Model Selection

CRISP-DM

Cross Industry Standard Process for Data Mining

Data Mining Process Model developed within an EU project

Several phases that are repeated until data mining project is finished

CRISP-Phases

Project understanding

Data understanding

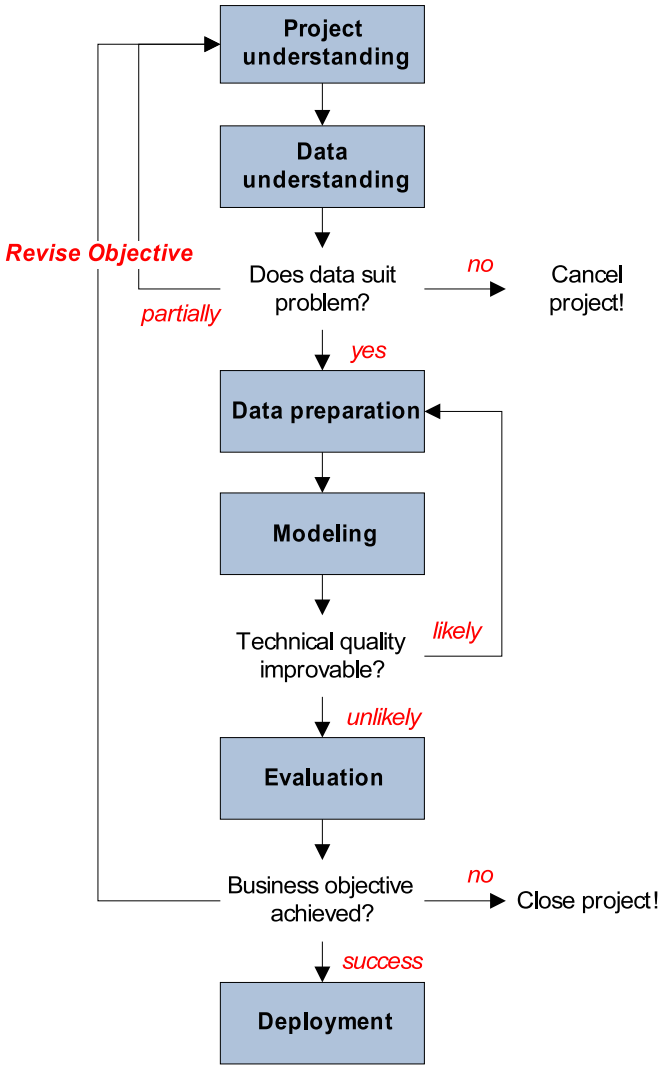
Data preparation

Modeling

Evaluation

Deployment

CRISP-DM Model



Project understanding

- What exactly is the problem, what the expected benefit?
- What should a solution look like?
- What is known about the domain?

Data understanding

- What (relevant) data is available?
- What about data quality/quantity/recency?

Data preparation

- Can data quality be increased?
- How can it be transformed for modeling?

Modeling

- What models is best suited to solve the problem?
- What is the best technique to get the model?
- How good does the model perform technically?

Evaluation

- How good is the model in terms of project requirements?
- What have we learned from the project?

Deployment

- How can the model be best deployed?
- Is there a way to know if the model is still valid?

Modelling Steps

Select the score function. The score function evaluates the possible *models*.

- We aim to find the best model with respect to our goals.
- What we mean by good or best model is formalized by the score function.
- In the case of the simple linear regression function, our score function will tell us which specific choice of the coefficients is better when we compare different linear functions.

Apply the algorithm. The score function helps us compare models, but needs to be optimized.

Validate the results. Even if our optimization algorithm find the best model, the chosen model class might still be bad.

The **model class** or **architecture** specifies the general form of the analysis result.

Model Classes

Linear Models The simplest case of a linear model is a regression line $y = ax+b$, describing the idealized relationship between attributes x and y.

Constant Models E.g. mean or median value for numerical, mode for categorical attributes.

