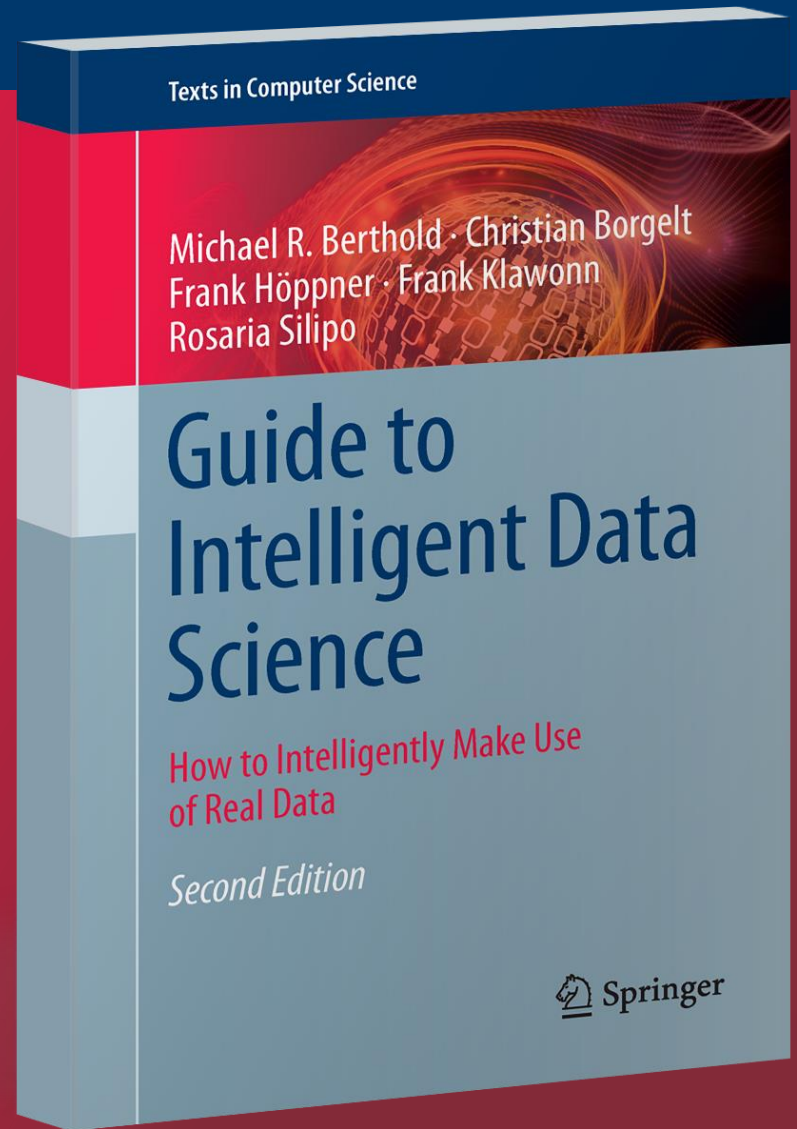


Basic Principles of Machine Learning



„Machine intelligence is the last invention that humanity will ever need to make“

-Nick Bostrom

What are common aspects of every Machine Learning algorithm?

**This lesson refers to chapter 5 of the GIDS book*

- Basic strategies for ML algorithms
 - Close form solutions
 - Gradient descent
 - Search strategies
- Error measures to validate models
 - Confusion matrix
 - Class statistics measures
 - Accuracy measures
 - ROC curves
 - Cohen's kappa
 - Numeric error measures
- Model validation
 - Types of model errors
 - Strategies for more reliable model evaluation
 - Cross-validation
 - Bootstrapping
 - Coping with unbalanced datasets
 - Measures for model complexity

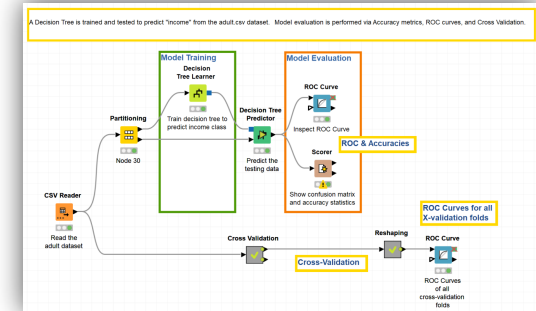
Datasets

– Datasets used : adult dataset and iris dataset

– Example Workflows:

