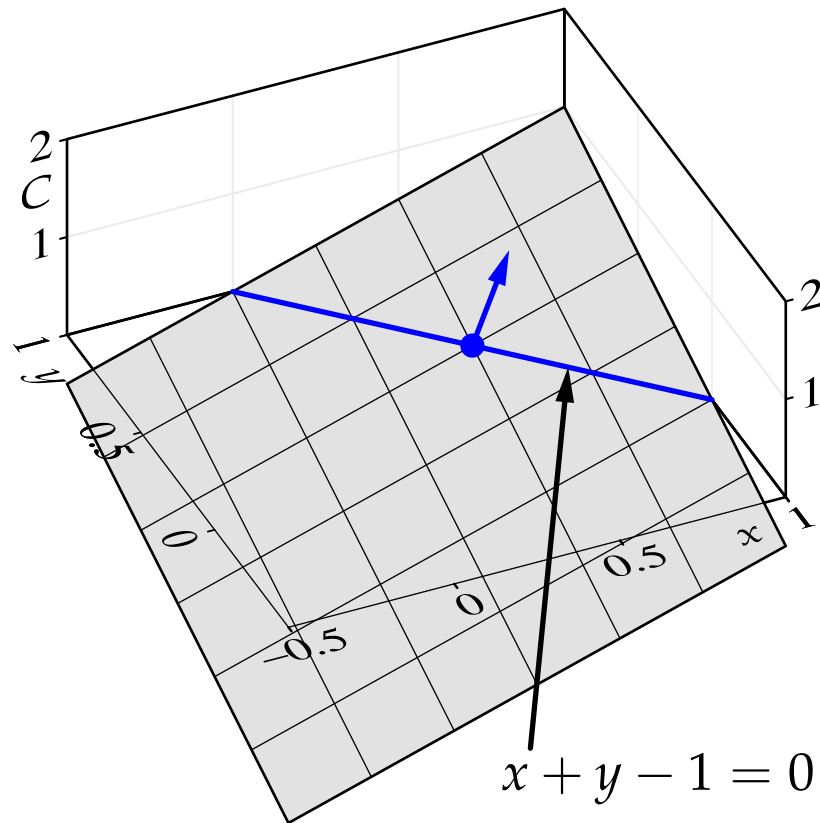
$$\frac{\partial \mathcal{L}}{\partial x} = 2x - \lambda \stackrel{!}{=} 0, \quad \frac{\partial \mathcal{L}}{\partial y} = 2y - \lambda \stackrel{!}{=} 0, \quad \frac{\partial \mathcal{L}}{\partial \lambda} = -(x + y - 1) \stackrel{!}{=} 0.$$

4. Solve the resulting (here: linear) equation system:

$$\lambda = 1, \quad x = y = \frac{1}{2}.$$

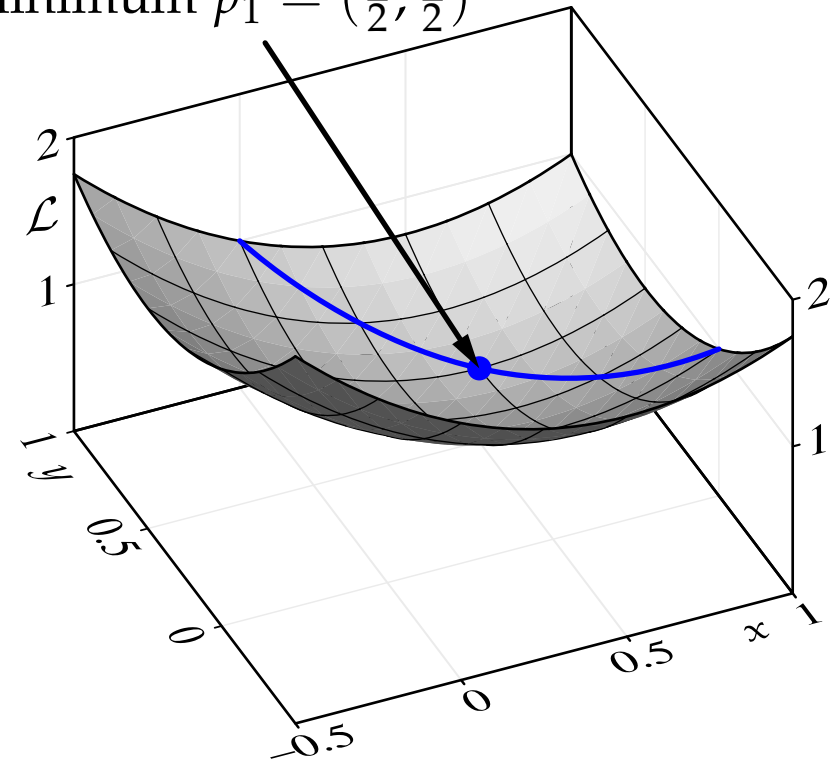
# Lagrange Theory: Example 1

$$C(x, y) = x + y - 1$$



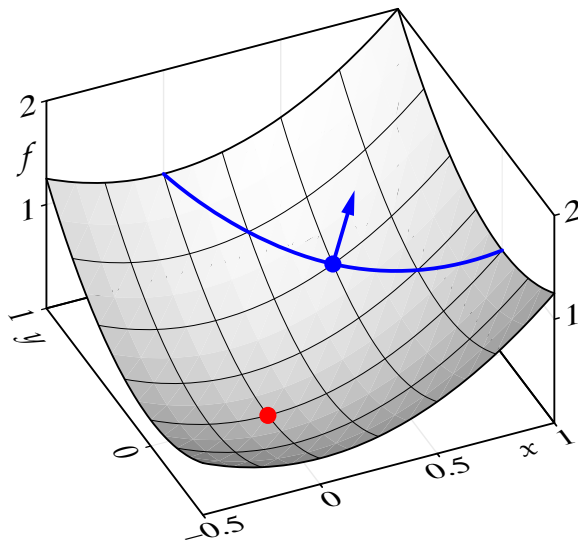
$$\mathcal{L}(x, y, 1) = x^2 + y^2 - (x + y - 1)$$

$$\text{minimum } \vec{p}_1 = \left(\frac{1}{2}, \frac{1}{2}\right)$$

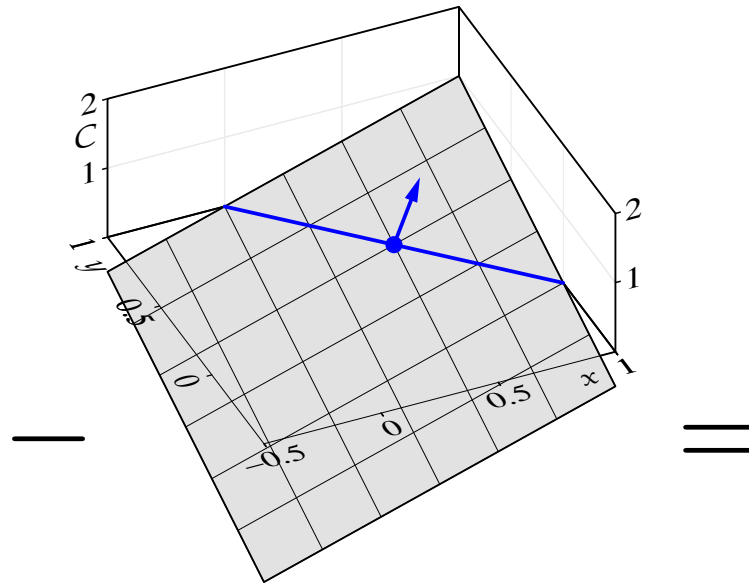


The gradient of the constraint is perpendicular to the constrained subspace.  
The (unconstrained) minimum of the Lagrange function  $\mathcal{L}(x, y, \lambda)$   
is the minimum of the objective function  $f(x, y)$  in the constrained subspace.

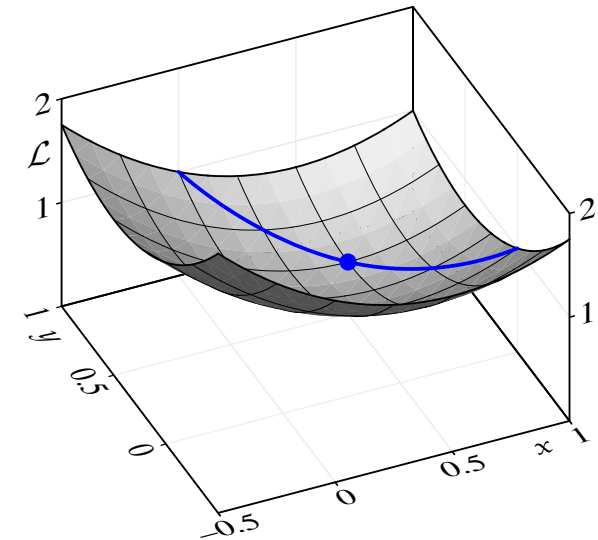
# Lagrange Theory: Example 1



$$f(x, y) = x^2 + y^2$$



$$C(x, y) = x + y - 1$$



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

The gradient of the function  $f$  is perpendicular to the constrained subspace.  
The gradient of the constraint  $C$  is perpendicular to the constrained subspace.  
With a proper Lagrange multiplier  $\lambda$ , the gradient of the function  $f$  can be compensated by the gradient of the constraint  $C$ .  
The (unconstrained) minimum of the Lagrange function  $\mathcal{L}(x, y, \lambda)$  is the minimum of the objective function  $f(x, y)$  in the constrained subspace.



## Lagrange Theory: Example 2

**Example task:** Find the side lengths  $x, y, z$  of a box with maximum volume for a given area  $S$  of the surface.

**Formally:** Maximize  $f(x, y, z) = xyz$   
subject to  $2xy + 2xz + 2yz = S$ .

**Solution procedure:**

1. The constraint is  $C(x, y, z) = 2xy + 2xz + 2yz - S = 0$ .

2. The Lagrange function is

$$\mathcal{L}(x, y, z, \lambda) = xyz - \lambda(2xy + 2xz + 2yz - S).$$

3. Taking the partial derivatives yields (in addition to the constraint):

$$\frac{\partial \mathcal{L}}{\partial x} = yz - 2\lambda(y + z) \stackrel{!}{=} 0, \quad \frac{\partial \mathcal{L}}{\partial y} = xz - 2\lambda(x + z) \stackrel{!}{=} 0, \quad \frac{\partial \mathcal{L}}{\partial z} = xy - 2\lambda(x + y) \stackrel{!}{=} 0.$$

4. The solution is:  $\lambda = \frac{1}{4}\sqrt{\frac{S}{6}}, \quad x = y = z = \sqrt{\frac{S}{6}}$  (i.e., the box is a cube).

# Lagrange Theory with Inequality Constraints

- The general problem of minimizing functions subject to constraints is

$$\text{minimize (w.r.t. } \vec{x}) \quad f(\vec{x}) \quad \text{subject to} \quad \begin{cases} C_i(\vec{x}) = 0, & i \in \mathcal{E}, \\ C_i(\vec{x}) \geq 0, & i \in \mathcal{I}, \end{cases}$$

where  $\mathcal{E}$  is the set of indices of the equality constraints  
and  $\mathcal{I}$  is the set of indices of the inequality constraints.

- This problem is also approached with the **method of Lagrange multipliers**:

$$\mathcal{L}(\vec{x}, \vec{\lambda}) = f(\vec{x}) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i C_i(\vec{x}).$$

- The **Karush–Kuhn–Tucker conditions** state that if  $\vec{x}^*$  is a (local) minimum of  $f$  satisfying all  $C_i$ , then there exists a vector  $\vec{\lambda}^*$  of Lagrange multipliers such that

$$\nabla_{\vec{x}} \mathcal{L}(\vec{x}^*, \vec{\lambda}^*) = \vec{0},$$

$$\forall i \in \mathcal{E} : \quad C_i(\vec{x}^*) = 0,$$

$$\forall i \in \mathcal{I} : \quad \lambda_i \geq 0,$$

$$\forall i \in \mathcal{I} : \quad C_i(\vec{x}^*) \geq 0,$$

$$\forall i \in \mathcal{E} \cup \mathcal{I} : \quad \lambda_i C_i(\vec{x}^*) = 0.$$

# Lagrange Theory with Inequality Constraints

- Proof of Karush–Kuhn–Tucker conditions:  $\Rightarrow$

- At a feasible point  $\vec{x}$  ( $\vec{x}$  satisfies the constraints), a constraint  $C_i$  is said to be **active** if  $C_i(\vec{x}) = 0$  and **inactive** if  $C_i(\vec{x}) > 0$ .

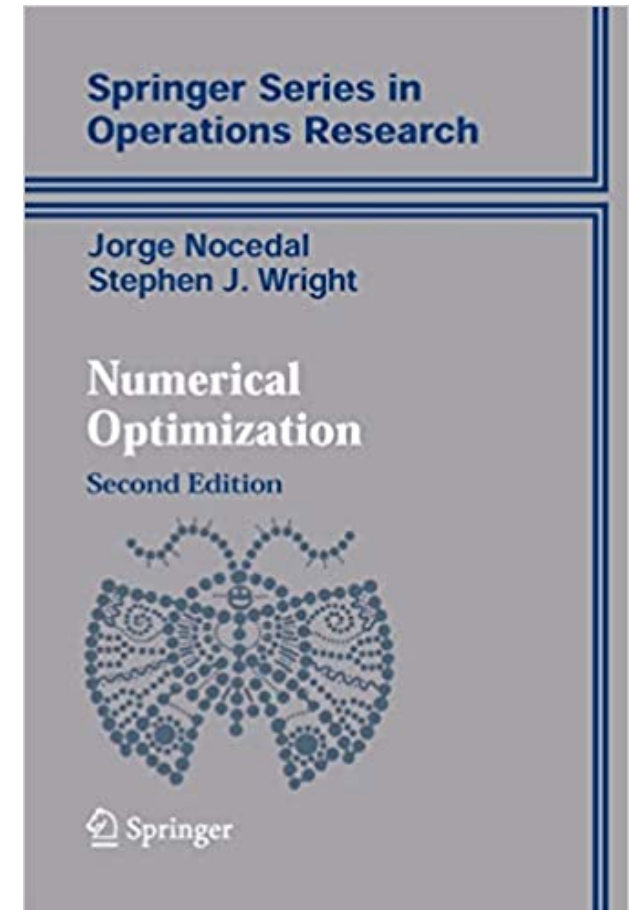
$\Rightarrow$  equality constraints are always active,  
inequality constraints only sometimes.

- The Karush–Kuhn–Tucker conditions state that either an inequality constraint is active or its Lagrange multiplier is 0, due to the so-called **complementary conditions**

$$\forall i \in \mathcal{E} \cup \mathcal{I} : \quad \lambda_i C_i(\vec{x}^*) = 0.$$

- These conditions are fairly intuitive:

If an inequality constraint is inactive at  $\vec{x}$ , one can make a (sufficiently small) step in any direction without violating the constraint. If, however, it is active, it needs a positive Lagrange multiplier to compensate the gradient of  $f$ .

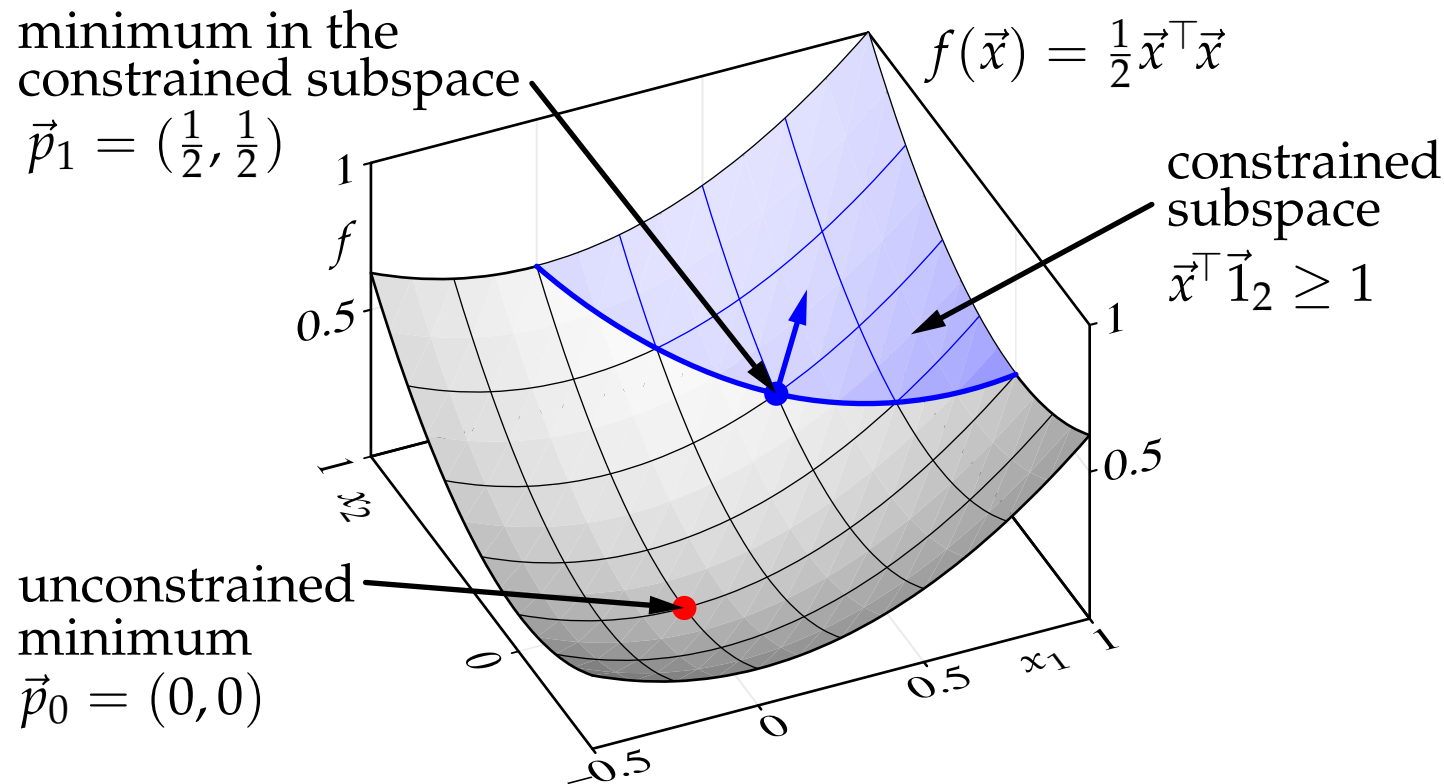


# Lagrange Theory with Inequality Constraints

- Why are the Lagrange multipliers of inequality constraints non-negative?
- Equality constraints are not uniquely directed, because  $C(\vec{x}) = 0$  and  $-C(\vec{x}) = 0$  are the same constraint (but the form fixes the sign of the Lagrange multiplier).  
(cf. Example 1: If  $C(x, y) = x + y - 1$ , then  $\lambda = +1$ , but if  $C(x, y) = 1 - x - y$ , then  $\lambda = -1$ .)
- Inequality constraints, however, are uniquely directed, because  $C(\vec{x}) \geq 0$  and  $-C(\vec{x}) \geq 0 \Leftrightarrow C(\vec{x}) \leq 0$  are (obviously) **not** the same constraint.
- The gradient of an inequality constraint points in the positive direction, that is, towards increasing values of  $C(\vec{x})$ . (After all, that is what “gradient” means.)
- A constraint appears in the Lagrange function as  $-\lambda C(\vec{x})$ .  
Therefore we already have the correct sign for compensating the gradient of the objective function and the Lagrange multiplier cannot be negative.
- Intuitively: The size of the Lagrange multiplier of an active constraint at a solution  $\vec{x}^*$  can be interpreted as how hard the constraint “pushes” against the gradient of the objective function (in order to compensate it).