Parameterised models are determined by specifying the values of a fixed number of parameters. Example: A regression line: $f(x) = a_1 x + a_0$ is uniquely determined by the two parameters a_0 and a_1.

Non-parameterised models:

- Polynomials: $f(x) = a_k x^k + \ldots + a_0$ where the degree k is not fixed.
- Association rules

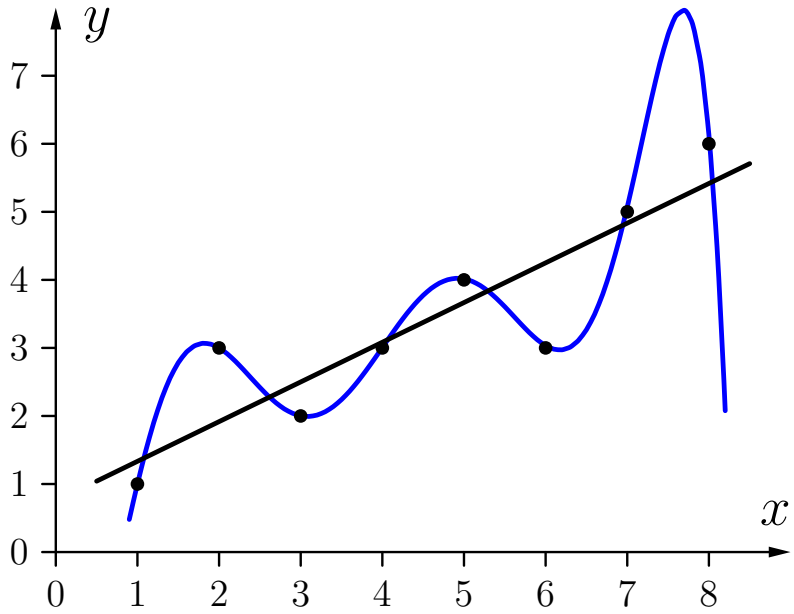
Global Models provide a (not necessarily good) description for the whole data set. Example: Regression line

Local Models Local models or patterns provide a description for only a part or subset of the data set. Example: Association rules

Model Selection

Objective: select the model (also: architecture) 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.*

How to find the best model?

Project understanding will limit the choice of available models but will usually not restrict to one class. Example: If the data analysis task is identified as a regression problem, clustering models are not suitable. But it is not specified which kind of regression function should be chosen.

The model class must still be selected. The choice of the wrong model class can spoil the whole data analysis process.

It must be formalised what we mean by a *good* model.

Criteria for good models

Fitting criterion How well does the model fit the data?
Examples: Error function for regression or for MDS.

Model complexity Usually, simpler models are preferred because they are easier to understand, easier to fit and avoid overfitting. Overfitting refers to the problem that complex models might be flexible enough to fit the given data almost perfectly without representing the general structure in the data.

Interpretability is often desired although sometimes also black box models suffice.
Example: A line is easier to interpret as a complex polynom.

Computational aspects Finding at least a good model w.r.t. the criteria data fitting, model complexity and interpretability is also a matter of computational complexity.

Fitting Criteria and Score Functions

The choice of the model class determines only the general structure of the model, very often in terms of a set of parameters.

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

$$f : M \mapsto \mathbb{R}$$

where M is the set of considered models.

Example: Regression line $y = ax + b$. A model is characterized by two real parameters, so that the objective function simplifies to

$$f : \mathbb{R}^2 \mapsto \mathbb{R}$$

Typical choice for f: Mean square error.

$$f = E(a, b) = \frac{1}{n} \sum_{i=1}^{n} (ax_i + b - y_i)^2$$

where $(x_1, y_1) \ldots (x_n, y_n)$ is the data set.

Fitting Criteria and Score Functions

The objective function provides only means to compare different solutions.

In the case of the regression line one can compare two lines given by the parameter combinations (a_1, b_1) and (a_2, b_2) respectively, by the objective function. The regression line given by (a_1, b_1) fits better than the one given by (a_2, b_2) if $E(a_1, b_1) < E(a_2, b_2)$.