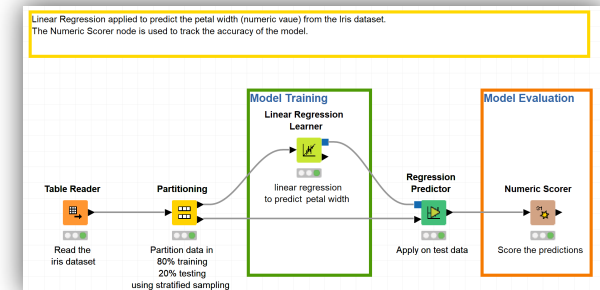
– „Training and testing a model (decision tree)“ <https://kni.me/w/-0nN9BzUOCi6vCXI>

- confusion matrix
- accuracy measures
- ROC curve
- cross-validation



– „Numeric Error Measures to score numeric predictions“ <https://kni.me/w/KGDQtyTZ4FPCgXrH>

- Partitioning
- Numeric error measures



Select the Model

What's the best model to use?

From the Data:

- Classification vs. Numerical
- Supervised vs. Unsupervised

Finding the “best” model is not a trivial task at all, since the question what a good (or best) model means is not always easy to answer.

From the business case:

- Performances: what is acceptable?
- Simplicity: do not use a cannon for a simple problem
- Interpretability: do I need to know the decision process?
- Computational costs: it must be trainable and applicable in a reasonable time with reasonable hardware

Acceptable performances depend on:

- The adopted score function
- Our tolerance to errors

True class	Predicted class	
	Positive	Negative
Positive	14	6
Negative	5	75

overall accuracy = 0.89

Cohen's Kappa \approx 0.65

Similar overall
accuracy
But different
Cohen's Kappa

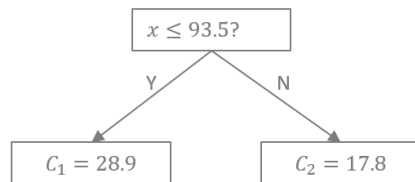
True class	Predicted class	
	Positive	Negative
Positive	6	14
Negative	5	75

overall accuracy = 0.81

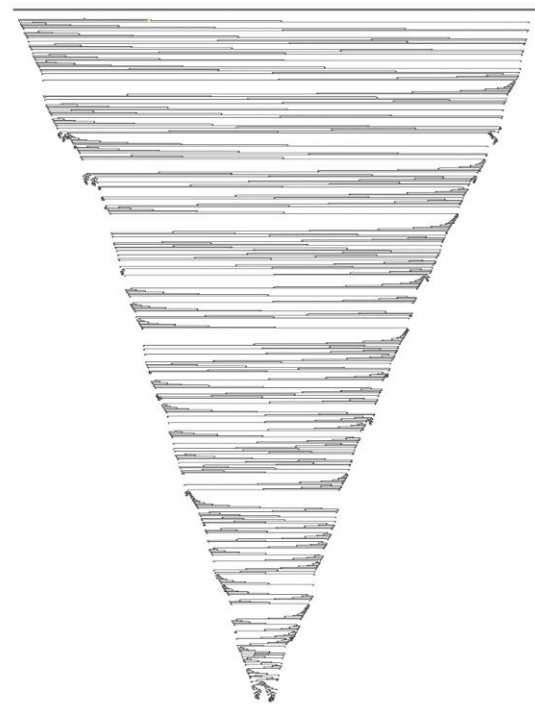
Cohen's Kappa = 0.29

Do not use a cannon to shoot a fly!

- Simple models are usually easier to understand and to interpret.
- Their computational complexity is lower.
- Too complex models often lead to overfitting.