# Lagrange Theory: Inequality Constraint Example

**Example task:** Minimize  $f(\vec{x}) = \frac{1}{2} \vec{x}^\top \vec{x}$  subject to  $\vec{x}^\top \vec{1}_2 \geq 1$ .



The unconstrained minimum is not in the constrained subspace.  
At the minimum in the constrained subspace the gradient of  $f$  does not vanish.

# Lagrange Theory: Inequality Constraint Example

**Example task:** Minimize  $f(\vec{x}) = \frac{1}{2} \vec{x}^\top \vec{x}$  subject to  $\vec{x}^\top \vec{1}_2 \geq 1$ .

(Note that with the exception of the factor  $\frac{1}{2}$  and the inequality constraint, this is completely analogous to Example 1.)

- Construct the Lagrange function by incorporating the (inequality) constraint into the objective function with a Lagrange multiplier  $\lambda$ :

$$\mathcal{L}(\vec{x}, \lambda) = \frac{1}{2} \vec{x}^\top \vec{x} - \lambda(\vec{x}^\top \vec{1}_2 - 1).$$

- Form the gradient of this Lagrange function w.r.t.  $\vec{x}$  and set it to  $\vec{0}$ :

$$\nabla_{\vec{x}} \mathcal{L}(\vec{x}, \lambda) = \vec{x} - \lambda \vec{1}_2 \stackrel{!}{=} \vec{0} \quad \Leftrightarrow \quad \vec{x} = \lambda \vec{1}_2.$$

- Substitute the result back into the Lagrange function:

$$\mathcal{L}(\lambda) = \frac{1}{2}(\lambda \vec{1}_2)^\top \lambda \vec{1}_2 - \lambda(\lambda \vec{1}_2^\top \vec{1}_2 - 1) = \lambda^2 - 2\lambda^2 + \lambda = \lambda - \lambda^2.$$

- The reduced Lagrange function  $\mathcal{L}(\lambda) = \lambda - \lambda^2$  has to be maximized for  $\lambda \geq 0$ .
- Clearly, the solution is  $\lambda = \frac{1}{2}$ , and hence the optimal  $\vec{x}$  is  $\vec{x} = \frac{1}{2} \cdot \vec{1}_2 = (\frac{1}{2}, \frac{1}{2})^\top$ .

# Lagrange Theory: Duality

- For the **maximum margin classifier**, we arrived at this problem:

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \vec{a}^\top \vec{a} \\ & \text{subject to} && \forall i; 1 \leq i \leq n : \quad y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} - 1 \geq 0. \end{aligned}$$

Note that we have **only linear inequality constraints**, no equality constraints.

- This is called the **primal form** of the optimization problem.  
It has the Lagrange function (where  $\vec{b}$  comprises the Lagrange multipliers)

$$\mathcal{L}(\vec{a}^{\circ}, \vec{b}) = \frac{1}{2} \vec{a}^\top \vec{a} - \sum_{i=1}^n b_i (y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} - 1)$$

- A necessary condition for an optimum is (first Karush–Kuhn–Tucker condition):

$$\nabla_{\vec{a}^{\circ}} \mathcal{L}(\vec{a}^{\circ*}, \vec{b}^*) = \begin{bmatrix} 0 \\ \vec{a}^* \end{bmatrix} - \sum_{i=1}^n b_i^* y_i \vec{x}_i^{\circ} \stackrel{!}{=} 0 \quad \Leftrightarrow \quad 0 = \sum_{i=1}^n b_i^* y_i \quad \wedge \quad \vec{a}^* = \sum_{i=1}^n b_i^* y_i \vec{x}_i^{\circ}.$$

That is, we have found a way of expressing the **primal parameters**  $\vec{a}$  in terms of the Lagrange multipliers, which are the **dual parameters**  $\vec{b}$ .

# Lagrange Theory: Duality

- Substituting  $\vec{a} = \sum_{i=1}^n b_i y_i \vec{x}_i^\circ$  back into the Lagrange function yields

$$\begin{aligned}
 \mathcal{L}(\vec{b}) &= \frac{1}{2} \left( \sum_{j=1}^n b_j y_j \vec{x}_j \right)^\top \left( \sum_{j=1}^n b_j y_j \vec{x}_j \right) - \sum_{i=1}^n b_i \left( y_i \left( \sum_{j=1}^n b_j y_j \vec{x}_j \right)^\top \vec{x}_i + y_i a_0 - 1 \right) \\
 &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n b_i y_i \vec{x}_i^\top \vec{x}_j y_j b_j - \sum_{i=1}^n \sum_{j=1}^n b_i y_i \vec{x}_i^\top \vec{x}_j y_j b_j - a_0 \underbrace{\sum_{i=1}^n y_i b_i}_{=0 \text{ for } b_i = b_i^*} + \sum_{i=1}^n b_i \\
 &= \sum_{i=1}^n b_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n b_i y_i \vec{x}_i^\top \vec{x}_j y_j b_j \\
 &= \vec{1}_n^\top \vec{b} - \frac{1}{2} \vec{b}^\top \text{diag}(\vec{y}) \mathbf{G} \text{diag}(\vec{y}) \vec{b} \quad \left[ = \vec{1}_n^\top \vec{b} - \frac{1}{2} (\vec{b} \odot \vec{y})^\top \mathbf{G} (\vec{y} \odot \vec{b}) \right],
 \end{aligned}$$

( $\odot$  is the so-called Hadamard product)

where  $\vec{1}_n$  is an  $n$ -dim. vector with all elements 1,  $\mathbf{G} = \mathbf{X}\mathbf{X}^\top$  is the Gram matrix, and  $\text{diag}(\vec{y})$  is a diagonal  $n \times n$  matrix with the values of  $\vec{y}$  on its diagonal.

- This Lagrange function is also called the **reduced Lagrange function**, because the **primal parameters**  $\vec{a}$  have been **eliminated** from it.
- We now turn this Lagrange function back into an optimization problem.



# Lagrange Theory: Duality

- The optimization problem corresponding to the (reduced) Lagrange function

$$\mathcal{L}(\vec{b}) = \vec{1}_n^\top \vec{b} - \frac{1}{2} \vec{b}^\top \text{diag}(\vec{y}) \mathbf{G} \text{diag}(\vec{y}) \vec{b}$$

$$\begin{aligned} \text{is} \quad & \text{maximize} \quad \vec{1}_n^\top \vec{b} - \frac{1}{2} \vec{b}^\top \text{diag}(\vec{y}) \mathbf{G} \text{diag}(\vec{y}) \vec{b} \\ & \text{subject to} \quad \forall i; 1 \leq i \leq n: b_i \geq 0 \quad \text{and} \quad \vec{b}^\top \vec{y} = 0. \end{aligned}$$

(Note that  $\vec{b}^\top \vec{y} = 0$  is simply the condition  $\sum_{i=1}^n b_i y_i = 0$  derived above.)

- Why is this not a minimization problem like the primal problem?
- $\mathbf{M} = \text{diag}(\vec{y}) \mathbf{G} \text{diag}(\vec{y}) = \text{diag}(\vec{y}) \mathbf{X} \mathbf{X}^\top \text{diag}(\vec{y}) = (\text{diag}(\vec{y}) \mathbf{X})(\text{diag}(\vec{y}) \mathbf{X})^\top$  is positive semi-definite, because it is (also) a Gram matrix (like  $\mathbf{G}$ ).
- Since  $\mathbf{M}$  is positive semi-definite, the quadratic form  $\vec{b}^\top \mathbf{M} \vec{b}$  is convex.
- The term  $\vec{1}_n^\top \vec{b}$  is linear in  $\vec{b}$  and thus the Lagrange function  $\mathcal{L}(\vec{b})$  is **concave**.
- Therefore  $\mathcal{L}(\vec{b})$  has to be maximized to find a solution. (More general justification:  
see the book on slide 155.)

# Lagrange Theory: Duality

- However, it is more common to write the problem as a minimization problem (by simply negating the objective):

$$\begin{aligned} &\text{minimize} && \frac{1}{2} \vec{b}^\top \text{diag}(\vec{y}) \mathbf{G} \text{diag}(\vec{y}) \vec{b} - \vec{1}_n^\top \vec{b} \\ &\text{subject to} && \forall i; 1 \leq i \leq n: b_i \geq 0 \quad \text{and} \quad \vec{b}^\top \vec{y} = 0. \end{aligned}$$

By the same arguments as before, this objective is **convex**.

- A solution of this optimization problem is an estimate  $\hat{\vec{b}}$  of the **dual parameters**. The data points  $\vec{x}_i$  for which  $\hat{b}_i > 0$  are the **support vectors**.

- The corresponding values  $\hat{\vec{a}}$  of the **primal parameters** are obtained via

$$\hat{\vec{a}} = \sum_{i=1}^n \hat{b}_i y_i \vec{x}_i.$$

- The value of  $\hat{a}_0$  may be found via a Karush–Kuhn–Tucker condition

$$\hat{b}_i (y_i (\hat{\vec{a}}^\top \vec{x}_i + \hat{a}_0) - 1) = 0 \quad \text{for some } i \text{ with } \hat{b}_i > 0,$$

or as a weighted average of the  $\hat{b}_i$ :

$$\hat{a}_0 = \sum_{i=1}^n \hat{b}_i (y_i - \hat{\vec{a}}^\top \vec{x}_i) \bigg/ \sum_{i=1}^n \hat{b}_i.$$

# Soft Margin Classifiers: Slack Variables

- Up to now we assumed that the data is **linearly separable** and that therefore a perfect solution, a so-called **hard margin classifier**, can be obtained.
- This assumption is, of course, unrealistic. Even if linear separation is feasible, real data usually does not allow for a “clean” linear separation.
- To handle this problem, we introduce a set of so-called **slack variables**, by which we intend to allow a few data points to lie inside the margin or even on the wrong side of the separating line, plane or hyperplane.
- Formally, we change the constraints to  $y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq 1 - \zeta_i$ ,  $i = 1, \dots, n$ , where the  $\zeta_i \geq 0$  are slack variables, which relax the constraints.
- Of course, we have to limit the slack variables, otherwise the classification constraints have no effect anymore. We do this by **penalizing their sum**.
- That is, we add a term  $\nu \sum_{i=1}^n \zeta_i$  to the objective, where  $\nu$  is a parameter. An common alternative parameterization is  $\frac{1}{C} \sum_{i=1}^n \zeta_i$  with the parameter  $C$ .
- Clearly, the hard margin classifier is recovered for  $\nu \rightarrow \infty$  or  $C \rightarrow 0$ .

# Soft Margin Classifiers

- Thus we arrive at the optimization problem of a **soft margin classifier**:

$$\text{minimize} \quad \frac{1}{2} \vec{a}^\top \vec{a} + \nu \sum_{i=1}^n \xi_i$$

$$\text{subject to} \quad \forall i; 1 \leq i \leq n : \quad y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0.$$

- Trade-off between a larger margin (i.e. smaller  $|\vec{a}|$ ) and smaller values of the  $\xi_i$ .
- The larger we choose  $\nu$ , the less we accept violations of the constraints.  
The smaller we choose  $\nu$ , the “softer” the margin will be (more violations).
- A smaller margin usually gives better results on the training data, but possibly worse results on new data.
- A larger margin (which is then more often violated), may give worse results on the training data, but is possibly more robust on new data.
- **Remark:** This specific variant is called the **1-norm soft margin classifier**, because the penalty (or regularization) term is a simple sum of the  $\xi_i$ .
- In a **2-norm soft margin classifier** the penalty term is  $\frac{\nu}{2} \sum_{i=1}^n \xi_i^2$ . (not considered here)

# Soft Margin Classifiers: Dual Problem

- In order to arrive at the dual problem, we proceed as before.  
The **Lagrange function** of the 1-norm soft margin classifier is

$$\mathcal{L}(\vec{a}^\circ, \vec{\zeta}, \vec{b}) = \frac{1}{2} \vec{a}^\top \vec{a} - \sum_{i=1}^n b_i (y_i \vec{a}^{\circ\top} \vec{x}^\circ - 1 + \zeta_i) + \nu \sum_{i=1}^n \zeta_i.$$

- First we form the gradient w.r.t.  $\vec{a}$  and set it equal to  $\vec{0}$ :

$$\nabla_{\vec{a}} \mathcal{L} = \vec{a} - \sum_{i=1}^n b_i y_i \vec{x} \stackrel{!}{=} \vec{0} \quad \Leftrightarrow \quad \vec{a}^* = \sum_{i=1}^n b_i^* y_i \vec{x}^*.$$

- Next we form the gradients w.r.t. the  $\zeta_i$  and set them equal to 0:

$$\frac{\partial \mathcal{L}}{\partial \zeta_i} = \nu - b_i \stackrel{!}{=} 0 \quad \Leftrightarrow \quad \zeta_i^* (b_i^* - \nu) = 0.$$

Note that we cannot conclude that always  $b_i = \nu$ , because  $\zeta_i$  **is constrained**: it has to be non-negative, that is, we have  $\forall i; 1 \leq i \leq n: \zeta_i \geq 0$ .

Therefore, if this constraint is inactive (i.e., if  $\zeta_i > 0$ ), then it must be  $b_i = \nu$ . However, if this constraint is active (i.e., if  $\zeta_i = 0$ ), then it may be  $b_i \neq \nu$ .

# Soft Margin Classifiers: Dual Problem

- Substituting the result back into the Lagrange function, we obtain:

$$\begin{aligned}
 \mathcal{L}(\vec{b}) &= \frac{1}{2} \left( \sum_{i=1}^n b_i y_i \vec{x} \right)^\top \left( \sum_{i=1}^n b_i y_i \vec{x} \right) + \nu \sum_{i=1}^n \xi_i - \sum_{i=1}^n b_i \left( y_i \left( \sum_{j=1}^n b_j y_j \vec{x} \right)^\top \vec{x} - 1 + \xi_i \right) \\
 &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n b_i y_i \vec{x}^\top \vec{x} y_j b_j + \nu \sum_{i=1}^n \xi_i - \sum_{i=1}^n \sum_{j=1}^n b_i y_i \vec{x}^\top \vec{x} y_j b_j + \sum_{i=1}^n b_i - \sum_{i=1}^n b_i \xi_i \\
 &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n b_i y_i \vec{x}^\top \vec{x} y_j b_j + \sum_{i=1}^n \underbrace{\xi_i (\nu - b_i)}_{=0} + \sum_{i=1}^n b_i \\
 &= \sum_{i=1}^n b_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n b_i y_i \vec{x}^\top \vec{x} y_j b_j \quad (\odot \text{ is the so-called Hadamard product}) \\
 &= \vec{1}_n^\top \vec{b} - \frac{1}{2} \vec{b}^\top \text{diag}(\vec{y}) \mathbf{G} \text{diag}(\vec{y}) \vec{b} \quad \left[ = \vec{1}_n^\top \vec{b} - \frac{1}{2} (\vec{b} \odot \vec{y})^\top \mathbf{G} (\vec{y} \odot \vec{b}) \right],
 \end{aligned}$$

- Note that this **reduced Lagrange function** is exactly the same as the one for the hard margin classifier derived before.

The only difference is the term  $\sum_{i=1}^n \xi_i (\nu - b_i)$ , which gets eliminated by the Karush–Kuhn–Tucker condition  $\xi_i^* (b_i^* - \nu) = 0$ .

# Soft Margin Classifiers: Dual Problem

- The dual optimization problem of the **soft margin classifier** is:

$$\begin{aligned} &\text{minimize} && \frac{1}{2} \vec{b}^\top \text{diag}(\vec{y}) \mathbf{G} \text{diag}(\vec{y}) \vec{b} - \vec{1}_n^\top \vec{b} \\ &\text{subject to} && \forall i; 1 \leq i \leq n: 0 \leq b_i \leq \nu \quad \text{and} \quad \vec{b}^\top \vec{y} = 0. \end{aligned}$$

The only difference to the hard margin classifier is the upper bound on the  $b_i$ .

- However, the Karush–Kuhn–Tucker conditions change to

$$\forall i; 1 \leq i \leq n: \quad (b_i^* - \nu) \zeta_i^* = 0 \quad \text{and} \quad b_i^* (y_i \vec{a}^{o* \top} \vec{x}_i^o - 1 + \zeta_i^*) = 0.$$

- With the exception of some rare degenerate cases it is:

$\hat{b}_i = 0$ : data point  $\vec{x}_i$  is on the correct side and outside the margin,

$0 < \hat{b}_i < \nu$ : data point  $\vec{x}_i$  is on the margin boundary (a **support vector**),

$\hat{b}_i = \nu$ : data point  $\vec{x}_i$  is inside the margin or even on the wrong side.

- This shows again: the lower  $\nu$  is, the more dual parameters  $\hat{b}_i$  get fixed to  $\nu$ , and hence the more points are inside the margin or even wrongly classified.

# Soft Margin Classifiers: Dual Problem

- The relationship of the **dual** to the **primal parameters** is unchanged:

$$\hat{\vec{a}} = \sum_{i=1}^n \hat{b}_i y_i \vec{x}_i.$$

With this formula the primal parameters can be computed from the dual.

- The value of  $\hat{a}_0$  may be found via a Karush–Kuhn–Tucker condition

$$\hat{b}_i (y_i \hat{\vec{a}}^\top \vec{x}_i - \hat{a}_0 - 1) = 0 \quad \text{for some } i \text{ with } 0 < \hat{b}_i < \nu,$$

(note that for such a  $\hat{b}_i$  the slack variable  $\hat{\zeta}_i = 0$  due to  $(\hat{b}_i - \nu)\hat{\zeta}_i = 0$ )  
or as a weighted average of the  $\hat{b}_i$ :

$$\hat{a}_0 = \sum_{i=1}^n \hat{b}_i (\nu - \hat{b}_i) (y_i - \hat{\vec{a}}^\top \vec{x}_i) / \sum_{i=1}^n \hat{b}_i (\nu - \hat{b}_i).$$

- Note that **any** non-negative value of  $\nu$  ensures that there is a solution, regardless of whether the data are actually linearly separable or not.

(This can be seen from the fact that  $\vec{a} = \vec{0}$  can always be chosen as a feasible (though not optimal) solution of the original problem — although this is not a feasible solution for a hard margin. If a feasible solution exists, some optimal solution must exist as well.)



# Mangasarian–Musicant Variant and Its Solution

- A slight variation of the optimization problem of a **soft margin classifier** is [Mangasarian and Musicant 1999]

$$\text{minimize} \quad \frac{1}{2} \vec{a}^{\circ\top} \vec{a}^{\circ} + \nu \sum_{i=1}^n \zeta_i$$

$$\text{subject to} \quad \forall i; 1 \leq i \leq n : \quad y_i \vec{a}^{\circ\top} \vec{x}_i^{\circ} \geq 1 - \zeta_i \quad \text{and} \quad \zeta_i \geq 0.$$

Note that the only difference is the additional term  $\frac{1}{2} a_0^2$  in the objective.