The objective function does not tell us *how* to find the best model.

For the simple example of a constant value m as model for a numerical attribute, one could choose the mean squared error which leads to the mean value as best choice or the mean absolute error which in return leads to the median as the best choice for m

Types of Errors

When fitting a model to data based on an error measure, a perfect fit will be seldom possible.

A perfect fit with zero error is suspicious in most cases.

It is often an indication that the model fitted to the data might be too flexible and is able to represent the data exactly, but does not show the actual structures in the data.

This effect is called overfitting.

Types of Errors

The fitting error can decomposed into four components.

The **pure or experimental error** is inherent in the data and due to: noise, random variations, impreciseness, incluence of hidden variables that cannot be observed. It is *impossible* to overcome this error.

The **sample error** is caused by the fact that the data set is only an imperfect representation of the underlying distribution of the data.
Especially for small samples, the sample distribution might deviate strongly from the true distribution.
Sometimes the sample is biased. Some types of instances might be missing or underrepresented in the sample, e.g. questionnaires might not be answered and returned by all customer groups according to their distribution.

The **lack of fit** or **model error** is caused by an ill-suited model. E.g. fitting a simple regression line to data with quadratic dependency.

The **algorithmic error** is caused by the method that is used to fit the model or the model parameters. Since in most cases heuristic search or fitting strategies are used to fit the model, the algorithm might not find the best model (parameters), but only a model that corresponds to a local optimum of the score function.

No Free Lunch!

The algorithmic and the model errors are often considered together as the machine learning bias.

The machine learning bias can be controlled to a certain extend, by choosing an appropriate model and good fitting algorithm.

There cannot be a single best algorithm providing a good model for every data set.

No-Free-Lunch-Theorem: *Roughly* For every kind of model an algorithm is good at, there is another kind of model it is bad at.

Error Functions For Classification Problems

Very common choice as error function for classification problem: The misclassification rate, i.e. the proportion of records that are wrongly classified.

A low misclassification rate does not necessarily tell anything about the quality of a classifier.

A low misclassification rate might be easily achieved when classes are extremely unbalanced, i.e. when the distribution of classes deviates strongly from a uniform distribution
One can cope with that by correcting the misclassification rate for chance, i.e. consider how much better the result is than guessing.

Alternative Error Functions: Cost Matrix

Example Data from production of tea cups where the classes are *broken* and *ok*. When 99% of the production are *ok*, a classifier always predicting *ok* will have a misclassification rate of 1% .

More general approach than the misclassification rate: cost function or cost matrix.

The consequences (costs) for misclassifications for one class might be different than for another class.

Example Tea cup production.

	predicted class	
true class	OK	broken
OK	0	c_1
broken	c_2	0

When an intact cup is classified as broken, the cup must be produced again. Therefore, the costs c_1 caused by this error causes are equal to the production costs for one cup.

The costs c_2 for a broken cup classified as intact are more difficult to estimate. They include e.g. reproduction cost, mailing cost, loss of reputation.

In general a cost matrix is an $m \times m$ matrix for m classes.

Alternative Error Functions: Cost Matrix

When such a cost matrix is provided, the expected loss given by

$$loss(c_i|E) = \sum_{j=1}^{m} P(c_j|E)c_{ji}$$

should be minimized.

E is the evidence, i.e. the observed values of the predictor attributes used for the classification.

$P(c_j|E)$ is the predicted probability that the true class is c_j given observation E.

Example(Hypothecial) cost matrix for the tea cup production problem

	predicted	class
true class	OK	broken
OK	0	1
broken	10	0

- A classifier might classify a specific cup with 80% to the class *ok* and with 20% to the class *broken*.
- Expected loss for choosing *ok* : $0.8 \cdot 0 + 0.2 \cdot 10 = 2$.
- Expected loss for choosing *broken*: $0.8 \cdot 1 + 0.2 \cdot 0 = 0.8$.

Choose *broken* in this case to minimises the expected loss!