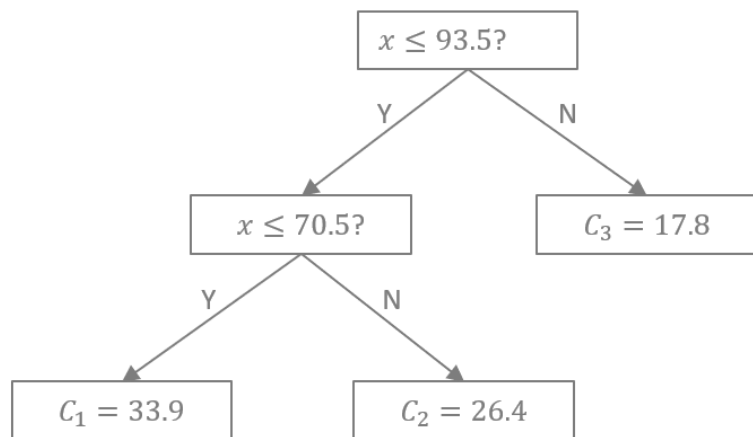
VS.



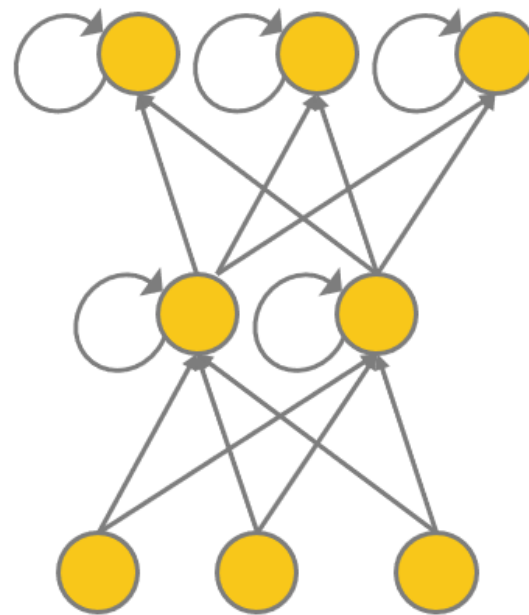
For some use cases interpretability is a must

For some other use cases interpretability is not necessary

Interpretable model vs. Black-box

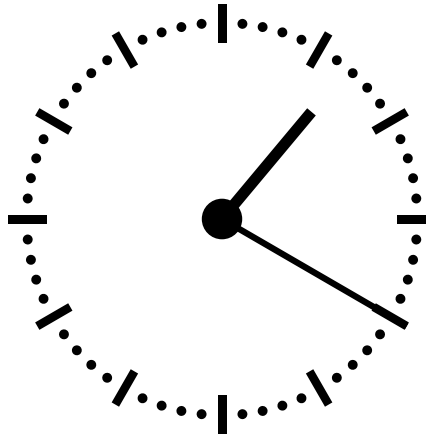


VS.



Computational costs mean:

- More advanced (and expensive) hardware
- More time spent on training the model (possibly multiple times)



Select the score function

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

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 $f: M \rightarrow \mathbb{R}$.

Fitting the model M means to choose its parameters as to minimize the function f (**error**) or to maximize the function f (**likelihood**).

Classic Error function is the Mean Square Error (MSE):

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - M(x_i))^2$$

Where (x_i, y_i) are the n points in the dataset and M is the model.

Variations of MSE are also used as error functions to fit models.

Note that MSE and its variations work for model classes with numeric outputs.

- Classic Error function for classification is the **misclassification rate**.
- Misclassification rate is the proportion of records wrongly assigned to a class.
- Misclassification rate can be associated to a **cost matrix**

Tea cup misclassification cost matrix:

True class	Predicted class	
	OK	broken
OK	0	c1
broken	c2	0

- c1 = cost of misclassification of good cup for broken cup, i.e. cost of cup to be trashed
- c2 = cost of misclassification of broken cup for good cup, i.e. new cup + unhappy customer

Classic loss function for classification:

$$Loss(c_j | \mathbf{x}) = \sum_{j=1}^m P(c_j | \mathbf{x}) c_{ji}$$

Where:

- (\mathbf{x}, c_k) is a point in the dataset with input features \mathbf{x} and true class c_k
- $P(c_j | \mathbf{x})$ the probability of input vector \mathbf{x} to be assigned to class c_j
- c_{ji} the misclassification cost of assigning input vector \mathbf{x} to class c_i instead of the correct class c_j
- m the number of classes

The loss function can be extended to the sum on all samples in the dataset

Algorithms for model fitting

Fitting the model M means to choose its parameters as to minimize the function f (**error**) or to maximize the function f (**likelihood**).