- This variant has the advantage that it can be solved by the method of **successive over-relaxation (SOR)**, which allows for a very brief algorithm.
- The dual optimization problem of the Mangasarian–Musicant variant is:

$$\text{minimize} \quad \frac{1}{2} \vec{b}^{\top} \text{diag}(\vec{y}) \mathbf{G}^{\circ} \text{diag}(\vec{y}) \vec{b} - \vec{1}_n^{\top} \vec{b}$$

$$\text{subject to} \quad \forall i; 1 \leq i \leq n : \quad 0 \leq b_i \leq \nu \quad \text{and} \quad \vec{b}^{\top} \vec{y} = 0.$$

- This variant has a simpler expression for  $\hat{a}_0$ :  $\hat{a}_0 = \sum_{i=1}^n \hat{b}_i y_i$ .

- The prediction function is also simpler: 
$$f(\vec{x}) = \sum_{i=1}^n \hat{b}_i y_i \vec{x}_i^{\circ\top} \vec{x}^{\circ}.$$

# Mangasarian–Musicant Variant and Its Solution

- The Mangasarian–Musicant variant can be solved by a **relaxation procedure**:

- $\mathbf{M}^\circ \leftarrow \text{diag}(\vec{y}) \mathbf{G}^\circ \text{diag}(\vec{y})$       and       $\vec{b} \leftarrow \vec{0}$ .

- Repeat the relaxation replacement, for  $i = 1, \dots, n$ :

$$b_i \leftarrow \max\left(0, \min\left(\nu, b_i - \frac{\omega}{\mathbf{M}_{ii}^\circ} \left(\sum_{j=1}^n \mathbf{M}_{ij}^\circ b_j - 1\right)\right)\right),$$

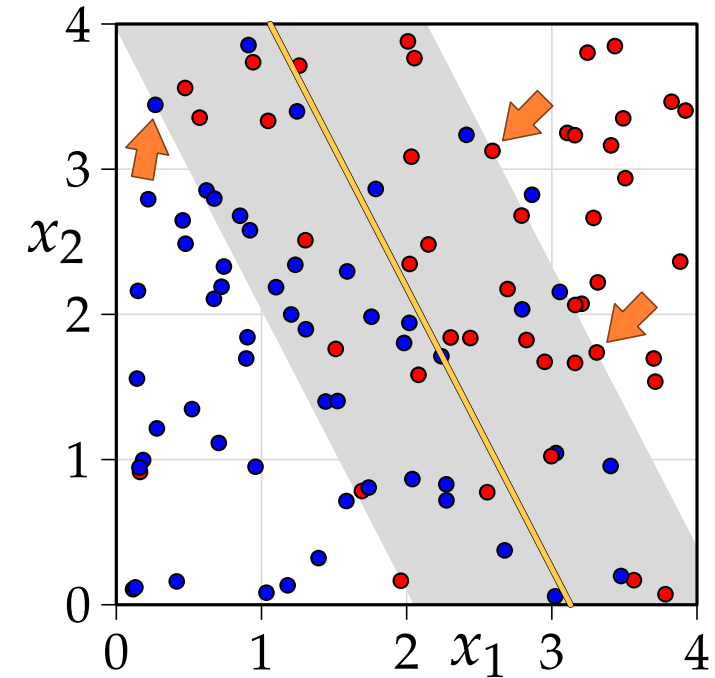
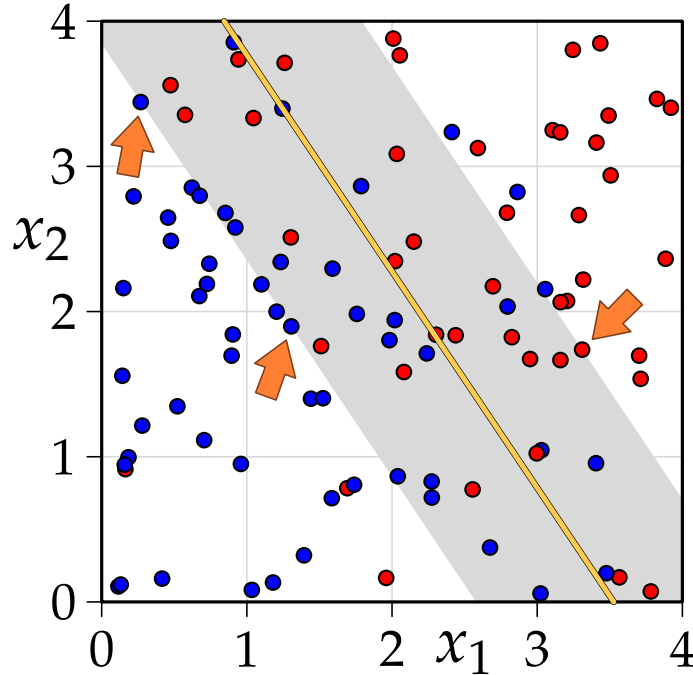
where  $\omega$  is the over-relaxation parameter.

It should be chosen as  $0 < \omega < 2$ , in the absence of better ideas:  $\omega = 1.3$ .

- Terminate if  $|\vec{b}^{(\text{new})} - \vec{b}^{(\text{old})}| \leq \theta$ , where typically  $\theta = 10^{-3}$  or  $\theta = 10^{-4}$ , or after a predefined (maximum) number of iterations.
- The order in which the  $b_i$  are relaxed has an influence on the convergence speed.
  - In an outer loop, first relax all parameters  $b_i$ , regardless of their value.
  - In an inner loop, relax only the non-zero parameters, i.e.  $b_i > 0$ .
  - Randomize the order in which the parameters are relaxed.

# Soft Margin Classifier: Example

- Data set with two classes, 100 points:
  - 55 data points belong to the **blue** class,
  - 45 data points belong to the **red** class.
- Soft margin classifiers trained with successive over-relaxation (Mangasarian–Musicant variant).



- Decision boundary (yellow line), margins (gray stripes), and support vectors (orange arrows).
- Regularization parameter  $\nu$ :  
top:  $\nu = 1$ , left:  $\nu = 100$ .  
(Note how the results differ!)

# Finding Soft Margin Classifiers in Practice

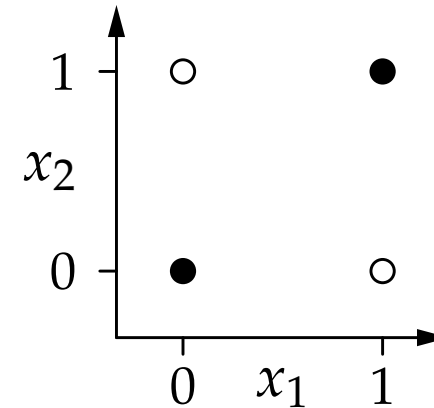
- The solution of the Mangasarian–Musicant variant is often identical to the solution of the standard 1-norm soft margin problem (although  $\nu$  differs) and almost never significantly different.
- However, it is **not the best** (meaning: fastest) approach to find a solution. It merely has the advantage of allowing for a fairly brief algorithm. It works well for small (and medium-sized) problems, but not for large ones.
- A fairly simple stochastic gradient descent approach based on the **hinge loss** was suggested by [David Forsyth 2018], which works well for small problems.
- An alternative are so-called **augmented Lagrangian methods** or **decomposition methods** that decompose the large quadratic programming problem into a set of smaller quadratic programming problems.
- A very nice implementation of a decomposition strategy is provided by SVM<sup>light</sup> [Thorsten Joachims 1999], while SVM<sup>perf</sup> [Thorsten Joachims 2006] uses a cutting-plane algorithm.

[https://www.cs.cornell.edu/people/tj/svm\\_light/](https://www.cs.cornell.edu/people/tj/svm_light/)

# Linear Separability: Limitations

The biimplication problem  $x_1 \leftrightarrow x_2$ : There is no separating line.

$x_1$	$x_2$	$y$
0	0	1
1	0	0
0	1	0
1	1	1



Formal proof by *reductio ad absurdum*:

$$\text{since } (0,0) \mapsto 1: \quad 0 \geq \theta, \quad (1)$$

$$\text{since } (1,0) \mapsto 0: \quad w_1 < \theta, \quad (2)$$

$$\text{since } (0,1) \mapsto 0: \quad w_2 < \theta, \quad (3)$$

$$\text{since } (1,1) \mapsto 1: \quad w_1 + w_2 \geq \theta. \quad (4)$$

(2) and (3):  $w_1 + w_2 < 2\theta$ . With (4):  $2\theta > \theta$ , or  $\theta > 0$ . Contradiction to (1).

# Linear Separability: Limitations

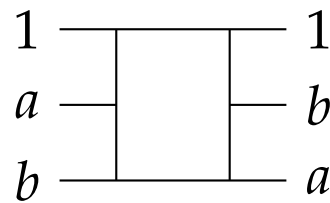
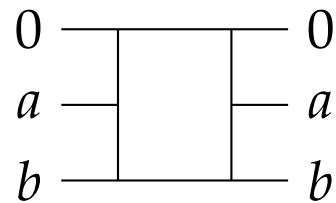
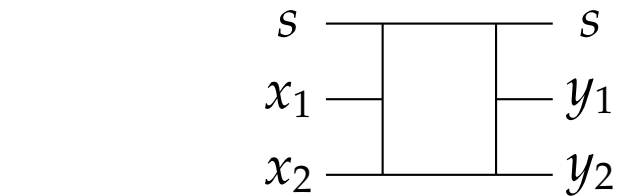
**Definition:** A set of points in a Euclidean space is called **convex** if it is non-empty and connected (that is, if it is a *region*) and for every pair of points in it every point on the straight line segment connecting the points of the pair is also in the set.

**Definition:** The **convex hull** of a set  $X$  of points in a Euclidean space is the smallest convex set of points that contains  $X$ . Alternatively, the **convex hull** of a set  $X$  of points is the intersection of all convex sets that contain  $X$ .

**Theorem:** Two sets of points in a Euclidean space are **linearly separable** if and only if their convex hulls are disjoint (that is, have no point in common).

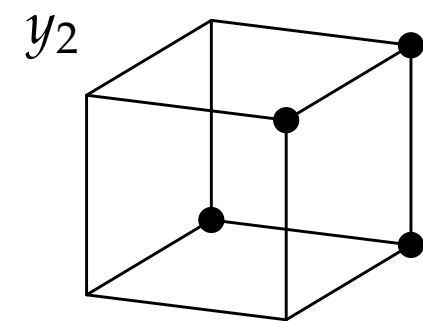
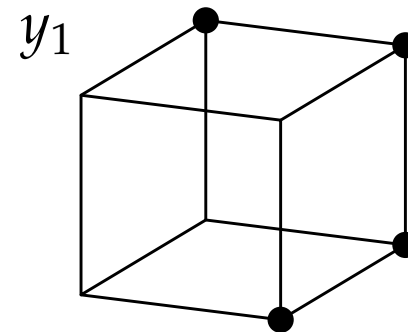
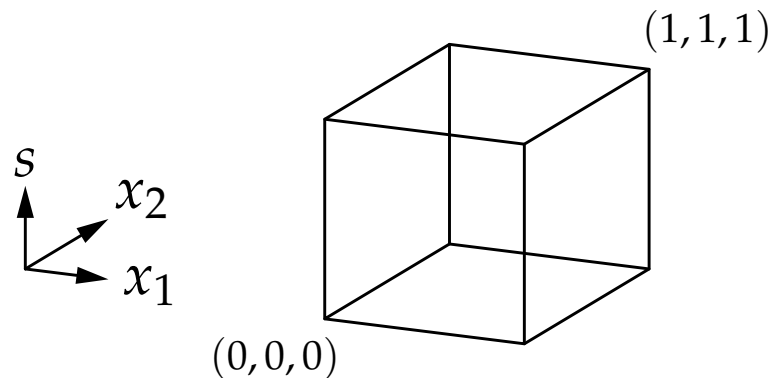
- For the biimplication problem, the convex hulls are the diagonal line segments.
- They share their intersection point and are thus not disjoint.
- Therefore the biimplication is not linearly separable.

# Limitations of Linear Separability: Fredkin Gate



$s$	0	0	0	0	1	1	1	1
$x_1$	0	0	1	1	0	0	1	1
$x_2$	0	1	0	1	0	1	0	1
$y_1$	0	0	1	1	0	1	0	1
$y_2$	0	1	0	1	0	0	1	1

$s$  is a “switch” variable, which determines whether  $x_1$  and  $x_2$  are passed through ( $y_1 = x_1$  and  $y_2 = x_2$  for  $s = 0$ ) or are swapped ( $y_1 = x_2$  and  $y_2 = x_1$  for  $s = 1$ ).



# Limitations of Linear Separability: Fredkin Gate

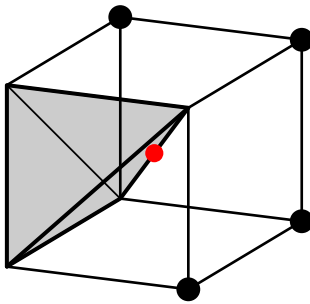
- The **Fredkin gate** (after Edward Fredkin \*1934) or **controlled swap gate** (CSWAP) is a computational circuit that is used in **conservative logic** and **reversible computing**.
- Conservative logic is a model of computation that explicitly reflects the physical properties of computation, like the reversibility of the dynamical laws and the conservation of certain quantities (e.g. energy) [Fredkin and Toffoli 1982].
- The Fredkin gate is **reversible** in the sense that the inputs can be computed as functions of the outputs in the same way in which the outputs can be computed as functions of the inputs (no information loss, no entropy gain).
- The Fredkin gate is **universal** in the sense that all Boolean functions can be computed using only Fredkin gates.
- Note that both outputs,  $y_1$  and  $y_2$  are **not linearly separable**, because the convex hull of the points mapped to 0 and the convex hull of the points mapped to 1 share the point in the center of the cube.



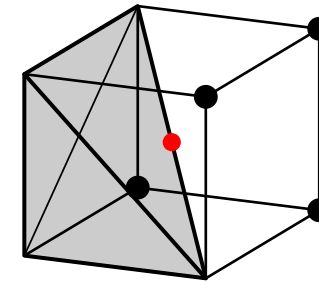
# Reminder: Convex Hull Theorem

**Theorem:** Two sets of points in a Euclidean space are **linearly separable** if and only if their convex hulls are disjoint (that is, have no point in common).

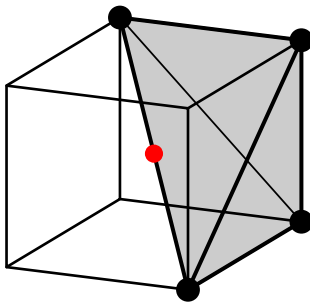
Both outputs  $y_1$  and  $y_2$  of a Fredkin gate are not linearly separable:



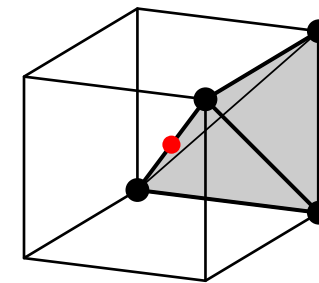
Convex hull of points with  $y_1 = 0$



Convex hull of points with  $y_2 = 0$



Convex hull of points with  $y_1 = 1$



Convex hull of points with  $y_2 = 1$

# Linear Separability: Limitations

**Total number and number of linearly separable Boolean functions**  
(On-Line Encyclopedia of Integer Sequences, [oeis.org](http://oeis.org), A001146 and A000609):

inputs	Boolean functions	linearly separable functions
1	4	4
2	16	14
3	256	104
4	65,536	1,882
5	4,294,967,296	94,572
6	18,446,744,073,709,551,616	15,028,134
$n$	$2^{(2^n)}$	no general formula known

- For many inputs almost no functions are linearly separable.
- To overcome this situation, several approaches are possible.

# Non-Linear Regression and Classification

- The fundamental principle of support vector machines can be formulated as:
  - If the task is regression, linear regression suffices.
  - If the task is classification, linear classification suffices.
- But how can linear methods always be sufficient?
  - Not all regression functions are linear.  
(e.g. polynomials, exponential functions, logistic function, sine/cosine etc.)
  - Not all classification problems are linearly separable.  
(e.g. biimplication, inside a circle versus outside, curved class boundaries etc.)
- The first core trick consists in **mapping the data into another space**, such that the problem becomes linear(ly separable) in this image space.
- We explore first how a problem can be made linear by such a mapping.  
(Mainly by looking at several simple examples for regression and classification.)
- Next we study how one can do this **without mapping the data explicitly**.

# Non-Linear Regression: Polynomials

- **Quadratic regression** can be reduced to bivariate linear regression by setting  $y_i = x_i^2, i = 1, \dots, n$ , and solving the resulting system of normal equations:

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

$$\left( \sum_{i=1}^n x_i \right) a + \left( \sum_{i=1}^n x_i^2 \right) b + \left( \sum_{i=1}^n \underbrace{x_i^3}_{=x_i y_i} \right) c = \sum_{i=1}^n z_i x_i$$

$$\left( \sum_{i=1}^n \underbrace{x_i^2}_{=y_i} \right) a + \left( \sum_{i=1}^n \underbrace{x_i^3}_{=x_i y_i} \right) b + \left( \sum_{i=1}^n \underbrace{x_i^4}_{=y_i^2} \right) c = \sum_{i=1}^n \underbrace{z_i x_i^2}_{=z_i y_i}$$

- **Multivariate polynomial regression** can be reduced to multivariate linear regression by adding the needed monomials (i.e. power products, e.g.  $x_1 x_2^3 x_3^2$ ) to the regressors / inputs. Other functions of regressors (e.g. exponential, logarithm, sine/cosine etc.) may be treated in an analogous fashion.

# Regression: Generalization, Logistic Regression

**Generalization of regression to more general functions.**

Simple example:  $y = ax^b$

Idea: Find a **transformation to the linear (or polynomial) case**.

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

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

---

**Special case: Logistic Function**

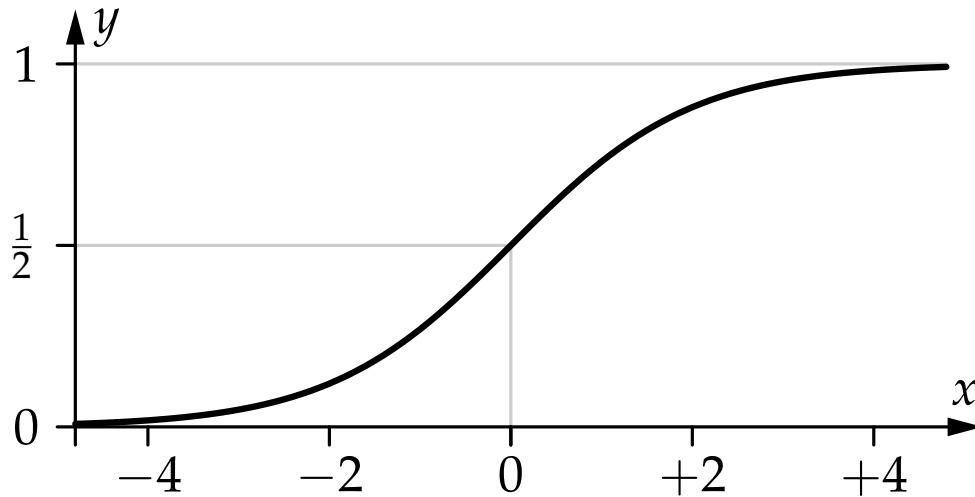
(with  $a_0 = \vec{a}^\top \vec{x}_0$ )

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

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

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

# Reminder: Logistic Function



Logistic function (univariate):

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

Special case  $y_{\max} = a = 1, x_0 = 0$ :

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

## Application areas of the logistic function:

- Can be used to describe **saturation processes** (growth processes with finite capacity/finite resources  $y_{\max}$ ).