Regression for Classification

Classification can be considered as a special case of regression.

Example.Classification problem with two classes coded as 0 and 1.

A classifier is a regression function that should only yield the values 0 and 1 (in the ideal case).

The regression function f derived from the might not yield the exact values 0 and 1. This could be amended by defining

$$\tilde{f}(x) = \begin{cases} 0 & \text{if } f(x) \leq 0.5 \\ 1 & \text{if } f(x) > 0.5 \end{cases}$$

Problem The objective function for regression problems (for example, the mean square error) does not aim at minimising the misclassification rate or a given cost function.

Measures of Interestingness

Fitting criteria for supervised learning problems (regression, classification), where the desired output for a given input is known (for the data), are usually based on error measures (mean square error, misclassification rate, expected loss,. . .).

For unsupervised learning tasks, where there is no desired output is specified in the data, error measure do not make sense.

For finding patterns,measures of interestingness are often used as score functions.

Example Find classification or association rules in the form of
if $A = a$ then $B = b$.

Statistical measures are often used in this case to evaluate the rule which is then considered as the *model*.

Measures of Interestingness

Let n_a, n_b be the number of instances where $A = a$ and $B = b$ respectively. n_{ab} is the number of instances where $A = a \wedge B = b$ and n is the size of the whole data set.

If $\frac{n_b}{n} \equiv \frac{n_{ab}}{n_a}$ holds true, the rule $if\, A = a$ then $B = b$ has no significance at all. Replacing records with $A = a$ by a random sample of size n_a would lead to the same expected fraction of record with $B = b$ in the sample.

The rule can be considered relevant if $\frac{n_b}{n} \gg \frac{n_{ab}}{n_a}$ holds.

Algorithms for Model Fitting

The objective function (scoring function) for models

does not tell us how to find the best or a good model,

it only provides a means for comparing models.

Optimisation algorithms to find the best or at least a good model are needed.

Closed form solutions In the best case, an explicit solution can be provided.
Example.Find a regression line $y = ax + b$ that minimises the mean square error for the data set $(x_1, y_1), \ldots, (x_n, y_n)$.
Computing partial derivatives of the objective (error) function

$$E(a, b) = \frac{1}{n} \sum_{i=1}^{n} (ax_i + b - y_i)^2$$

w.r.t. the paramaters a and b yields

$$\frac{\delta E}{\delta a} = \frac{2}{n} \sum_{i=1}^{n} (ax_i + b - y_i)x_i = 0,$$

$$\frac{\delta E}{\delta b} = \frac{2}{n} \sum_{i=1}^{n} (ax_i + b - y_i) = 0.$$

The solution here is

$$a = \frac{n \sum_{i=1}^{n} x_i y_i - n^2 \bar{x}\bar{y}}{n \sum_{i=1}^{n} x_i^2 - (n\bar{x})^2}, b = \bar{y} - a\bar{x}$$

where $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$ and $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$.

Algorithms for Model Fitting

Can be generalised to regression functions that are linear in the parameters to be optimised.

Constraints can be incorporated Lagrange functions or Kuhn-Tucker conditions.

For arbitrary non-linear regression functions, no closed form solution exists.

For differentiable score functions, a gradient descend approach can be applied. Problems: It will only find a local optimum and/or parameters have to be adjusted or computed in each iteration step.

For discrete problems with a finite search space (like finding association rules), combinatorial optimisation strategies are needed.

In principle, an exhaustive search of the finite domain M is possible, however, in most cases it is not feasible, since M is much too large. Heuristic strategies are therefore needed.

Algorithms for Model Fitting: Heuristic Strategies

Random search Create random solutions and choose the best one among them. *Very inefficient*

Greedy strategies Formulate an algorithm that tries to improve the solution in each step. E.g. gradient method or hill climbing:

- Start with a random solution, generate new solutions in the *neighbourhood* of the solution.
  If a new solution is better than the old one, generate new solutions in its *neighbourhood*.

Can find a solution quickly, but get stuck in local optima.