In the best case, a closed-form solution P^* for the optimization problem can be obtained directly from the system of equations:

$$\frac{\partial E}{\partial P} = 0$$

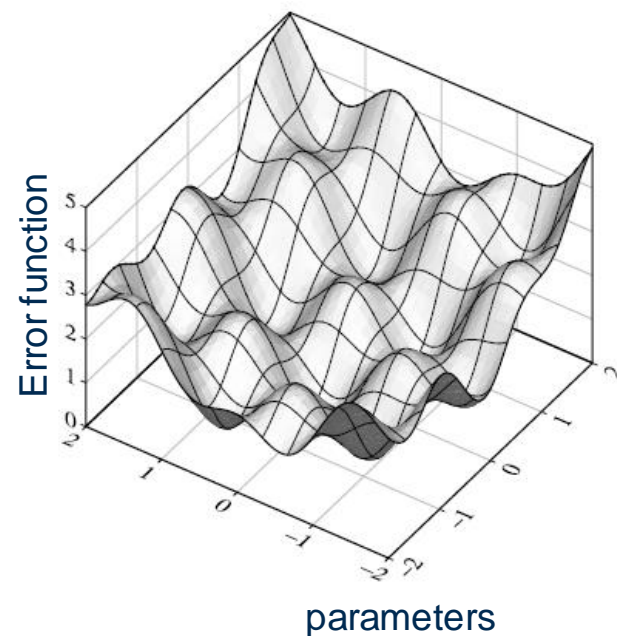
Where E is the error function, P the parameter set of the model, and P^* the solution parameters for the equation above.

This is the case, for example, of linear regression for the MSE as error function.

It is not always possible to find a closed-form solution, due to:

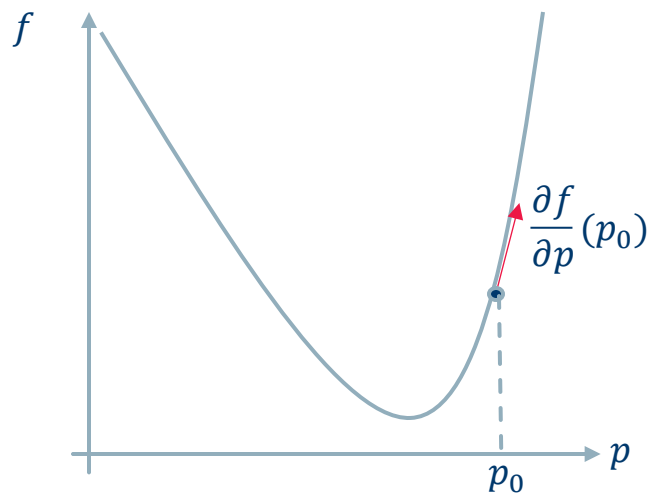
- Many more model parameters ($k > 2$)
- More complex algorithms than the linear regression
- Presence of local minima in the error function
- $f: \mathbb{R}^k \rightarrow \mathbb{R}$

When the objective function is differentiable,
a **gradient method** can be applied.



The gradient, i.e., the vector of partial derivatives of the objective function with respect to the model parameters, points in the direction of steepest ascend.

- The gradient is the vector of partial derivatives: $\left[\frac{\partial f}{\partial p_1}, \frac{\partial f}{\partial p_2}, \frac{\partial f}{\partial p_3}, \dots, \frac{\partial f}{\partial p_k}\right]$
- The gradient points in the direction of steepest ascend.



The gradient, i.e., the vector of partial derivatives of the error function with respect to the model parameters, points in the direction of steepest ascend.