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

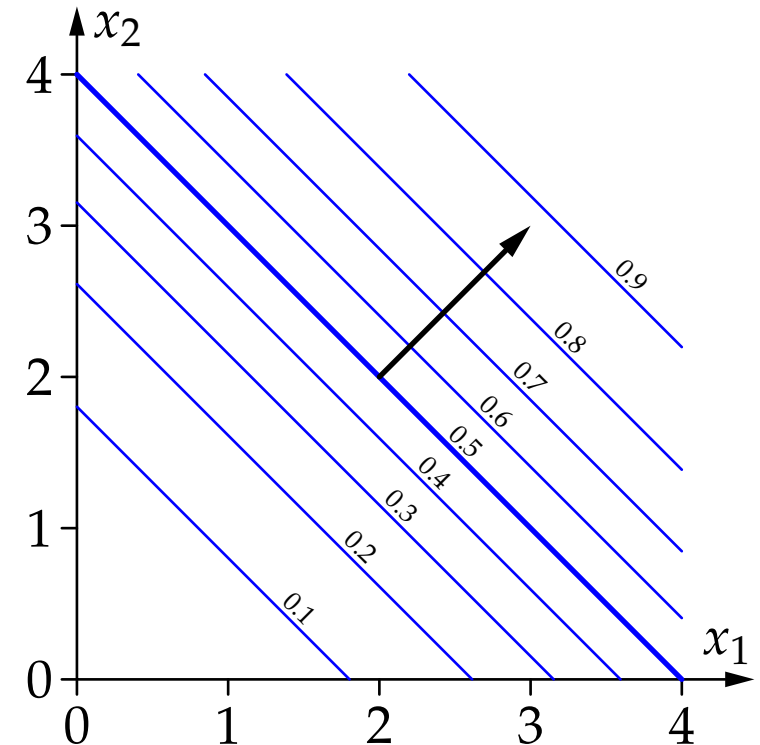
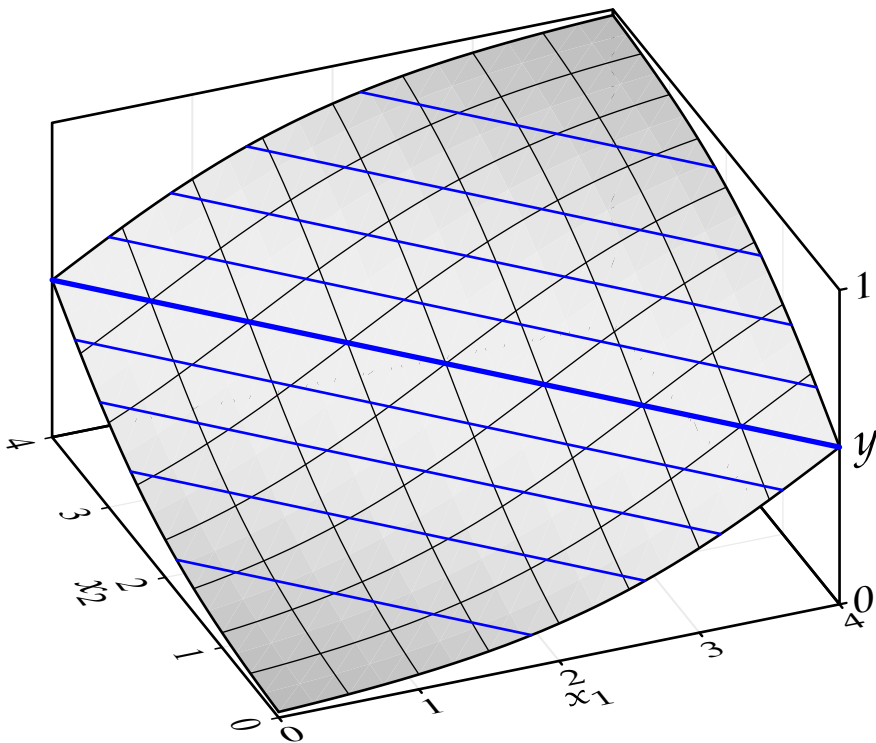
$$f'(x) = k \cdot f(x) \cdot (y_{\max} - f(x)) \quad (\text{yields } a = ky_{\max})$$

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

# Reminder: Logistic Function

Example: two-dimensional logistic function

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

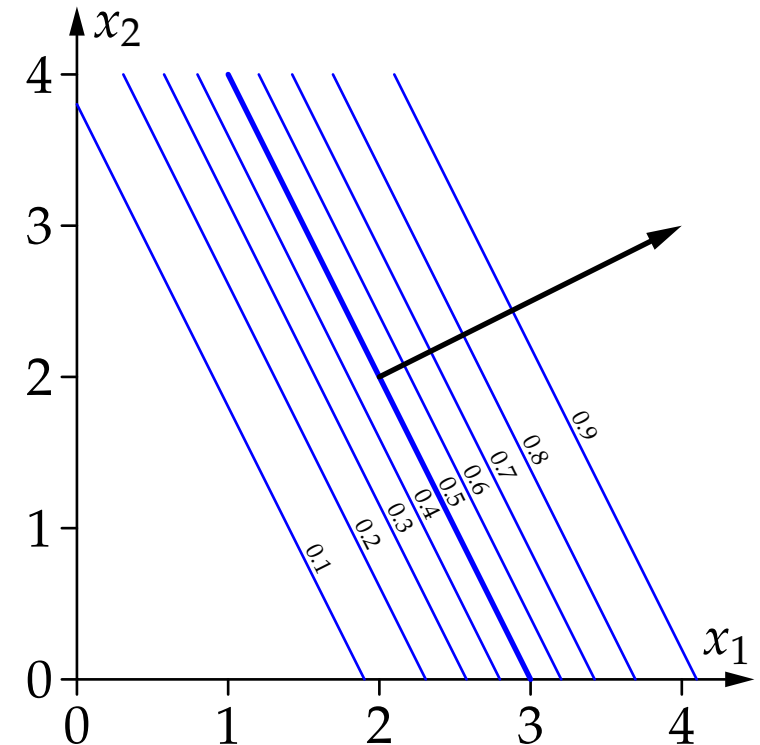
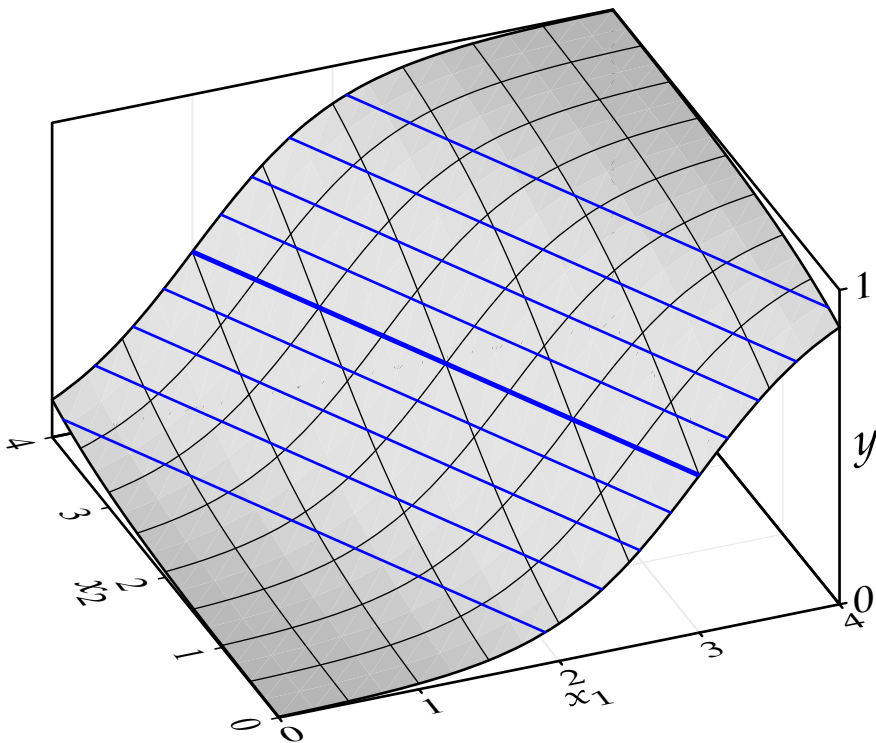


The “contour lines” of the logistic function are parallel lines/hyperplanes.

# Reminder: Logistic Function

Example: two-dimensional logistic function

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



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



# Univariate Logistic Regression: Example

Data points:

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

Apply the logit transform

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

Transformed data points:

(for linear regression)

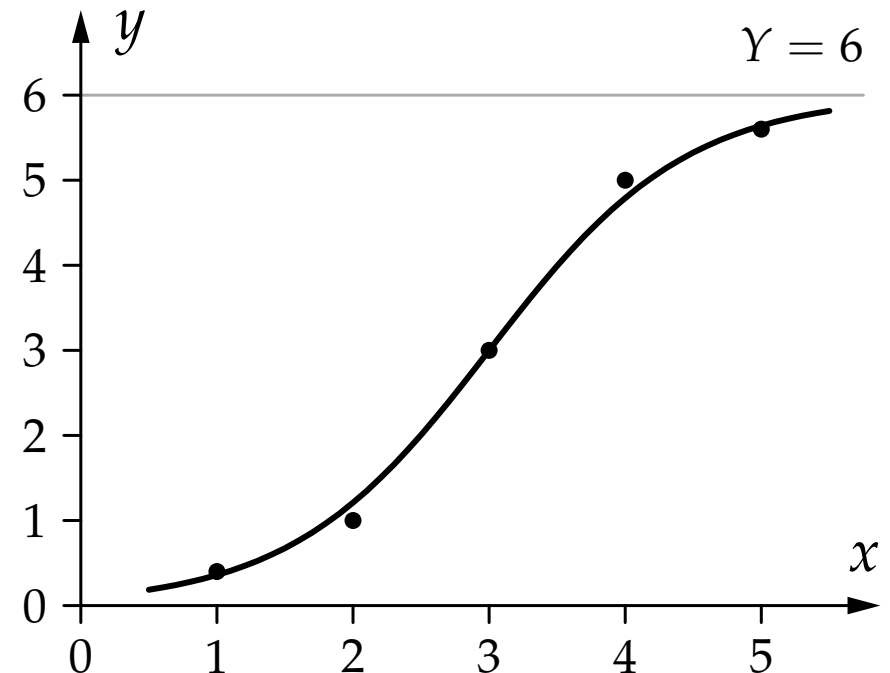
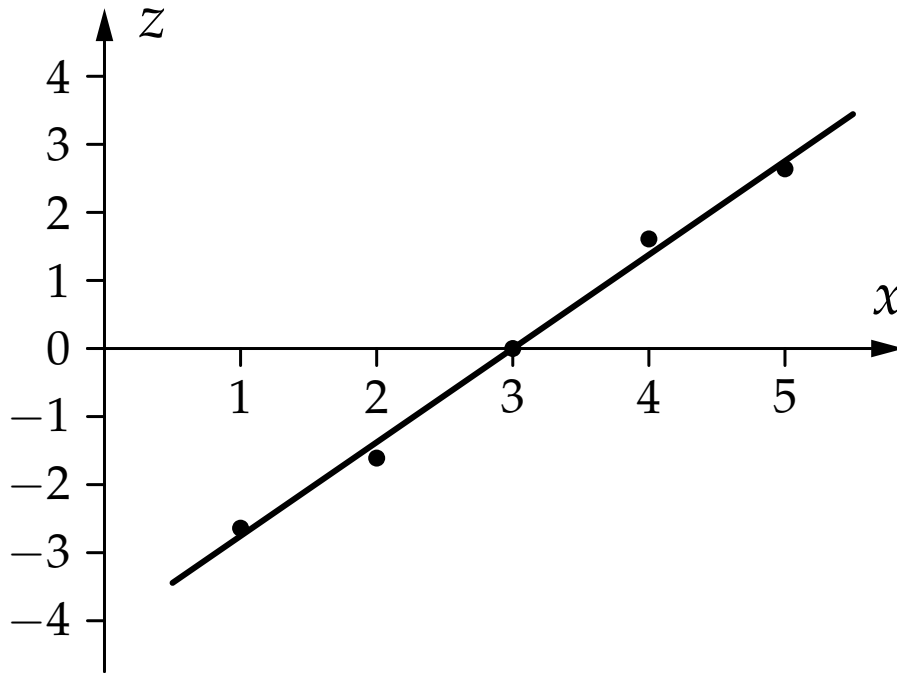
$x$	1	2	3	4	5
$z$	-2.64	-1.61	0.00	1.61	2.64

The resulting regression line and therefore the desired function are

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

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

# Univariate Logistic Regression: Example

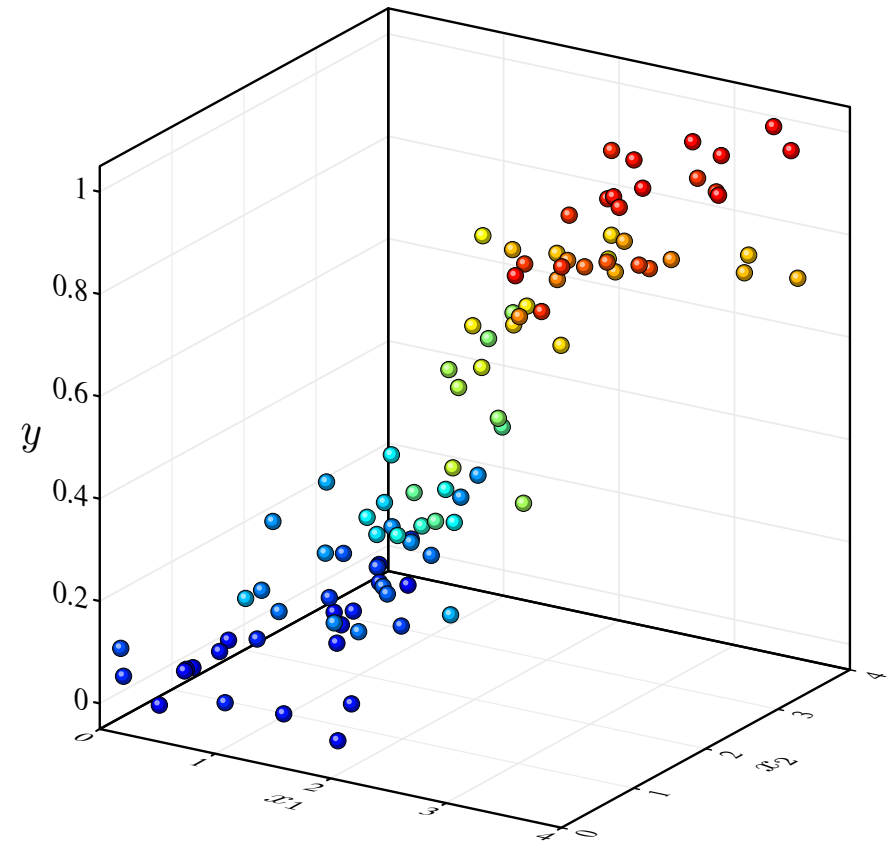
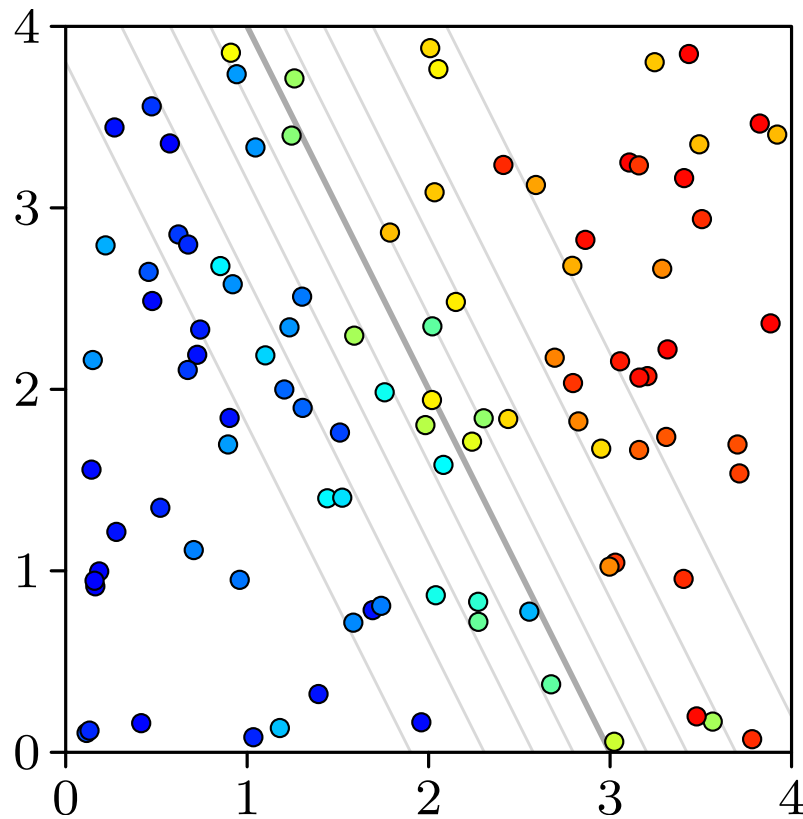


The resulting regression line and therefore the desired function are

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

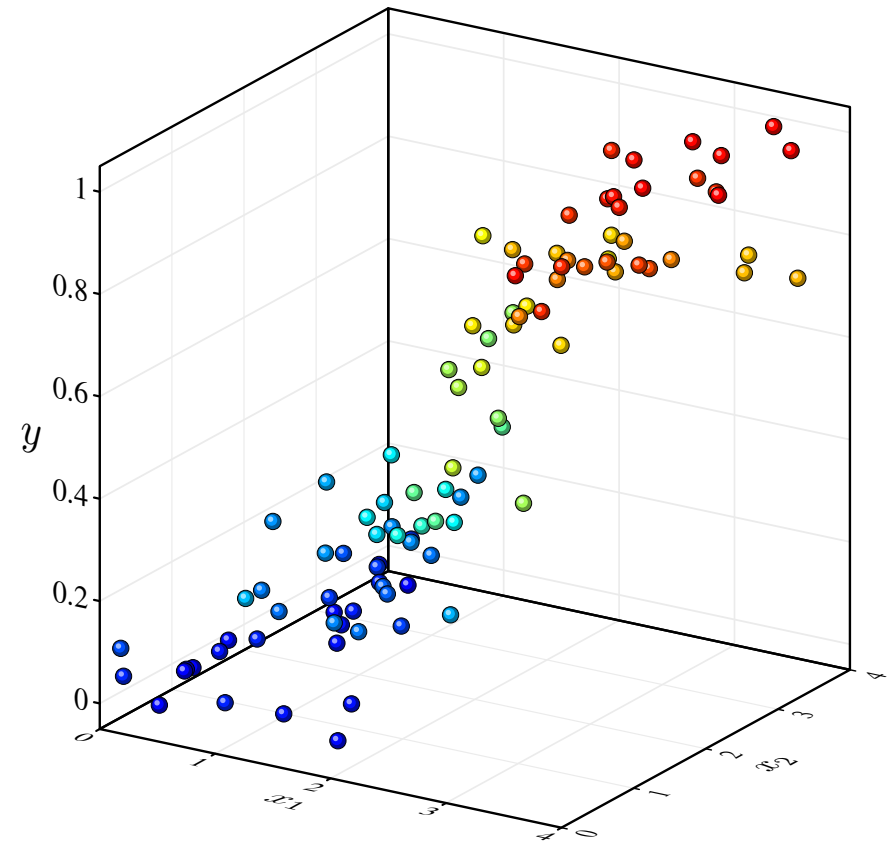
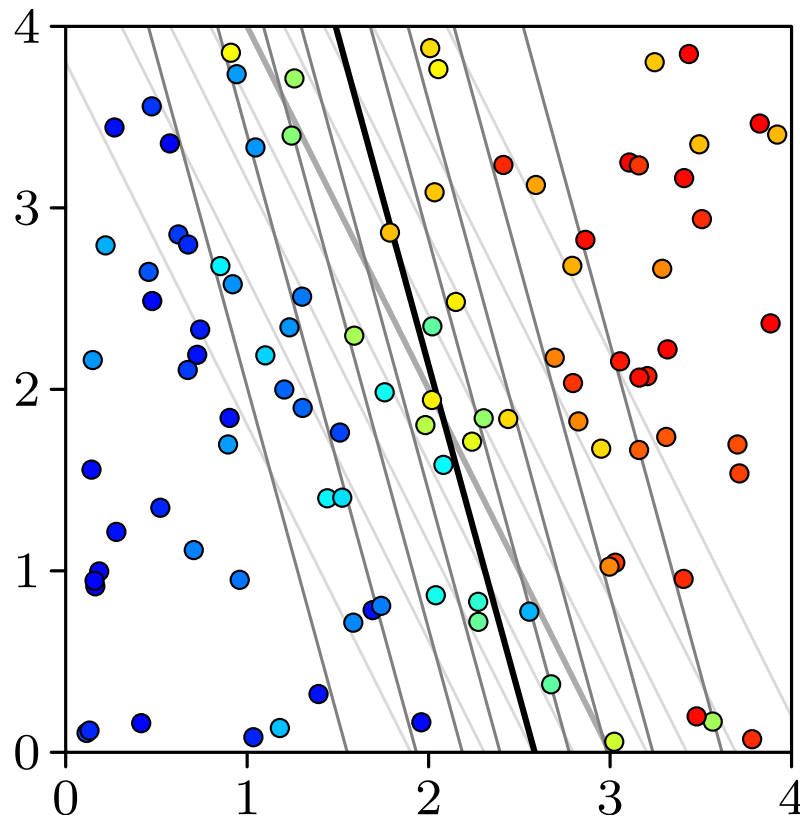
**Attention:** Note that the error is minimized only in the transformed space!  
Therefore the function may not be optimal in the original space!