Simulated annealing is a mixture between random search and a greedy strategy. Simulated annealing is a modified version of hillclimbing, sometimes replacing better solutions by worse ones with a (low) probability. This probability is decreased in each iteration step.

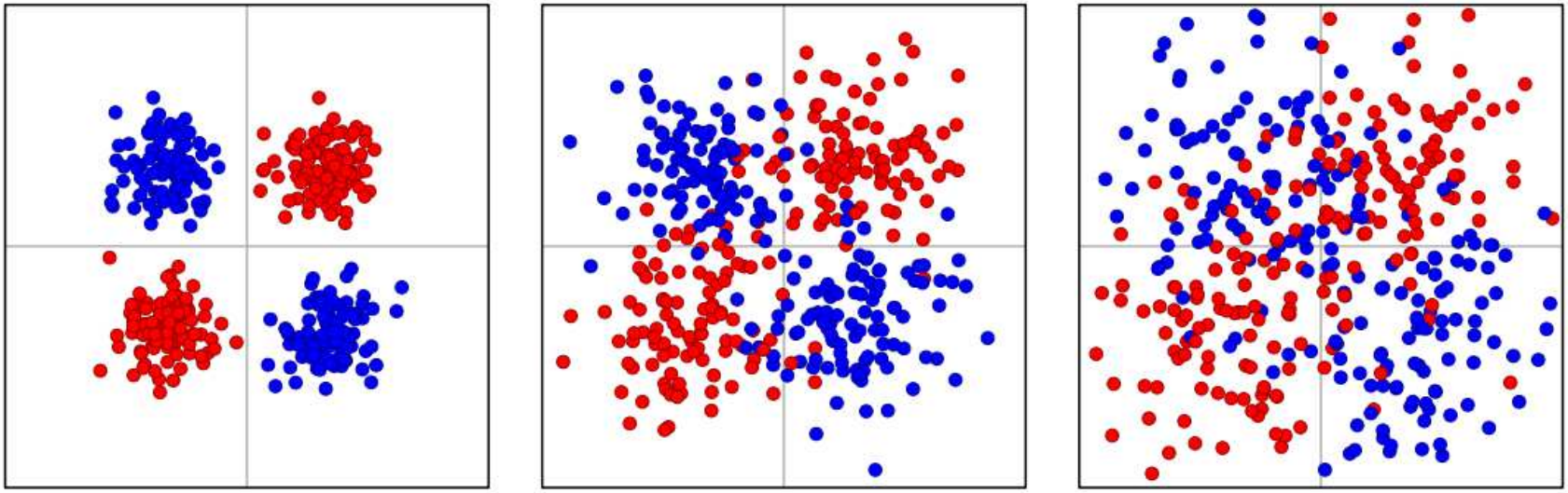
Evolutionary algorithms like evolution strategies or genetic algorithms combine random with greedy components, using a population of solutions in order to explore the search space in parallel and efficiently.

Algorithms for Model Fitting: Heuristic Strategies

Alternating optimisation can be applied when the set of parameters can be split into disjoint subsets in such a way that for each subset an analytical solution for the optimum can be provided, given the parameters in the other subsets are fixed. Alternating optimization computes the analytical solution for the parameter subsets alternatingly and iterates this scheme until convergence.

Bayes Error

In the context of classification problems the *intrinsic error* is also called **Bayes Error**.



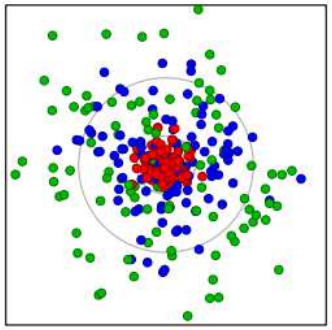
In all three cases there are two *typical prototypes* for each class:

Lower left and upper right corner for the red class.