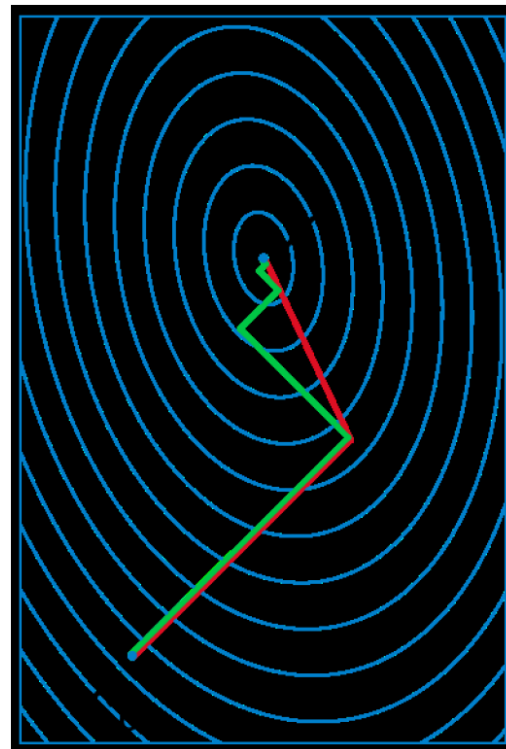
1. Start at a random point $p_{i=0}$, i.e. an arbitrary choice of the model parameters
2. Calculate the gradient of the objective function $[\frac{\partial f}{\partial p_1}, \frac{\partial f}{\partial p_2}, \frac{\partial f}{\partial p_3}, \dots, \frac{\partial f}{\partial p_k}]$ in p_i
3. From p_i move a certain step in the opposite direction of the gradient (for an error function to minimize) and reach point p_{i+1}
4. Calculate the new value of the objective function in $f(p_{i+1})$
5. Repeat from 2.

Procedure continues until no more improvements on the objective function $f(p)$ can be achieved (step 4), a fixed number of gradient steps has been carried out, or p_{i+1} is too close to p_i .

Notes:

- f must be differentiable
- The landscape of the objective function (f vs. P) cannot be plotted already for $k > 2$
- The gradient descent procedure moves in steps along the objective function
- The step size: constant or adaptive?
- Problem of the local minima (maxima)

It is recommended to run a gradient method repeatedly, starting with different initial points to increase the chance to find the global or at least a good local optimum.



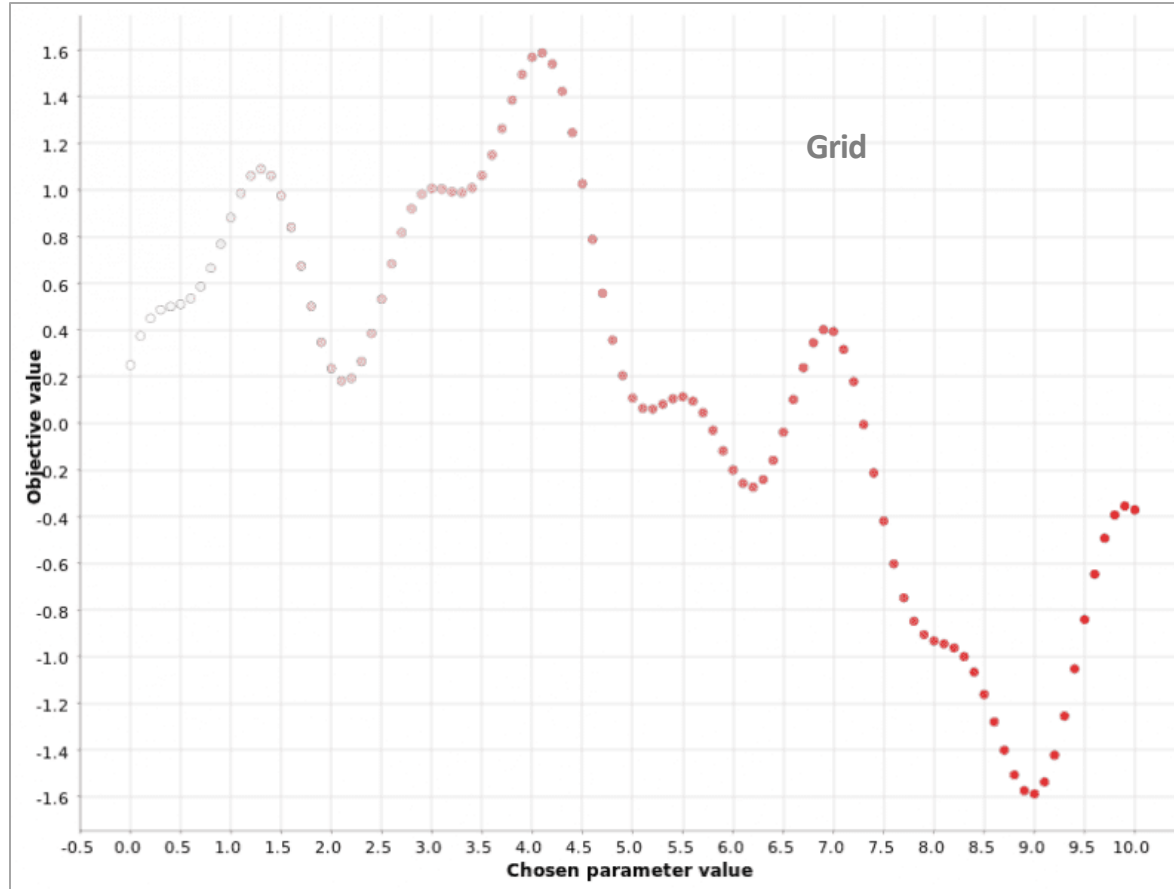
If objective function f is not differentiable, we can proceed through some search procedure for the best (good) parameter set.