# Bivariate Logistic Regression: Example



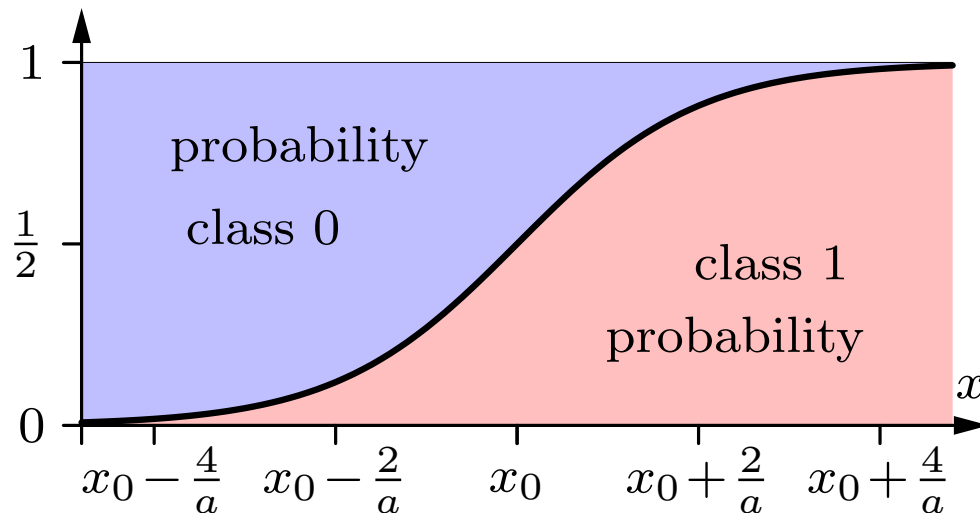
- Example data were drawn from a logistic function and noise was added. (The gray “contour lines” show the ideal logistic function.)
- Reconstructing the logistic function can be reduced to a multivariate linear regression by applying a logit transform to the  $y$ -values of the data points.

# Bivariate Logistic Regression: Example



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

# Reminder: Logistic Classification



Logistic function with  $y_{\max} = 1$ :

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

Interpret the logistic function as the probability of one class.

- Conditional class probability is logistic function:

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

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

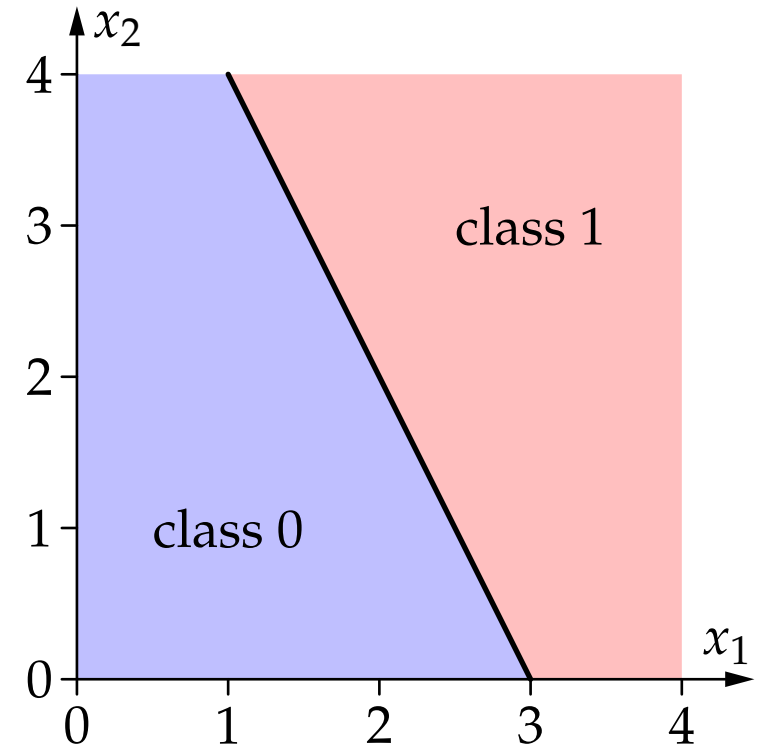
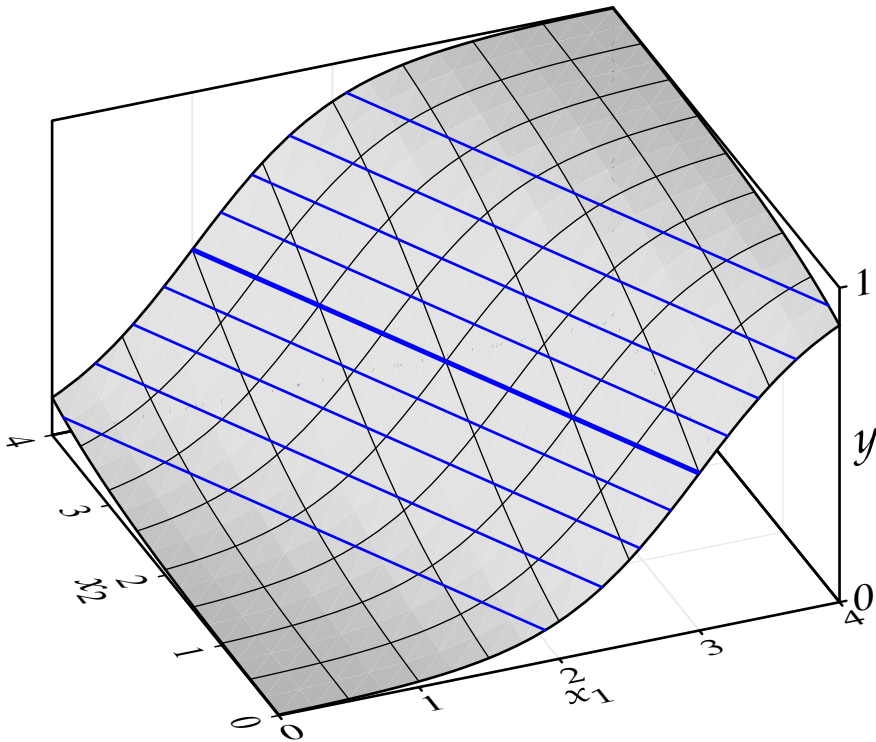
$$P(C = c_0 \mid \vec{X} = \vec{x}) = p_0(\vec{x}) = 1 - p(\vec{x}; \vec{a}^\circ).$$

- Classification rule:

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

$$\vec{a}^\circ = (a_0, a_1, \dots, a_m)^\top$$
$$\vec{x}^\circ = (1, x_1, \dots, x_m)^\top$$

# Reminder: Logistic Classification

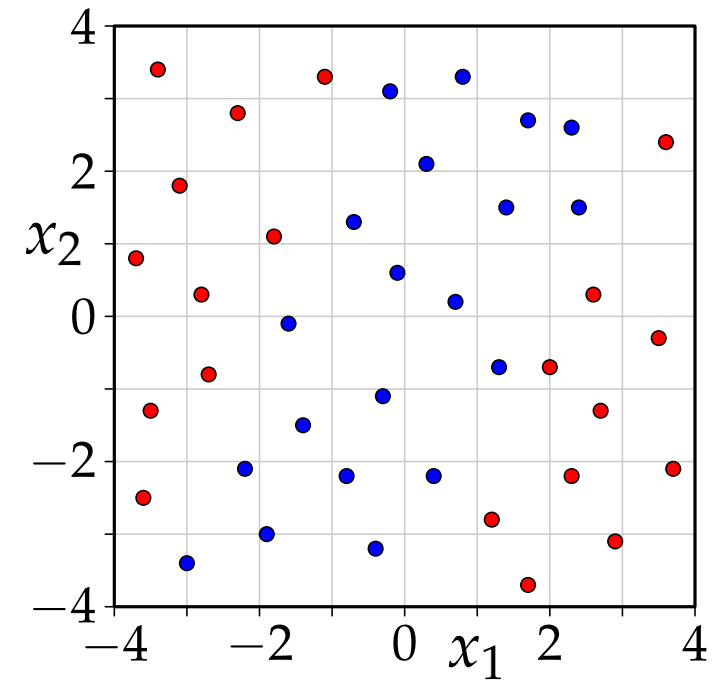


- Classes are separated by the “contour line”  $p(\vec{x}; \vec{a}^o) = \theta = 0.5$  (inflection line). Via the classification threshold  $\theta$  misclassification costs may be incorporated.
- The classification boundary is linear, therefore **linear classification**.
- For classification, using a logistic function does not yield non-linear properties.

# Non-Linear Classification: Stripes

- Simple data set with two classes:
  - 20 data points belong to the **red** class,
  - 20 data points belong to the **blue** class.
- These classes are **not linearly separable**:

There is no straight line such that all points of class **red** are on one side of the line and all points of class **blue** are on the other side.
- Idea: **Map points to a different space.**
- A proper mapping can render the images of the data points linearly separable.
  - Here: “Bend” the  $x_1$ - $x_2$ -plane, such that all red points end up above (or below) all blue points.
  - Only the added dimension may be needed to separate the classes (although showing all dimensions makes the idea clearer).

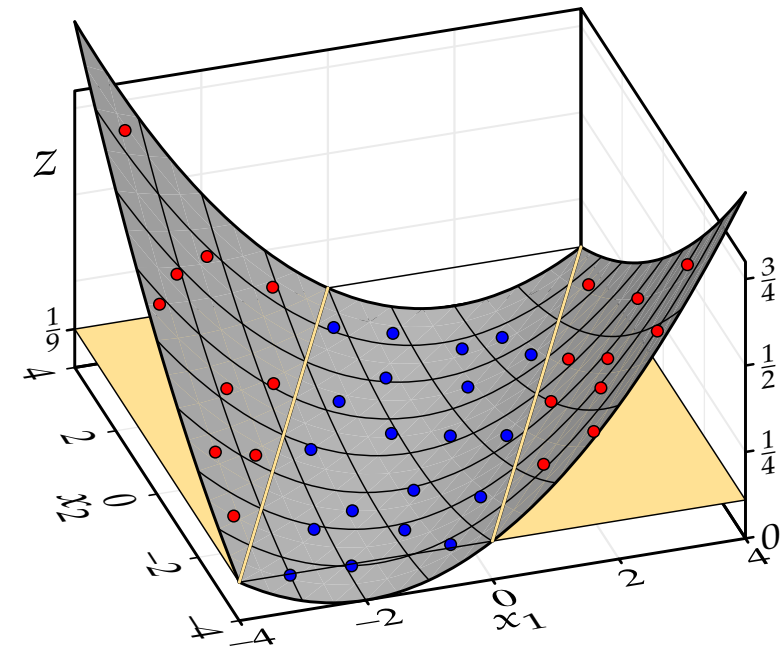
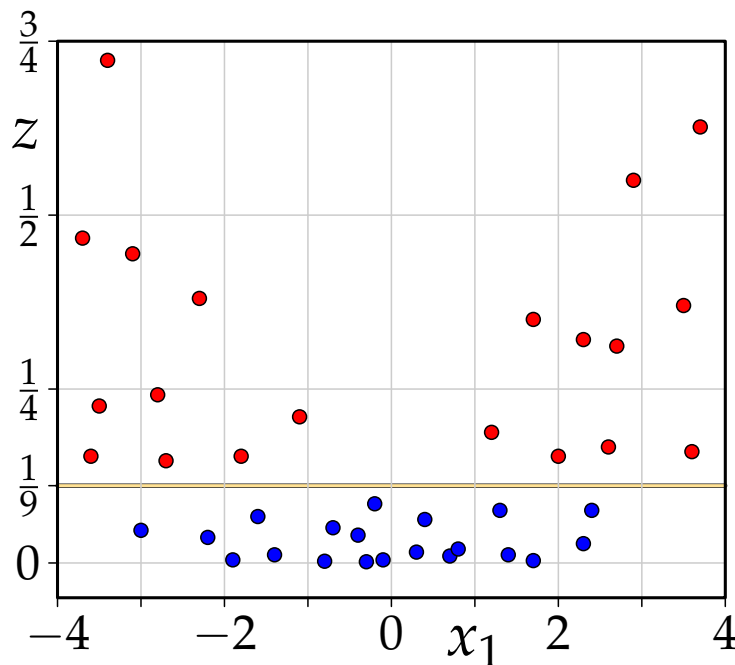


# Non-Linear Classification: Stripes

- For the mapping add a new dimension that is computed as, e.g.,

$$z = \left( \frac{x_2}{12} - \frac{x_1}{6} \right)^2$$

- This allows us to separate the classes with a horizontal plane (yellow).



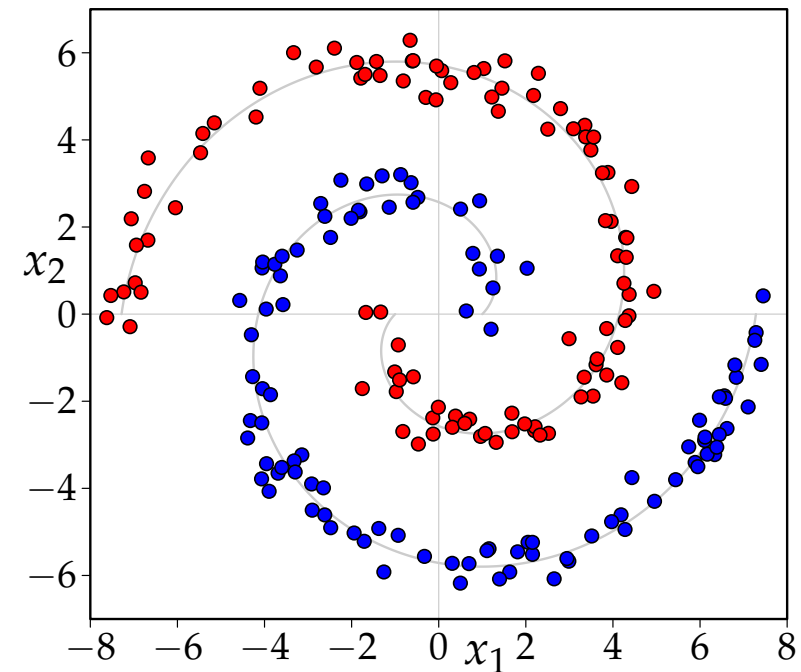
- For the classification, only the  $z$ -dimension is actually needed (hence only  $z$  may be the target space).
- Classify the data according to:
  - if  $z > \frac{1}{9}$ , assign the **red** class,
  - if  $z < \frac{1}{9}$ , assign the **blue** class.



# Non-Linear Classification: Archimedean Spirals

- More complex data set with two classes:
  - 100 data points belong to the **red** class,
  - 100 data points belong to the **blue** class.
- These classes are **not linearly separable**:

There is no straight line such that all points of class red are on one side of the line and all points of class blue are on the other side.
- Idea: **Map points to a different space.**
- A proper mapping can render the images of the data points linearly separable.
  - Here: Map to polar coordinates and handle the resulting stripes such that all red points end up above (or below) all blue points.
  - Only the added dimension may be needed to separate the classes (although showing all dimensions makes the idea clearer).