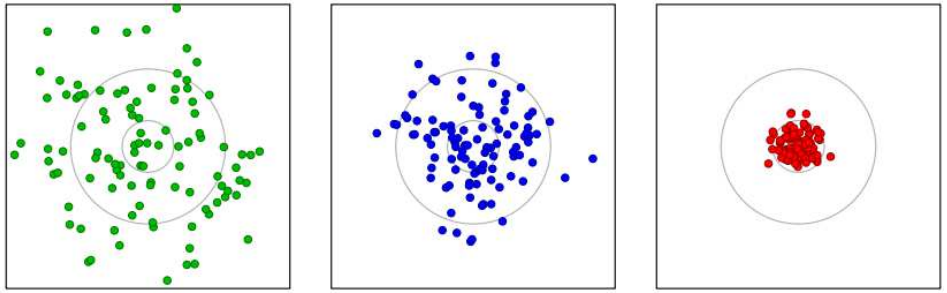
Upper left and lower right corner for the blue class.

There are not always easily distinguishable prototypes for different classes!

Bayes Error



Results from different darts players: A beginner (green), a hobby player (blue) and a professional player (red).



All three classes have the same *prototype*, since all players want to hit the centre of the dart board. The classes differ only in their variances, not in their *prototype*.

Evaluating a Classifier

Cost matrices can also be used to evaluate a classifier.

What if no explicit cost matrix is known/given and the misclassification rate cannot be used as evaluation criterion?

Consider a two class problem.

- Very often, classifiers provide a score for each class. In the example of the three dart players, the scores would be the likelihoods based on the location of the dart.
- In the simplest case, an object is assigned to the class with the best score.
- In the case of a cost matrix with very different entries, the class with the highest probability (score) might not be the one minimising the expected loss.

ROC Curves

Example Assume, only the attribute *Sepal Length* should be used to distinguish *Iris versicolor* from *Iris virginica* by a simple rule of the form

If Sepal Length $< c$, then versicolor, otherwise virginica

where c can be chosen in the range of the values of the attribute *Sepal Length*. The attribute *Sepal Length* provides the **score** in this case.

Consider versicolor as the *positive* and virginica as the *negative* class.

The higher the threshold c is chosen, the more instances are classified as *positive* (versicolor).

Those instances classified as *positive (negative)* that belong to the class *positive (negative)*, are called **true positives (negatives)**.

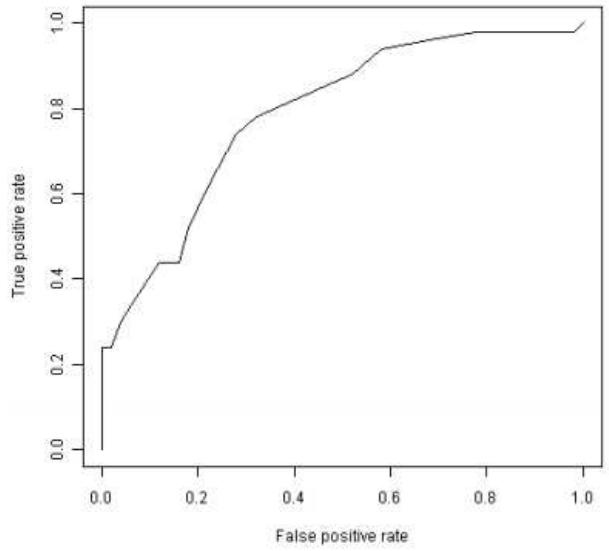
Those instances classified as *positive (negative)* that belong to the class *negative (positive)*, are called **false negatives (positives)**.

Increasing the threshold c, will increase the true positives as well as the true negatives.