- Grid search
- Random search
- Hill climbing
- Bayesian optimization

Example:

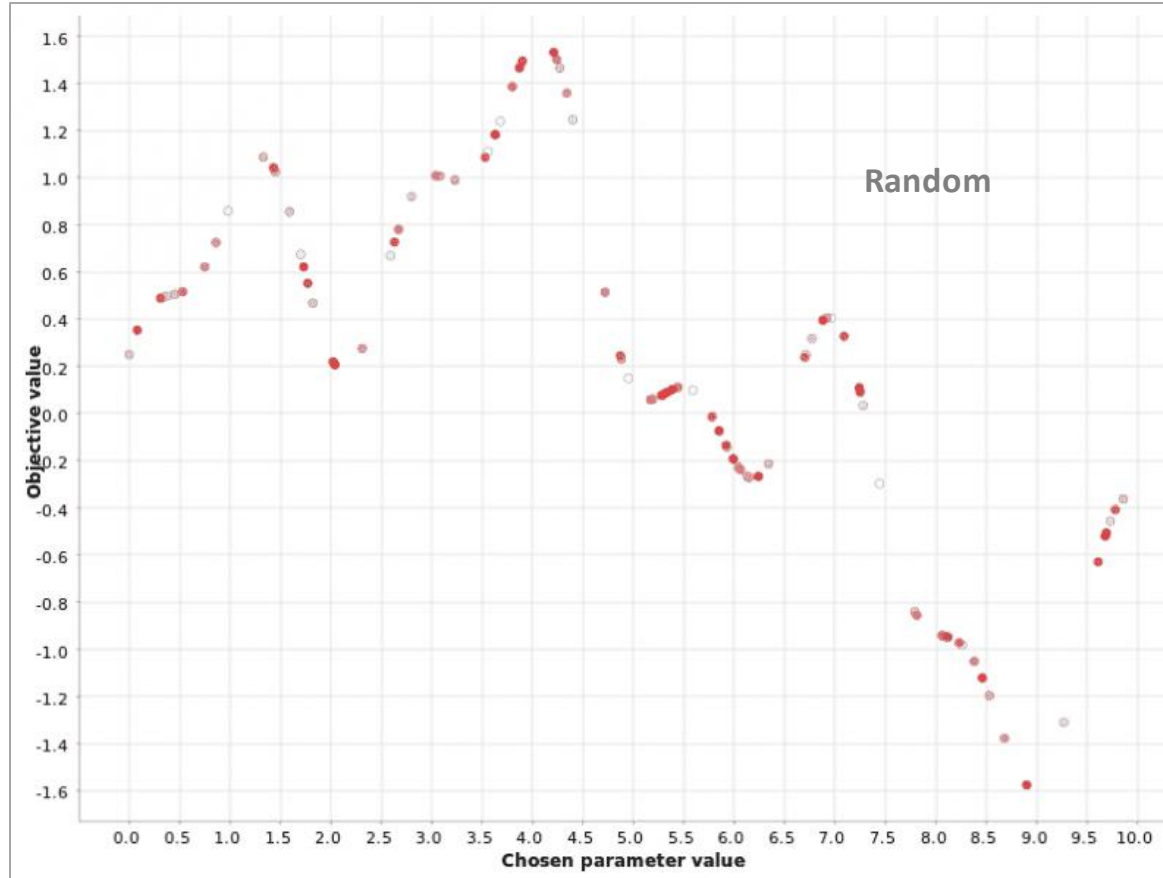
$$f(p) = \sin\left(\frac{p}{2}\right) + 0.5 \sin(2p) + 0.25 \cos(4.5p)$$