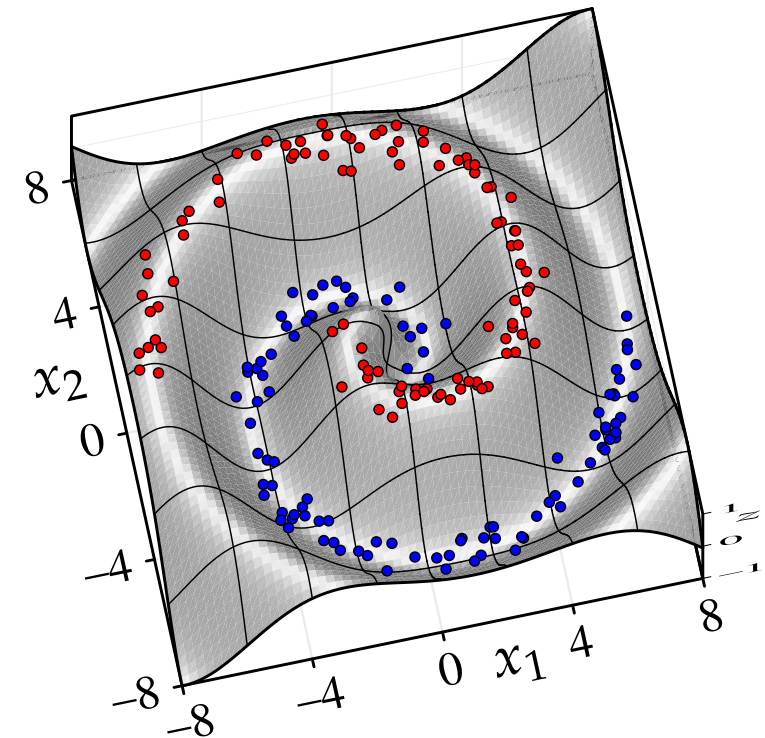
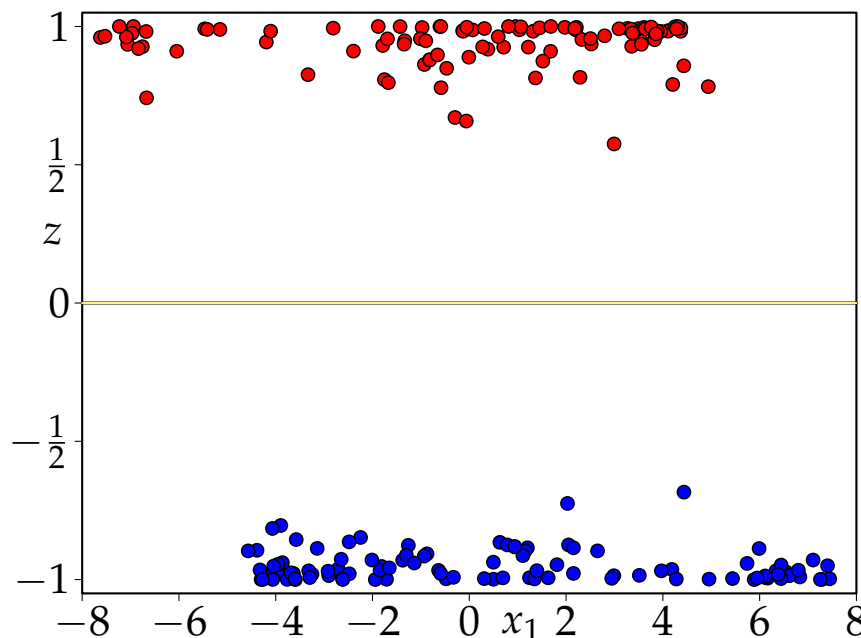


# Non-Linear Classification: Archimedean Spirals

- For the mapping add a new dimension that is computed as, e.g.,

$$z = 2 \sin^2 \left( \frac{r}{2} - \frac{\theta}{2} - \frac{\pi}{6} \right) - 1$$

where  $r = \text{hypot}(x_1, x_2) = \sqrt{x_1^2 + x_2^2}$   
and  $\theta = \text{atan2}(x_2, x_1)$ .



- For the classification, only the  $z$ -dimension is actually needed.
- Classify the data according to:
  - if  $z > 0$ , assign the **red** class,
  - if  $z < 0$ , assign the **blue** class.

# Kernel Functions

- In the following, **kernel functions** are used to **map data points implicitly** to some image space, in which they (hopefully) become linearly treatable.
- **Definition:** Let  $\mathcal{X}$  be some data space, e.g.,  $\mathcal{X} = \mathbb{R}^m$ .  
A **kernel function** or simply **kernel** is a function  $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  that satisfies

$$\forall \vec{x}_1, \vec{x}_2 \in X : \quad \kappa(\vec{x}_1, \vec{x}_2) = \kappa(\vec{x}_2, \vec{x}_1) \quad (\kappa \text{ is symmetric})$$

$$\forall \vec{x}_1, \vec{x}_2 \in X : \quad \kappa(\vec{x}_1, \vec{x}_2) \geq 0 \quad (\kappa \text{ is non-negative})$$

- In the image space, a linear model is constructed / trained.
- Since the (implicit) mapping to the image space may be non-linear, this **model may be non-linear in the original (data) space**.
- How well this works depends decisively on the choice of the kernel, since it has to capture all information about the non-linear structure of the data.
- Kernel functions are typically chosen to quantify the similarity between two objects  $\vec{x}_1$  and  $\vec{x}_2$  in  $\mathcal{X}$ : The higher  $\kappa(\vec{x}_1, \vec{x}_2)$ , the more similar the objects.

# Kernel Functions and Gram Matrices

- A matrix  $\mathbf{G}$  is called **Gram matrix** or **Gramian matrix** [Jørgen Pedersen Gram] if it is a matrix of pairwise scalar products of data points  $\vec{z}_1, \dots, \vec{z}_n \in \mathcal{Z}$ :

$$\mathbf{G} = \begin{bmatrix} \langle \vec{z}_1, \vec{z}_1 \rangle & \langle \vec{z}_1, \vec{z}_2 \rangle & \cdots & \langle \vec{z}_1, \vec{z}_n \rangle \\ \langle \vec{z}_2, \vec{z}_1 \rangle & \langle \vec{z}_2, \vec{z}_2 \rangle & \cdots & \langle \vec{z}_2, \vec{z}_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \vec{z}_n, \vec{z}_1 \rangle & \langle \vec{z}_n, \vec{z}_2 \rangle & \cdots & \langle \vec{z}_n, \vec{z}_n \rangle \end{bmatrix}$$

where  $\langle \cdot, \cdot \rangle$  denotes the scalar product of the space  $\mathcal{Z}$ .

- However, for support vector machines, a Gram matrix is computed with the help of a kernel function  $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  as

$$\mathbf{G} = \begin{bmatrix} \kappa(\vec{x}_1, \vec{x}_1) & \kappa(\vec{x}_1, \vec{x}_2) & \cdots & \kappa(\vec{x}_1, \vec{x}_n) \\ \kappa(\vec{x}_2, \vec{x}_1) & \kappa(\vec{x}_2, \vec{x}_2) & \cdots & \kappa(\vec{x}_2, \vec{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\vec{x}_n, \vec{x}_1) & \kappa(\vec{x}_n, \vec{x}_2) & \cdots & \kappa(\vec{x}_n, \vec{x}_n) \end{bmatrix}$$

- For this to be possible,  $\kappa$  cannot be just any kernel function whatsoever. It has to be what is commonly called a **Mercer kernel**. [James Mercer 1909]

# Mercer's Theorem and Mercer Kernels

- In functional analysis, **Mercer's theorem** [James Mercer 1909] states that a symmetric and positive semi-definite matrix can be represented as a sum of a convergent sequence of product functions.
- From Mercer's theorem it follows that a matrix  $\mathbf{G}$  that is computed with the help of a kernel function  $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  as

$$\mathbf{G} = \begin{bmatrix} \kappa(\vec{x}_1, \vec{x}_1) & \kappa(\vec{x}_1, \vec{x}_2) & \cdots & \kappa(\vec{x}_1, \vec{x}_n) \\ \kappa(\vec{x}_2, \vec{x}_1) & \kappa(\vec{x}_2, \vec{x}_2) & \cdots & \kappa(\vec{x}_2, \vec{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\vec{x}_n, \vec{x}_1) & \kappa(\vec{x}_n, \vec{x}_2) & \cdots & \kappa(\vec{x}_n, \vec{x}_n) \end{bmatrix}$$

Note that  $\mathbf{G}$  is symmetric, since  $\kappa$  is a kernel function and a kernel function is symmetric by definition.

is a Gram matrix (that is, it is a scalar product matrix in some space) if and only if  $\mathbf{G}$  is positive semi-definite.

- That is, if the (kernel) matrix  $\mathbf{G}$  is positive semi-definite, then there exists a function  $\varphi : \mathcal{X} \rightarrow \mathcal{Z}$  such that  $\kappa(\vec{x}, \vec{x}') = \langle \varphi(\vec{x}), \varphi(\vec{x}') \rangle$ .
- Note that  $\mathcal{Z}$  may have infinite dimension, but we do not consider this here, because it requires fairly sophisticated mathematical tools.

# Mercer's Theorem: Some Justification

- Suppose the matrix  $\mathbf{G}$  of pairwise kernel evaluations is positive definite.
- A symmetric positive definite matrix possesses an eigendecomposition

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

where the  $\lambda_i, i = 1, \dots, m$ , are the eigenvalues of  $\mathbf{G}$   
and the columns of  $\mathbf{R}$  are the (normalized) eigenvectors of  $\mathbf{G}$ .

- The eigenvalues of a symmetric positive definite matrix are all positive and its eigenvectors are orthonormal (that is,  $\mathbf{R}^{-1} = \mathbf{R}^\top$ ).
- As a consequence,  $\mathbf{G}$  can be written as  $\mathbf{G} = \mathbf{T}\mathbf{T}^\top$ , where

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

- Define  $\varphi(\vec{x}_i) = (\mathbf{R}_{i:} \operatorname{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_m}))^\top$ ,  
where  $\mathbf{R}_{i:}$  denotes the  $i$ -th row of the matrix  $\mathbf{R}$ .

With this definition it is  $\mathbf{G}_{ij} = \kappa(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i)^\top \varphi(\vec{x}_j)$ .

# Mercer's Theorem

- A full mathematical justification of **Mercer's theorem** requires the theory of **reproducing kernel Hilbert spaces**.
- We do not dive into these mathematical details here. They can be found in, e.g.:

pictures not available in online version

# Mercer Kernels and Basis Functions

**Definition:** Let  $\mathcal{X}$  be a data space, e.g.,  $\mathcal{X} = \mathbb{R}^m$ .

A **Mercer kernel** is a function  $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  that for all  $\vec{x}_1, \vec{x}_2$  satisfies

$$\kappa(\vec{x}_1, \vec{x}_2) = \langle \varphi(\vec{x}_1), \varphi(\vec{x}_2) \rangle,$$

where  $\varphi$  is a mapping from the data space  $\mathcal{X}$  to some space  $\mathcal{Z}$  and  $\langle \cdot, \cdot \rangle$  denotes the scalar product in the space  $\mathcal{Z}$ .

- The function  $\varphi$  is often called a **basis function**; the space  $\mathcal{Z}$  is often referred to as the **feature space**.
- We may say that we map our objects to a feature space using a basis function.
- Mathematical remark: For a Mercer kernel, a basis function  $\varphi$  can be written as a linear combination of eigenfunctions of the kernel  $\kappa$ .
- There are no restrictions on the dimensionality of the feature space  $\mathcal{Z}$ ; actually,  $\mathcal{Z}$  is potentially infinite dimensional.

Note that if this is the case,  $\varphi(\vec{x})$  may not even be explicitly representable.



# Mercer Kernels and the Kernel Trick

- We have seen before that a **linear model can be parameterized and trained using only scalar products** (dual parameters / dual form).
- We have also seen that a Mercer kernel can **express scalar products** in some target space **without explicitly mapping the data** to that space.
- Hence Mercer kernels allow us to **train a (linear) model in the image space**, without explicitly mapping the data points to that space.
- Since the (Mercer) kernel may represent a non-linear mapping, the model may be **non-linear in the original (data) space**.
- These three ingredients are what is usually referred to as the **kernel trick**:
  - implicitly mapping the data points to some image space,
  - computing scalar products in this image space via a kernel,
  - training a model using only these scalar products.
- The kernel trick allows us to **obtain non-linear models indirectly**.

# Some Commonly Used Kernel Functions

- **Linear Kernel** (scalar product in data space)  
 $\kappa(\vec{x}_1, \vec{x}_2) = \vec{x}_1^\top \vec{x}_2$  (linear separation in data space)
- **Normalized Linear Kernel** (cosine between vectors  $\vec{x}_1$  and  $\vec{x}_2$ )  
 $\kappa(\vec{x}_1, \vec{x}_2) = \frac{\vec{x}_1^\top \vec{x}_2}{\sqrt{(\vec{x}_1^\top \vec{x}_1)(\vec{x}_2^\top \vec{x}_2)}}$  ( $\sqrt{\vec{x}_i^\top \vec{x}_i}$ : length of vector  $\vec{x}_i$ ,  $i = 1, 2$ , hence vectors are scaled to length 1)
- **Polynomial Kernel** (symmetric polynomial function)  
 $\kappa(\vec{x}_1, \vec{x}_2) = (\alpha \vec{x}_1^\top \vec{x}_2 + \beta)^r$  (construct new features as products of original features)  
(often with  $\alpha = 1$ )
- **Gaussian Kernel**  
 $\kappa(\vec{x}_1, \vec{x}_2) = \exp\left(-\frac{1}{2}(\vec{x}_1 - \vec{x}_2)^\top \Sigma^{-1}(\vec{x}_1 - \vec{x}_2)\right)$  (general covariance matrix)  
 $\kappa(\vec{x}_1, \vec{x}_2) = \exp\left(-\frac{1}{2}\sum_{i=1}^m \frac{1}{\sigma_i^2}(x_{1i} - x_{2i})^2\right)$  (diagonal covariance matrix)  
 $\kappa(\vec{x}_1, \vec{x}_2) = \exp\left(-\frac{1}{2\sigma^2}(\vec{x}_1 - \vec{x}_2)^\top (\vec{x}_1 - \vec{x}_2)\right)$  (isotropic covariance matrix)  
(often with  $\frac{1}{2\sigma^2} = \gamma$ )
- **Laplace Kernel**  
 $\kappa(\vec{x}_1, \vec{x}_2) = \exp\left(-\frac{1}{\sigma}\sqrt{(\vec{x}_1 - \vec{x}_2)^\top (\vec{x}_1 - \vec{x}_2)}\right)$  (“steeper” than Gaussian kernel)

# Some Commonly Used Kernel Functions

## Kernel Functions for Structured Data

[Gärtner 2003]

- **Model-driven Kernels**

Based on models describing the input space,  
either constructed from background knowledge or learned from data.

- **Fisher Kernel** (gradient of log-likelihood w.r.t. parameters of a generative model)
- **Diffusion Kernel** (describe the local neighborhood of an instance)

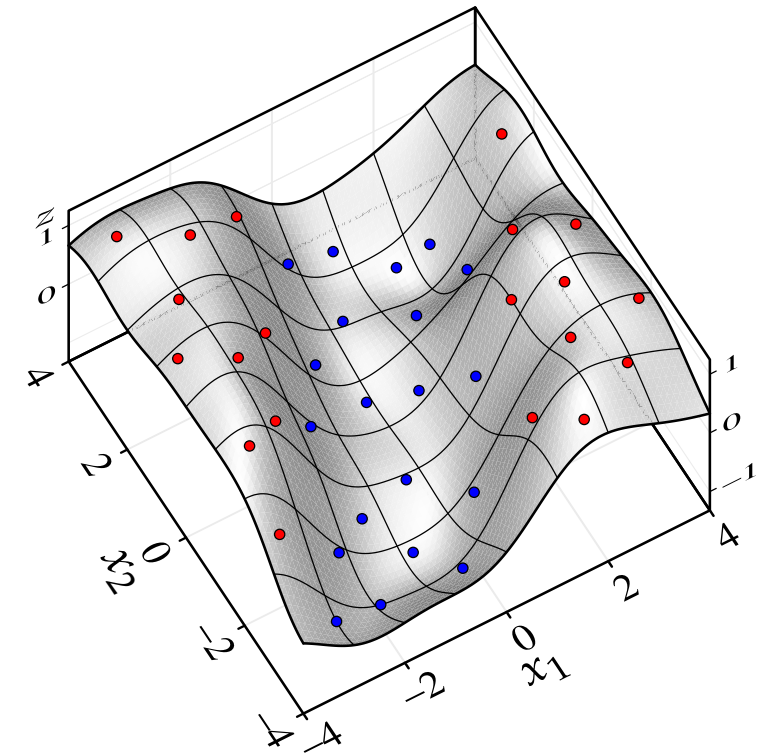
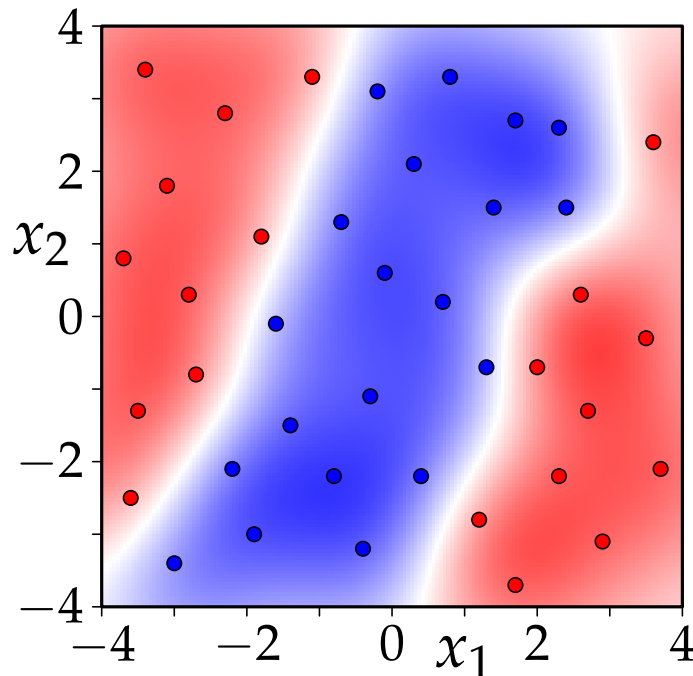
- **Syntax-driven Kernels**

Based on the syntax of the representation of the instances / data objects.  
Often special cases of **Convolution Kernels**, which combine kernels for parts.

- **String Kernels** (similarity of two strings based on number of common subsequences)
- **Tree Kernels** (consider all subtrees occurring in a (parse) tree for an instance)
- **Graph Kernels** (e.g. walk-based, i.e., count common walks in two graphs, or based on counting common paths, subtrees or subgraphs, or codeword-based, constructed from node equivalence classes)

# Example: Stripes with Gaussian Kernel

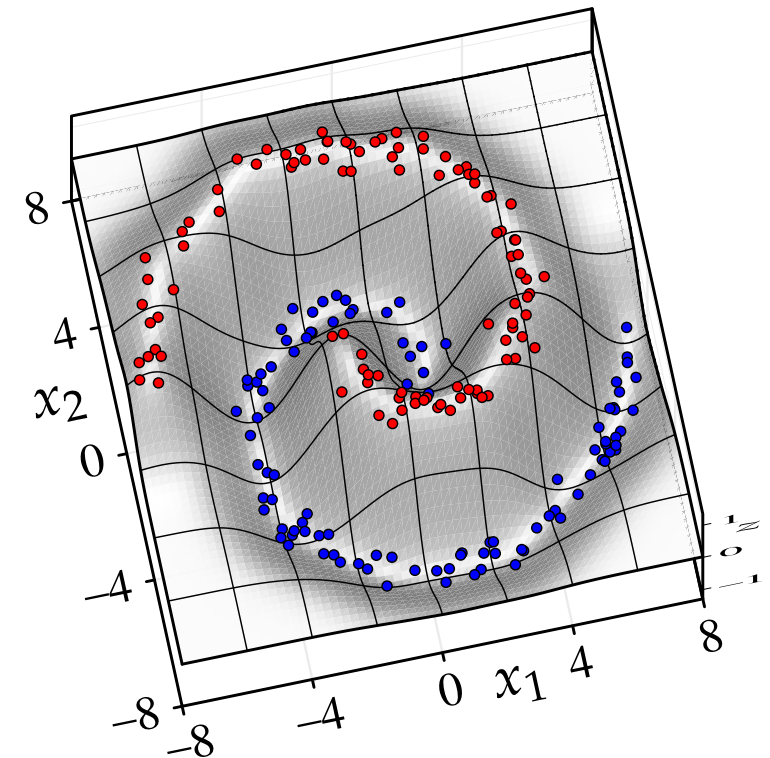
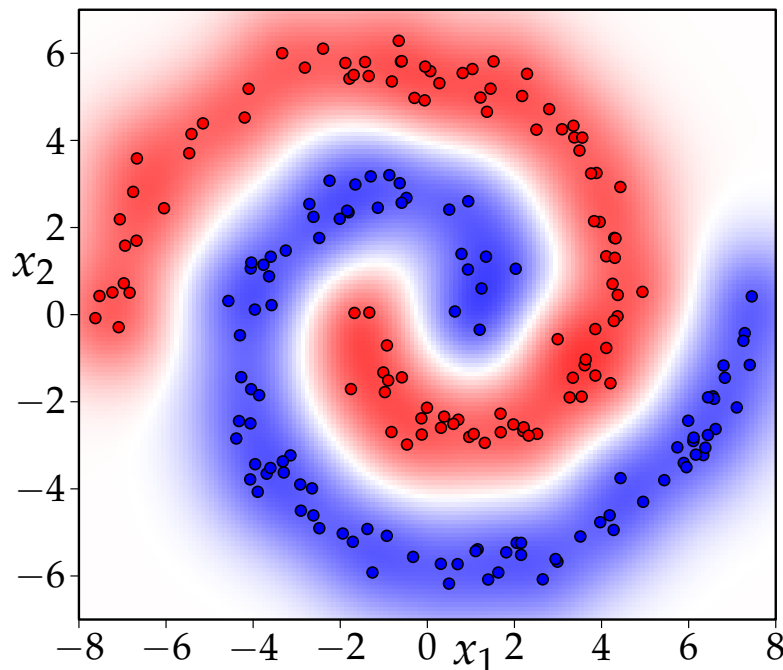
- Simple data set with two classes:
  - 20 data points belong to the **red** class,
  - 20 data points belong to the **blue** class.
- We apply an isotropic Gaussian kernel:
$$\kappa(\vec{x}_1, \vec{x}_2) = \exp\left(-\frac{1}{2\sigma^2}(\vec{x}_1 - \vec{x}_2)^\top (\vec{x}_1 - \vec{x}_2)\right).$$



- A Gaussian kernel models vicinity of data points in the data space, similar to a weighted nearest neighbor classifier.
- Result obtained with  $\sigma^2 = 1$  and  $\nu = 1$  (Mangasarian–Musicant variant).