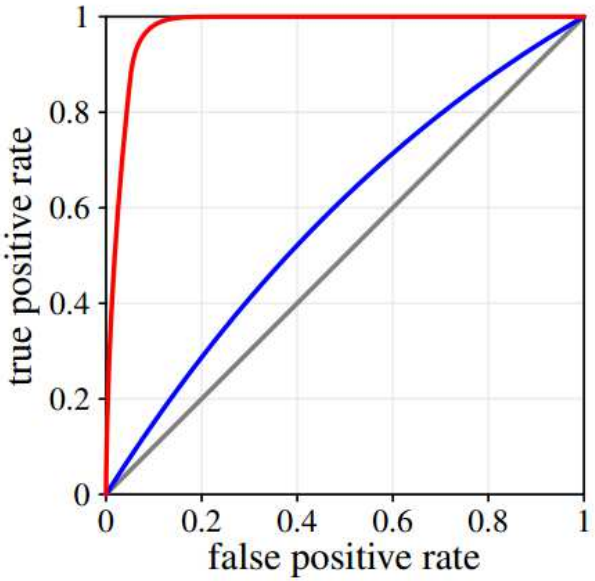
In the ideal case, we would first get only true positives and no false negatives (if the sepal length of versicolor would always be smaller than the sepal length of virginica).

ROC Curves

The ROC curve (receiver operating characteristic curve) draws the false positive rate against the false negative rate (depending on the choice of the threshold c).



ROC Curves



A ROC curve of a better performing (red) and a classifier with a performance (blue).

The area under curve(AUC), i.e. the area under the ROC curve, is an indicator how well the classifier solves the problem. The larger the area, the better the solution for the classification problem.

However, for comparing classifiers, AUC can lead to inconsistent results.

Confusion Matrix

A confusion matrix is a table where the rows represent the true classes and the columns the predicted classes. Each entry specifies how many objects from a given class are classified into the class of the corresponding column.
Below is a possible confusion matrix for the Iris data set.

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

Machine Learning Bias and Variance

In statistics, the notions are often used in a slightly different way. The mean squared error (MSE) of an estimator θ^* for an unknown parameter θ can be decomposed in terms of the variance of the estimator and its bias:

$$MSE = Var(\theta^*) + (Bias(\theta^*))^2.$$

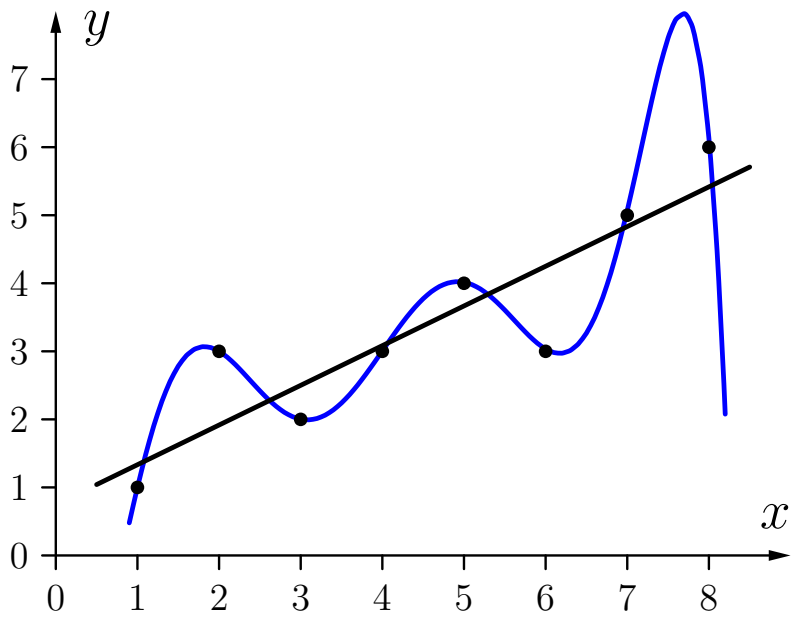
Here the variance depends on the intrinsic error, i.e. on the variance from the random variable from which the sample is generated, but also on the choice of the estimator θ^* which is considered as part of the model bias in machine learning.

Model Validation

Complex models can satisfy a simple fitting criterion better than simple models.

But too complex models tend to overfitting.

Especially for prediction tasks, a complex model might work perfectly for the data with which it was trained, but might predict future values incorrectly.



black line:
regression line
(2 free parameters)

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

Training and Test Data

To avoid overfitting and to have a more realistic estimation for the error that the model will make for new data, the data set is often split into *training data* and *test data*.

The model is computed (*trained*) only with the training data.

The test data set is used to evaluate the model (for instance, to calculate the misclassification rate).

Typically, 2/3 of the data are used for training and 1/3 for testing. Instances are assigned randomly to the training and the test data. For classification problems, *stratification* is often applied, guaranteeing that the training and test data have the distribution of classes as the original data.

Cross-Validation

In order to avoid random effects by splitting the data into training and test data,cross-validationis applied.

For k-fold **cross-validation**, the data set is split into k disjoint subsets of (roughly) equal size.

The training and testing is carried out k times.

Each time, one of the k subsets is used as the test set, whereas the data from the other $(k-1)$ subsets are used for training.

In this way, the error for the model can be computed k times.

The estimation of the error is the mean value of these k errors. Usually, $k = 10$ is chosen.

When the data set is very small and (almost) all instances are needed for training the **leave-one-out** or **jackknife** method is applied:
n-fold cross-validation, where n is the number of instances in the data set.

Bootstrapping

Bootstrapping is a resampling technique from statistics that does not directly evaluate the model error, but aims at estimating the variance of the estimated model parameters.

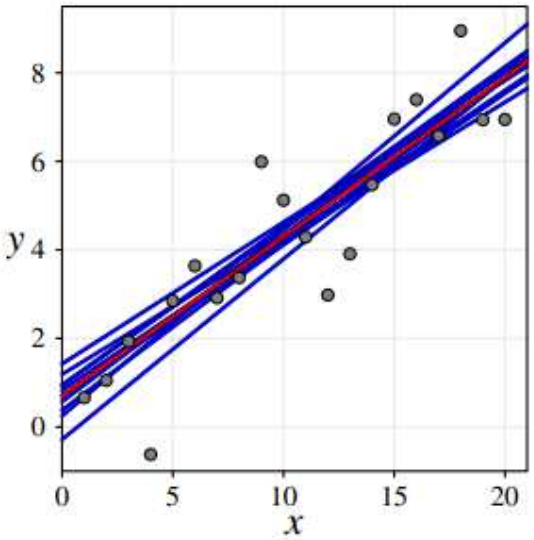
k bootstrap samples, each of size n, are drawn randomly **with replacement** from the original data set with n instances.

The model is fitted to each of these k samples, resulting in k estimates for the model parameters.

Based on these k estimates the empirical standard deviation can be computed for each parameter to provide information how reliable the estimation of the parameter is.

Bootstrapping

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



Measures for Model Complexity

Model selection - the choice of a suitable model - requires a trade-off between simplicity and (over-)fitting.

Based on the principle of **Occam's razor**, one should choose the simplest model that *explains* the data.

A main problem that the scoring function for model fitting - often an error measure - and measures for model complexity are not based on the unit and it is not clear how to combine them.

Information Criteria

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

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

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

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

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

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

Special cases:

- **Akaike Information Criterion** (AIC): $\kappa = 2$
- **Bayesian Information Criterion** (BIC): $\kappa = \ln n$, where n is the sample size.

The lower the value of these measures, the better the model.

Minimum Description Length

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

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

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

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

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

$$\begin{aligned} \textbf{description length} &= \textbf{length of model description} \\ &+ \textbf{length of data description} \end{aligned}$$

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

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

Minimum Description Length: Example

Given: a one-dimensional sample from a polynomial distribution.

Question: are the probabilities of the attribute values sufficiently different to justify a non-uniform distribution model?

Coding using **no model** (equal probabilities for all values):

$$l_1 = n \log_2 k,$$

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

Coding using a **polynomial distribution model**:

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

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

Minimum Description Length: Example

Some details about the codebook idea:

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

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

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

Data Description:
There are k groups of objects with x_i, $i = 1, \ldots, k$, elements in them.
(The values of the x_k are known from the model description.)

$$\text{Number of possible sequences:} \quad \frac{n!}{x_1! \ldots x_k!}$$