# Example: Archimedean Spirals with Gaussian Kernel

- More complex data set with two classes:
  - 100 data points belong to the **red** class,
  - 100 data points belong to the **blue** class.
- We apply an isotropic Gaussian kernel:
$$\kappa(\vec{x}_1, \vec{x}_2) = \exp\left(-\frac{1}{2\sigma^2}(\vec{x}_1 - \vec{x}_2)^\top(\vec{x}_1 - \vec{x}_2)\right).$$



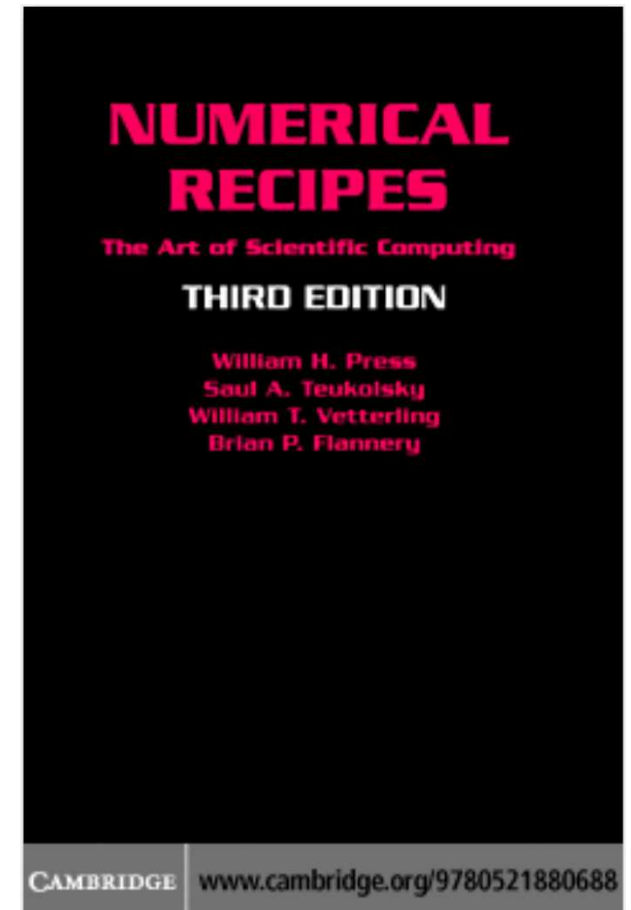
- A Gaussian kernel models vicinity of data points in the data space, similar to a weighted nearest neighbor classifier.
- Result obtained with  $\sigma^2 = 1$  and  $\nu = 1$  (Mangasarian–Musicant variant).

# Multi-Class Support Vector Machines

- **Support vector machines can solve directly only binary (two-class) classification problems.**
- Many real world problems have more than two classes, some have many. (e.g. iris / wine data: 3 classes, digit recognition: 10, letter recognition: 26/52)
- In order to assign data points to more than two classes, one usually transforms the problem into a **set of binary classification problems**.
- There are two main approaches for this:
  - **One classifier for each class**, separating it from all other classes.  
The class with the largest distance from the separating hyperplane wins.
  - **One classifier for each pair of classes**, separating them from each other.  
The class that wins the most class-pair classifications is chosen as the final classification.
- Hence one needs either  $c$  or  $c(c-1)/2$  (binary) support vector machines, where  $c > 2$  is the number of classes.

# Some Practical Advice

- Some advice from Section 16.5.6 of this book:
- For a polynomial kernel, start by choosing  $\alpha$  and  $\beta$  so that  $\alpha \vec{x}_i^\top \vec{x}_j + \beta$  lies in  $[-1, +1]$  for all  $i$  and  $j$ .
- The exponent  $r$  can be interpreted as the “size” (sum of powers, hence features) of the monomials to use. Larger  $r$  is not necessarily better.
- For a Gaussian kernel, choose  $\sigma$  as some characteristic distance between data points.
- For the penalty / regularization parameter  $\nu$  of the soft margin classifier, try  $\nu = 1$  first, then try increasing/decreasing by factors of 10. There is usually a wide plateau of good choices.
- While trying different  $\nu$ , check how many  $\hat{b}_i$  are fixed at 0, or at  $\nu$ , or are in-between. A good  $\nu$  has many at 0, fewer at  $\nu$ , and even fewer in-between.





# Support Vector Regression

- Support vector machines may also be used to address **regression** problems.
- The (hard) primal optimization problem of  **$\varepsilon$ -insensitive regression** is:

$$\begin{aligned} &\text{minimize} && \frac{1}{2} \vec{a}^\top \vec{a} && \text{(data points inside an } \varepsilon\text{-tube around } \vec{a}^{\circ\top} \vec{x}_i^{\circ}) \\ &\text{subject to} && \forall i; 1 \leq i \leq n : && y_i - \vec{a}^{\circ\top} \vec{x}_i^{\circ} \leq \varepsilon, && (\varepsilon \text{ is fixed,} \\ &&& \forall i; 1 \leq i \leq n : && \vec{a}^{\circ\top} \vec{x}_i^{\circ} - y_i \leq \varepsilon, && \text{as specified} \\ &&& && \varepsilon \geq 0. && \text{by a user)} \end{aligned}$$

(The double inequality constraints avoid using an absolute value, which causes differentiation problems.)

- The soft primal optimization problem uses slack variables  $\zeta_i^\uparrow$  and  $\zeta_i^\downarrow$ :

$$\begin{aligned} &\text{minimize} && \frac{1}{2} \vec{a}^\top \vec{a} + \frac{\nu}{2} \sum_{i=1}^n (\zeta_i^{\uparrow 2} + \zeta_i^{\downarrow 2}) && \text{(Attention: 2-norm penalty!)} \\ &\text{subject to} && \forall i; 1 \leq i \leq n : && y_i - \vec{a}^{\circ\top} \vec{x}_i^{\circ} \leq \varepsilon + \zeta_i^\uparrow, \\ &&& && \vec{a}^{\circ\top} \vec{x}_i^{\circ} - y_i \leq \varepsilon + \zeta_i^\downarrow, && \varepsilon \geq 0, \quad \zeta_i^\uparrow, \zeta_i^\downarrow \geq 0. \end{aligned}$$

- Note that  $\zeta_i^\uparrow$  and  $\zeta_i^\downarrow$  cannot both be positive at the same time, because a (true) target value  $y_i$  cannot be both above and below the  $\varepsilon$ -tube around the prediction function  $\vec{a}^{\circ\top} \vec{x}_i^{\circ}$ . Therefore  $\zeta_i^\uparrow \zeta_i^\downarrow = 0$ .



# Soft Support Vector Regression

- The **Lagrange function** of soft support vector regression is:

$$\begin{aligned}\mathcal{L}(\vec{a}^\circ, \vec{b}^\uparrow, \vec{b}^\downarrow) &= \frac{1}{2} \vec{a}^\top \vec{a} + \frac{\nu}{2} \sum_{i=1}^n (\xi_i^{\uparrow 2} + \xi_i^{\downarrow 2}) - \sum_{i=1}^n b_i^\uparrow (\varepsilon + \xi_i^\uparrow - y_i + \vec{a}^{\circ\top} \vec{x}_i^\circ) \\ &\quad - \sum_{i=1}^n b_i^\downarrow (\varepsilon + \xi_i^\downarrow - \vec{a}^{\circ\top} \vec{x}_i^\circ + y_i) \\ &= \frac{1}{2} \vec{a}^\top \vec{a} + \frac{\nu}{2} \sum_{i=1}^n (\xi_i^{\uparrow 2} + \xi_i^{\downarrow 2}) - \varepsilon \sum_{i=1}^n (b_i^\uparrow + b_i^\downarrow) \\ &\quad - \sum_{i=1}^n (b_i^\uparrow \xi_i^\uparrow + b_i^\downarrow \xi_i^\downarrow) - \sum_{i=1}^n (b_i^\uparrow - b_i^\downarrow) (\vec{a}^\top \vec{x}_i + a_0 - y_i).\end{aligned}$$

- In analogy to support vector classification, a **dual formulation** is obtained by exploiting the Karush–Kuhn–Tucker conditions to **eliminate the primal parameters**  $\vec{a}$  from the Lagrange function.

$$\nabla_{\vec{a}} \mathcal{L} \stackrel{!}{=} \vec{0}, \quad \frac{\partial \mathcal{L}}{\partial a_0} \stackrel{!}{=} 0, \quad \frac{\partial \mathcal{L}}{\partial \xi_i^\uparrow} \stackrel{!}{=} 0, \quad \frac{\partial \mathcal{L}}{\partial \xi_i^\downarrow} \stackrel{!}{=} 0.$$

# Soft Support Vector Regression

- As derivatives w.r.t. the primal parameters  $\vec{a}$  and  $a_0$  we obtain

$$\begin{aligned}\nabla_{\vec{a}} \mathcal{L} &= \vec{a} - \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \vec{x}_i \stackrel{!}{=} \vec{0}, & \Leftrightarrow & \vec{a} = \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \vec{x}_i \\ \frac{\partial \mathcal{L}}{\partial a_0} &= - \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \stackrel{!}{=} 0 & \Leftrightarrow & \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) = 0.\end{aligned}$$

- As derivatives w.r.t. the slack variables  $\zeta_i^{\uparrow}$  and  $\zeta_i^{\downarrow}$  we obtain

$$\begin{aligned}\forall i; 1 \leq i \leq n : \quad \frac{\partial \mathcal{L}}{\partial \zeta_i^{\uparrow}} &= \nu \zeta_i^{\uparrow} - b_i^{\uparrow} \stackrel{!}{=} 0 & \Leftrightarrow & \zeta_i^{\uparrow} = \frac{1}{\nu} b_i^{\uparrow}, \\ \frac{\partial \mathcal{L}}{\partial \zeta_i^{\downarrow}} &= \nu \zeta_i^{\downarrow} - b_i^{\downarrow} \stackrel{!}{=} 0 & \Leftrightarrow & \zeta_i^{\downarrow} = \frac{1}{\nu} b_i^{\downarrow}.\end{aligned}$$

- Recall that  $\zeta_i^{\uparrow}$  and  $\zeta_i^{\downarrow}$  cannot both be positive at the same time, because a (true) target value  $y_i$  cannot be both above and below the  $\varepsilon$ -tube around the prediction function  $\vec{a}^{\circ\top} \vec{x}_i^{\circ}$ . Therefore  $\zeta_i^{\uparrow} \zeta_i^{\downarrow} = 0$ .
- As a consequence,  $b_i^{\uparrow}$  and  $b_i^{\downarrow}$  cannot both be positive at the same time, because (at least) one constraint must be inactive. Therefore  $b_i^{\uparrow} b_i^{\downarrow} = 0$ .

# Soft Support Vector Regression

- The resulting **reduced Lagrange function** (to be maximized) is

$$\begin{aligned}
 \mathcal{L}(\vec{b}^{\uparrow}, \vec{b}^{\downarrow}) &= \frac{1}{2} \left( \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \vec{x}_i \right)^{\top} \left( \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \vec{x}_i \right) - \varepsilon \sum_{i=1}^n (b_i^{\uparrow} + b_i^{\downarrow}) \\
 &\quad + \frac{\nu}{2} \sum_{i=1}^n \left( \left( \frac{1}{\nu} b_i^{\uparrow} \right)^2 + \left( \frac{1}{\nu} b_i^{\downarrow} \right)^2 \right) - \sum_{i=1}^n \left( b_i^{\uparrow} \left( \frac{1}{\nu} b_i^{\uparrow} \right) + b_i^{\downarrow} \left( \frac{1}{\nu} b_i^{\downarrow} \right) \right) \\
 &\quad - \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \left( \left( \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \vec{x}_i \right)^{\top} \vec{x}_i + a_0 - y_i \right) \\
 &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \vec{x}_i^{\top} \vec{x}_j (b_j^{\uparrow} - b_j^{\downarrow}) - \varepsilon \sum_{i=1}^n (b_i^{\uparrow} + b_i^{\downarrow}) \\
 &\quad + \frac{1}{2\nu} \sum_{i=1}^n (b_i^{\uparrow 2} + b_i^{\downarrow 2}) - \frac{1}{\nu} \sum_{i=1}^n (b_i^{\uparrow 2} + b_i^{\downarrow 2}) \\
 &\quad - \sum_{i=1}^n \sum_{j=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \vec{x}_i^{\top} \vec{x}_j (b_j^{\uparrow} - b_j^{\downarrow}) + \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) (y_i - a_0) \\
 &= \dots
 \end{aligned}$$

(continued on next slide)

# Soft Support Vector Regression

- The resulting **reduced Lagrange function** (to be maximized) is

$$\begin{aligned}\mathcal{L}(\vec{b}^{\uparrow}, \vec{b}^{\downarrow}) &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \vec{x}_i^{\top} \vec{x}_j (b_j^{\uparrow} - b_j^{\downarrow}) - \varepsilon \sum_{i=1}^n (b_i^{\uparrow} + b_i^{\downarrow}) \\ &\quad - \frac{1}{2\nu} \sum_{i=1}^n (b_i^{\uparrow 2} + b_i^{\downarrow 2}) + \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) y_i - a_0 \underbrace{\sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow})}_{=0} \\ &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) \vec{x}_i^{\top} \vec{x}_j (b_j^{\uparrow} - b_j^{\downarrow}) - \varepsilon \sum_{i=1}^n (b_i^{\uparrow} + b_i^{\downarrow}) \\ &\quad - \frac{1}{2\nu} \sum_{i=1}^n \left( (b_i^{\uparrow} - b_i^{\downarrow})^2 - 2 \underbrace{b_i^{\uparrow} b_i^{\downarrow}}_{=0} \right) + \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) y_i \\ &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) (\vec{x}_i^{\top} \vec{x}_j + \frac{1}{\nu} \delta_{ij}) (b_j^{\uparrow} - b_j^{\downarrow}) \\ &\quad - \varepsilon \sum_{i=1}^n (b_i^{\uparrow} + b_i^{\downarrow}) + \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) y_i\end{aligned}$$

# Soft Support Vector Regression

- The **dual problem** of soft  $\varepsilon$ -insensitive support vector regression is therefore:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) (\vec{x}_i^{\top} \vec{x}_j + \frac{1}{\nu} \delta_{ij}) (b_j^{\uparrow} - b_j^{\downarrow}) + \varepsilon \sum_{i=1}^n (b_i^{\uparrow} + b_i^{\downarrow}) - \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) y_i \\ \text{subject to} \quad & \forall i; 1 \leq i \leq n : \quad b_i^{\uparrow} \geq 0, \quad b_i^{\downarrow} \geq 0 \quad \text{and} \quad \sum_{i=1}^n (b_i^{\uparrow} - b_i^{\downarrow}) = 0, \end{aligned}$$

where  $\vec{b}^{\uparrow}$  and  $\vec{b}^{\downarrow}$  are the **dual parameters**.

- In addition, we have the Karush–Kuhn–Tucker conditions

$$\begin{aligned} \forall i; 1 \leq i \leq n : \quad & b_i^{\uparrow} (\varepsilon + \xi_i^{\uparrow} - y_i + \vec{a}^{\top} \vec{x}_i + a_0) = 0, & \xi_i^{\uparrow} \xi_i^{\downarrow} &= 0, \\ & b_i^{\downarrow} (\varepsilon + \xi_i^{\downarrow} - \vec{a}^{\top} \vec{x}_i + a_0 + y_i) = 0, & b_i^{\uparrow} b_i^{\downarrow} &= 0. \end{aligned}$$

- The **primal parameters** can be computed from the dual parameters as (see above)

$$\hat{\vec{a}} = \sum_{i=1}^n (\hat{b}_i^{\uparrow} - \hat{b}_i^{\downarrow}) \vec{x}_i.$$

- The **prediction function** (i.e. the function to compute  $y$  from given  $\vec{x}$ ) is

$$f(\vec{x}) = \sum_{i=1}^n (\hat{b}_i^{\uparrow} - \hat{b}_i^{\downarrow}) \vec{x}_i^{\top} \vec{x} + a_0.$$

( $a_0$  can be determined from a Karush–Kuhn–Tucker condition with  $b_i^{\uparrow} > 0$  or  $b_i^{\downarrow} > 0$ )

# Support Vector Regression and Ridge Regression

- By defining  $b_i = b_i^\uparrow - b_i^\downarrow$ , the **dual problem** can be written as

$$\begin{aligned} &\text{minimize} && \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n b_i (\vec{x}_i^\top \vec{x}_j + \frac{1}{v} \delta_{ij}) b_j + \varepsilon \sum_{i=1}^n |b_i| - \sum_{i=1}^n b_i y_i \\ &\text{subject to} && \sum_{i=1}^n b_i = 0. \end{aligned}$$

- In this form, if we choose  $\varepsilon = 0$ , we have as the **Lagrange function**

$$\begin{aligned} \mathcal{L}(\vec{b}, \lambda) &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n b_i (\vec{x}_i^\top \vec{x}_j + \frac{1}{v} \delta_{ij}) b_j - \sum_{i=1}^n b_i y_i - \lambda \sum_{i=1}^n b_i \\ &= \frac{1}{2} \vec{b}^\top (\mathbf{G} + \frac{1}{v} \mathbf{I}_n) \vec{b} - \vec{b}^\top \vec{y} - \lambda \sum_{i=1}^n b_i. \end{aligned}$$

This special case is actually equivalent to **ridge regression**.

- Reminder: In ridge regression, the dual parameters are computed as

$$\vec{b} = (\mathbf{G}^\circ + \lambda \mathbf{I}_n)^{-1} \vec{y} \quad \text{with a regularization parameter } \lambda.$$

Here we get an analogous result with regularization parameter  $\frac{1}{v}$ .

# Support Vector Regression and Ridge Regression

- We can find the **optimum of the Lagrange function** by taking derivatives w.r.t. the parameters (here:  $\vec{b}$ ) and setting them equal to zero:

$$\nabla_{\vec{b}} \mathcal{L} = \vec{y} - (\mathbf{G} + \frac{1}{v} \mathbf{I}_n) \vec{b} \stackrel{!}{=} 0.$$

- This leads to  $\vec{y} = (\mathbf{G} + \frac{1}{v} \mathbf{I}_n) \vec{b} = (\mathbf{X}\mathbf{X}^\top + \frac{1}{v} \mathbf{I}_n) \vec{b}$   
and therefore  $\vec{b} = (\mathbf{G} + \frac{1}{v} \mathbf{I}_n)^{-1} \vec{y} = (\mathbf{X}\mathbf{X}^\top + \frac{1}{v} \mathbf{I}_n)^{-1} \vec{y}$ ,  
which is the dual form of the ridge regression solution.

- Note, however, that in the original discussion we had:

$$\vec{b} = (\mathbf{G}^\circ + \lambda \mathbf{I}_n)^{-1} \vec{y} = (\mathbf{X}^\circ \mathbf{X}^{\circ\top} + \lambda \mathbf{I}_n)^{-1} \vec{y}.$$

What is the reason for this difference? ( $\mathbf{G}$  versus  $\mathbf{G}^\circ = \mathbf{G} + \vec{1}_n \vec{1}_n^\top$ )

- The difference results from what parameter vector's length is constrained.
  - If the constraint is  $\vec{a}^\top \vec{a} \leq c^2$ , then we have to use  $\mathbf{G}$ .
  - If the constraint is  $\vec{a}^{\circ\top} \vec{a}^\circ \leq c^2$ , then we have to use  $\mathbf{G}^\circ$ .

# Ridge Regression / Tikhonov Regularization

- One justification of **ridge regression** is to introduce the constraint  $\vec{a}^\top \vec{a} \leq c^2$ , where  $c$  is some user-specified number, and to incorporate it into the functional to optimize with a **Lagrange multiplier**  $\lambda$ :

$$\mathcal{L}(\vec{a}^\circ, \lambda) = (\mathbf{X}^\circ \vec{a}^\circ - \vec{y})^\top (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}) - \lambda(c^2 - \vec{a}^\top \vec{a})$$

The resulting **Lagrange function**  $\mathcal{L}(\vec{a}^\circ, \lambda)$  has to be minimized.

- Alternatively, one may introduce the constraint  $\vec{a}^{\circ\top} \vec{a}^\circ \leq c^2$ , which leads to

$$\mathcal{L}(\vec{a}^\circ, \lambda) = (\mathbf{X}^\circ \vec{a}^\circ - \vec{y})^\top (\mathbf{X}^\circ \vec{a}^\circ - \vec{y}) - \lambda(c^2 - \vec{a}^{\circ\top} \vec{a}^\circ)$$

- Based on the former, **ridge regression** consists in solving the problem:

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^n \xi_i^2 \\ &\text{subject to} && \forall i; 1 \leq i \leq n : \quad \vec{a}^{\circ\top} \vec{x}_i^\circ - \vec{y} = \xi_i \quad \text{and} \quad \vec{a}^\top \vec{a} \leq c. \end{aligned}$$

- The **Lagrange function** (where  $\vec{\beta}$  are the Lagrange multipliers) is

$$\mathcal{L}(\vec{a}^\circ, \vec{\xi}, \vec{b}, \lambda) = \sum_{i=1}^n \xi_i^2 - \sum_{i=1}^n b_i (\vec{a}^{\circ\top} \vec{x}_i^\circ - \vec{y} - \xi_i) - \lambda(c^2 - \vec{a}^\top \vec{a}).$$



# Ridge Regression / Tikhonov Regularization

- The **Lagrange function** (where  $\vec{\beta}$  are the Lagrange multipliers) is

$$\mathcal{L}(\vec{a}^\circ, \vec{\xi}, \vec{b}, \lambda) = \sum_{i=1}^n \xi_i^2 - \sum_{i=1}^n b_i (\vec{a}^{\circ\top} \vec{x}_i^\circ - \vec{y} - \xi_i) - \lambda (c^2 - \vec{a}^\top \vec{a}).$$

- As derivatives w.r.t. the primal parameters  $\vec{a}$  and  $a_0$  we obtain

$$\begin{aligned} \nabla_{\vec{a}} \mathcal{L} &= - \sum_{i=1}^n \beta_i \vec{x}_i + 2\lambda \vec{a} \stackrel{!}{=} \vec{0} & \Leftrightarrow & \quad \vec{a} = \frac{1}{2\lambda} \sum_{i=1}^n \beta_i \vec{x}_i, \\ \frac{\partial \mathcal{L}}{\partial a_0} &= - \sum_{i=1}^n \beta_i \stackrel{!}{=} 0 & \Leftrightarrow & \quad \sum_{i=1}^n \beta_i = 0, \end{aligned}$$

- As derivatives w.r.t. the residuals  $\xi_i$  we obtain

$$\forall i; 1 \leq i \leq n : \quad \frac{\partial \mathcal{L}}{\partial \xi_i} = 2\xi_i + \beta_i \stackrel{!}{=} 0 \quad \Leftrightarrow \quad \xi_i = -\frac{1}{2}\beta_i.$$

- As usual, we substitute the obtained expressions into the Lagrange function, in order to eliminate the parameters  $\vec{a}$  and the residuals  $\xi_i, i = 1, \dots, n$ . This yields the **reduced Lagrange function**  $\mathcal{L}(\vec{\beta}, \lambda)$ .

# Ridge Regression / Tikhonov Regularization

- The resulting **reduced Lagrange function** is:

$$\begin{aligned}\mathcal{L}(\vec{\beta}, \lambda) &= \sum_{i=1}^n \left(-\frac{1}{2}\beta_i\right)^2 - \sum_{i=1}^n \beta_i \left(\frac{1}{2\lambda} \sum_{j=1}^n \beta_j \vec{x}_j\right)^\top \vec{x}_i + \sum_{i=1}^n \beta_i (y_i - a_0) \\ &\quad + \sum_{i=1}^n \beta_i \left(-\frac{1}{2}\beta_i\right) + \lambda \left(\frac{1}{2\lambda} \sum_{i=1}^n \beta_i \vec{x}_i\right)^\top \left(\frac{1}{2\lambda} \sum_{i=1}^n \beta_i \vec{x}_i\right) - \lambda c^2 \\ &= \frac{1}{4} \sum_{i=1}^n \beta_i^2 - \frac{1}{2\lambda} \sum_{i=1}^n \sum_{j=1}^n \beta_i \vec{x}_i^\top \vec{x}_j \beta_j + \sum_{i=1}^n \beta_i (y_i - a_0) \\ &\quad - \frac{1}{2} \sum_{i=1}^n \beta_i^2 + \frac{1}{4\lambda} \sum_{i=1}^n \sum_{j=1}^n \beta_i \vec{x}_i^\top \vec{x}_j \beta_j - \lambda c^2 \\ &= -\frac{1}{4} \sum_{i=1}^n \beta_i^2 - \frac{1}{4\lambda} \sum_{i=1}^n \sum_{j=1}^n \beta_i \vec{x}_i^\top \vec{x}_j \beta_j + \sum_{i=1}^n \beta_i y_i - a_0 \underbrace{\sum_{i=1}^n \beta_i}_{=0} - \lambda c^2 \\ &= -\frac{1}{4\lambda} \sum_{i=1}^n \sum_{j=1}^n \beta_i (\vec{x}_i^\top \vec{x}_j + \lambda \delta_{ij}) \beta_j + \sum_{i=1}^n \beta_i y_i - \lambda c^2\end{aligned}$$

# Ridge Regression / Tikhonov Regularization

- Letting  $b_i = \frac{\beta_i}{2\lambda}$  be the dual parameters, that is,  $\beta_i = 2\lambda b_i$ , we get

$$\begin{aligned}\mathcal{L}(\vec{b}, \lambda) &= -\frac{1}{4\lambda} \sum_{i=1}^n \sum_{j=1}^n (2\lambda b_i) (\vec{x}_i^\top \vec{x}_j + \lambda \delta_{ij}) (2\lambda b_j) + \sum_{i=1}^n 2\lambda b_i y_i - \lambda c^2 \\ &= -\lambda \sum_{i=1}^n \sum_{j=1}^n b_i (\vec{x}_i^\top \vec{x}_j + \lambda \delta_{ij}) b_j + 2\lambda \sum_{i=1}^n b_i y_i - \lambda c^2\end{aligned}$$

- With  $\lambda \neq 0$ , we can divide by  $2\lambda$  and also discard the constant term  $c^2$ , because it does not change (the location of) the optimum. This leads to

$$\mathcal{L}(\vec{b}, \lambda) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n b_i (\vec{x}_i^\top \vec{x}_j + \lambda \delta_{ij}) b_j + \sum_{i=1}^n b_i y_i.$$

- The corresponding optimization problem is

$$\begin{aligned}&\text{minimize} && \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n b_i (\vec{x}_i^\top \vec{x}_j + \lambda \delta_{ij}) b_j - \sum_{i=1}^n b_i y_i \\ &\text{subject to} && \sum_{i=1}^n b_i = 0.\end{aligned}$$

# Reminder: Kernel Regression in Statistics

- If no parametric description of the functional relationship  $f$  is known, non-parametric techniques like **kernel regression** may be used.

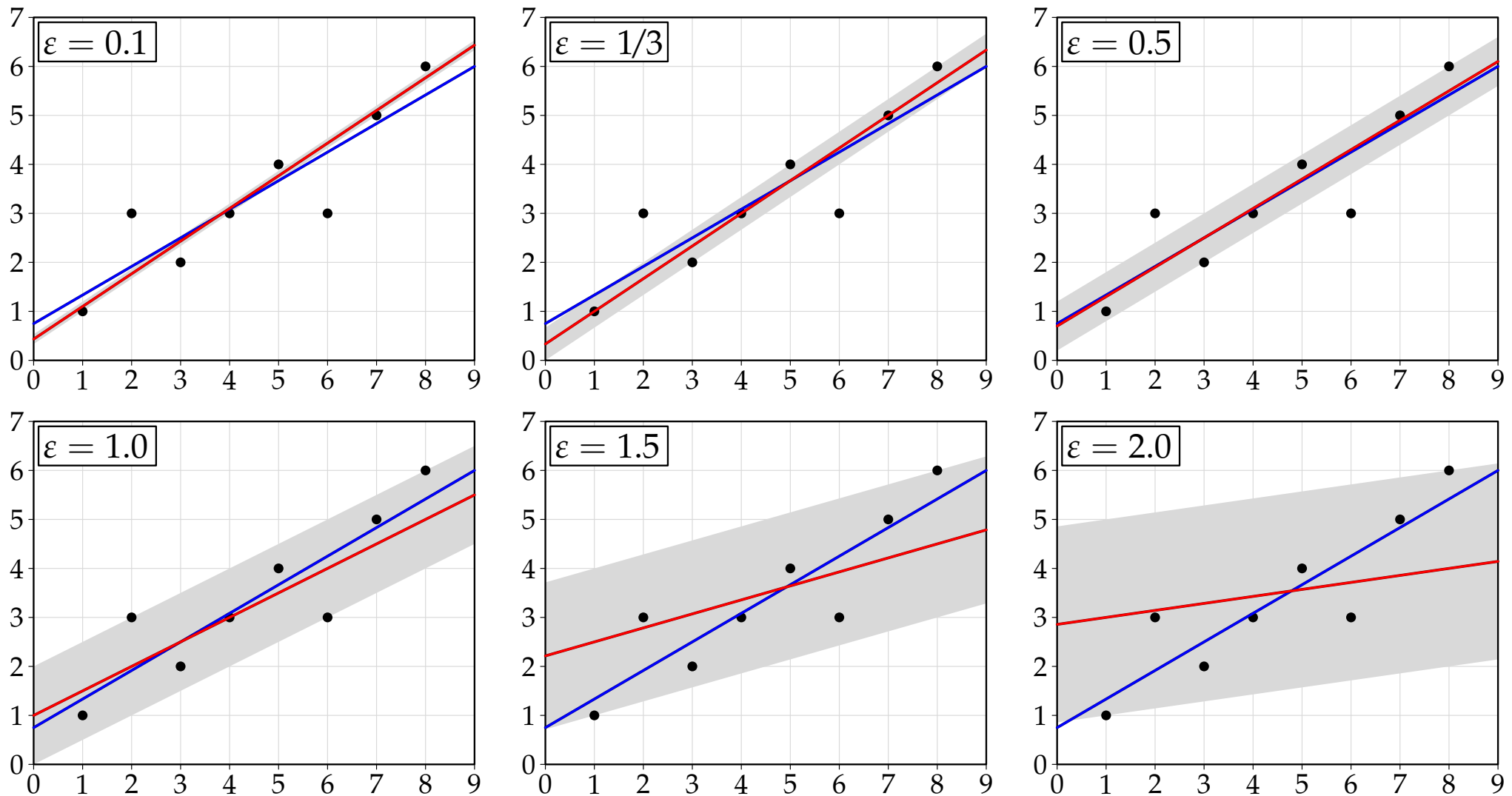
- **(Nadaraya–Watson) Kernel Regression:**

Given data  $(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_n, y_n)$ , form the regression model:

$$\hat{f}_h(\vec{x}) = \frac{\sum_{i=1}^n \kappa(\vec{x}, \vec{x}_i; h) y_i}{\sum_{i=1}^n \kappa(\vec{x}, \vec{x}_i; h)}, \quad \text{where}$$

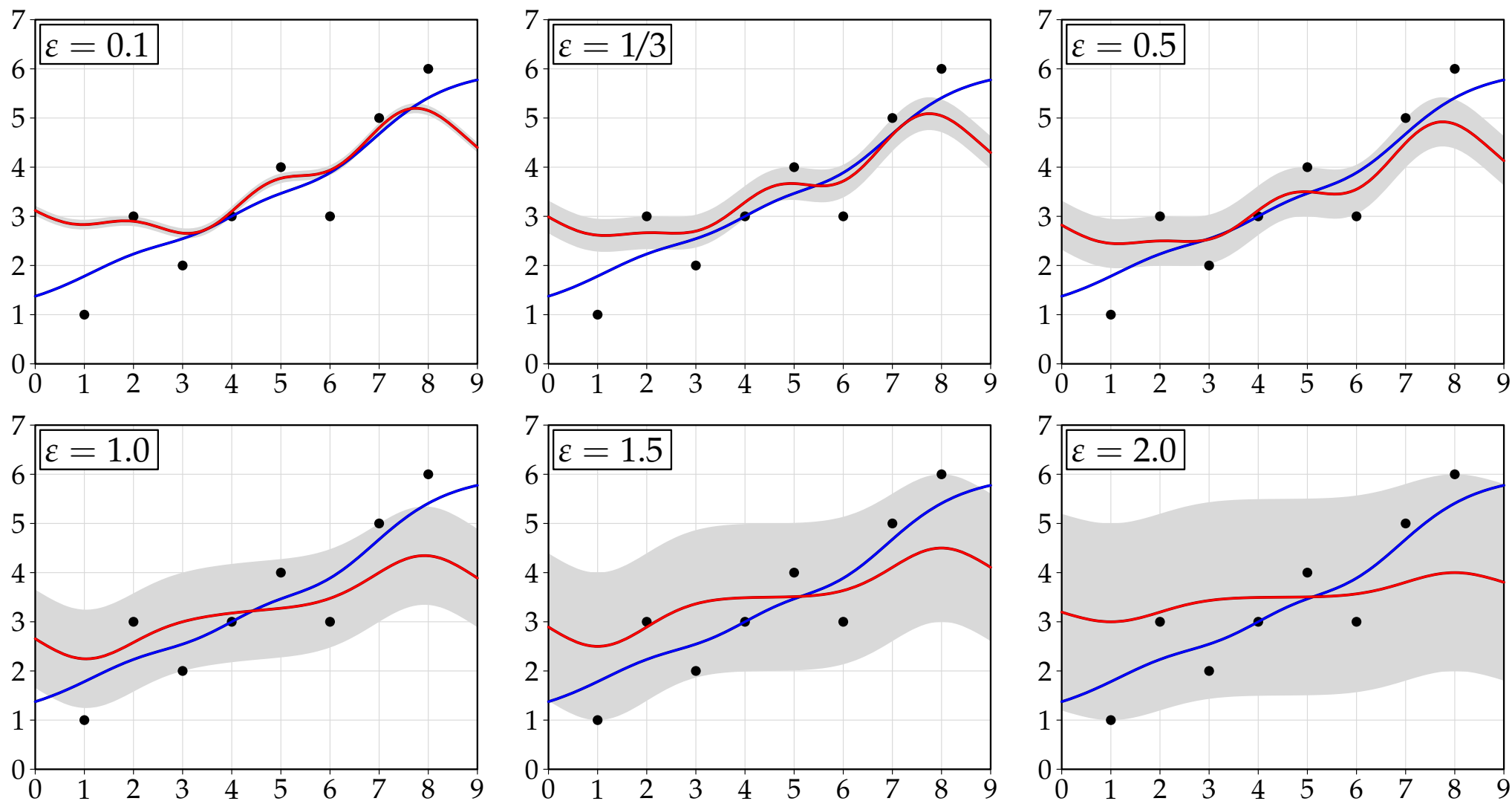
- $\kappa$ : **kernel** (in statistics usually a probability density, e.g. density of  $\mathcal{N}(0, 1)$  or density of  $\mathcal{U}(-1, 1)$ , applied to  $\frac{|\vec{x} - \vec{x}_i|}{h}$ )
- $h$ : **bandwidth** (smoothing parameter,  $h > 0$ )
- $\vec{x}$ : point at which the estimator is evaluated.
- $\hat{f}_h(\vec{x})$  is a weighted mean of the values  $y_1, \dots, y_n$ ; the larger  $|\vec{x} - \vec{x}_i|$ , the less weight  $y_i$  has for the calculation of  $\hat{f}_h(\vec{x})$ .
- **Support vector regression:** parameters  $\hat{b}_i$  instead of  $y_i$   $(\sum_{i=1}^n \kappa(\vec{x}, \vec{x}_i; h))^{-1}$ .

# Support Vector Regression: Linear Example



- Linear regression (blue) and support vector regression with  $\nu = 1$  (red).

# Support Vector Regression: Gaussian Kernel Example



- Kernel regression ( $h = \frac{1}{2}$ , blue) and support vector regression with  $\nu = 1$  (red).

# Summary Support Vector Machines

- The fundamental principle of support vector machines can be formulated as:
  - If the task is regression, linear regression suffices.
  - If the task is classification, linear classification suffices.
- For this principle to work the so-called **kernel trick** is decisive:
  - implicitly **map the data points** to some image space,
  - **compute scalar products** in this image space **via a kernel**,
  - **train a (linear) model** using only these scalar products.
- Since the (implicit) mapping to the image space may be non-linear, this classifier / predictor may be **non-linear in the original (data) space**.
- How well this works depends decisively on the **choice of the kernel**, since it has to capture all information about the non-linear structure of the data.
- To allow for some misclassifications and larger deviations from a regression function, so-called **slack variables** may be introduced.