Grid Search



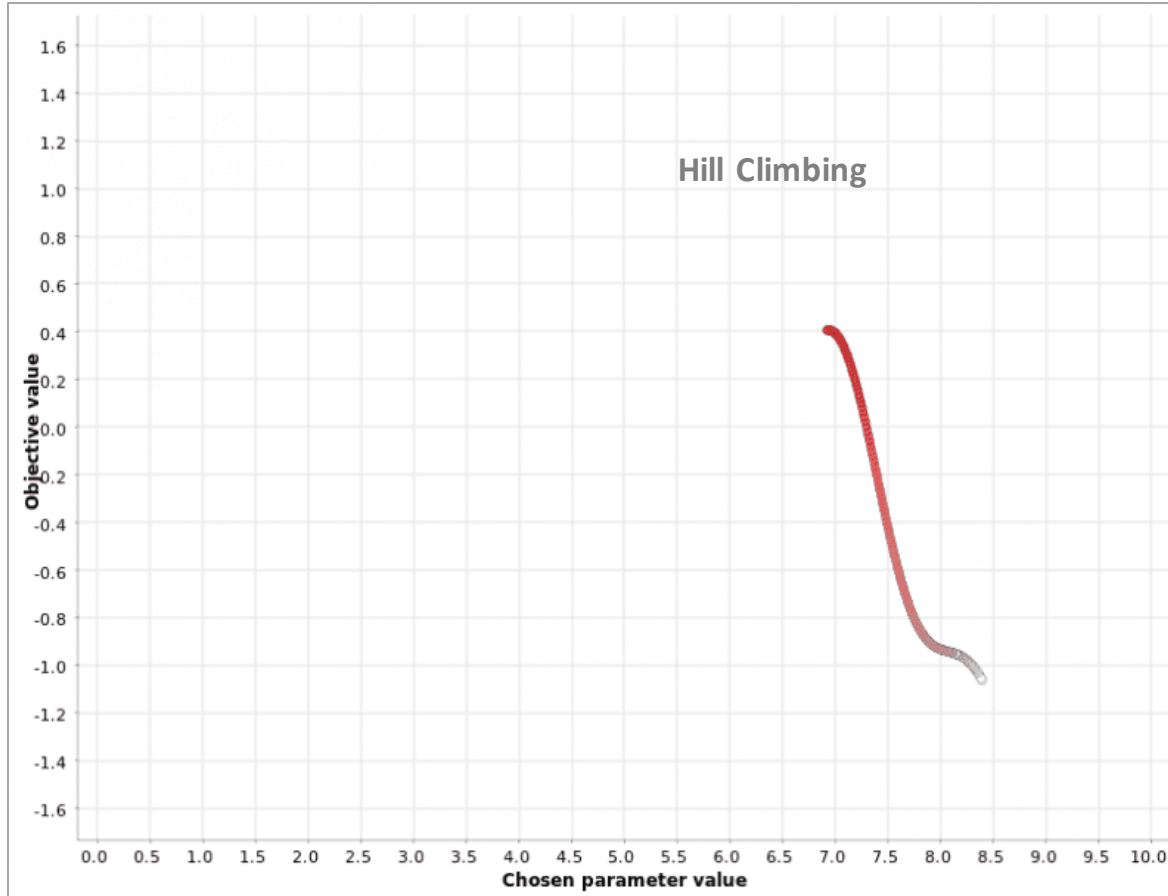
- Brute-force strategy
- If we do not know which value, we try them all
- At the end we find the optimum
- In figure:
 - range [0,10]
 - step size 0.1
 - Start point 0
 - Whiter points generated earlier
 - Global optimum at $p = 4.1$

Random Search



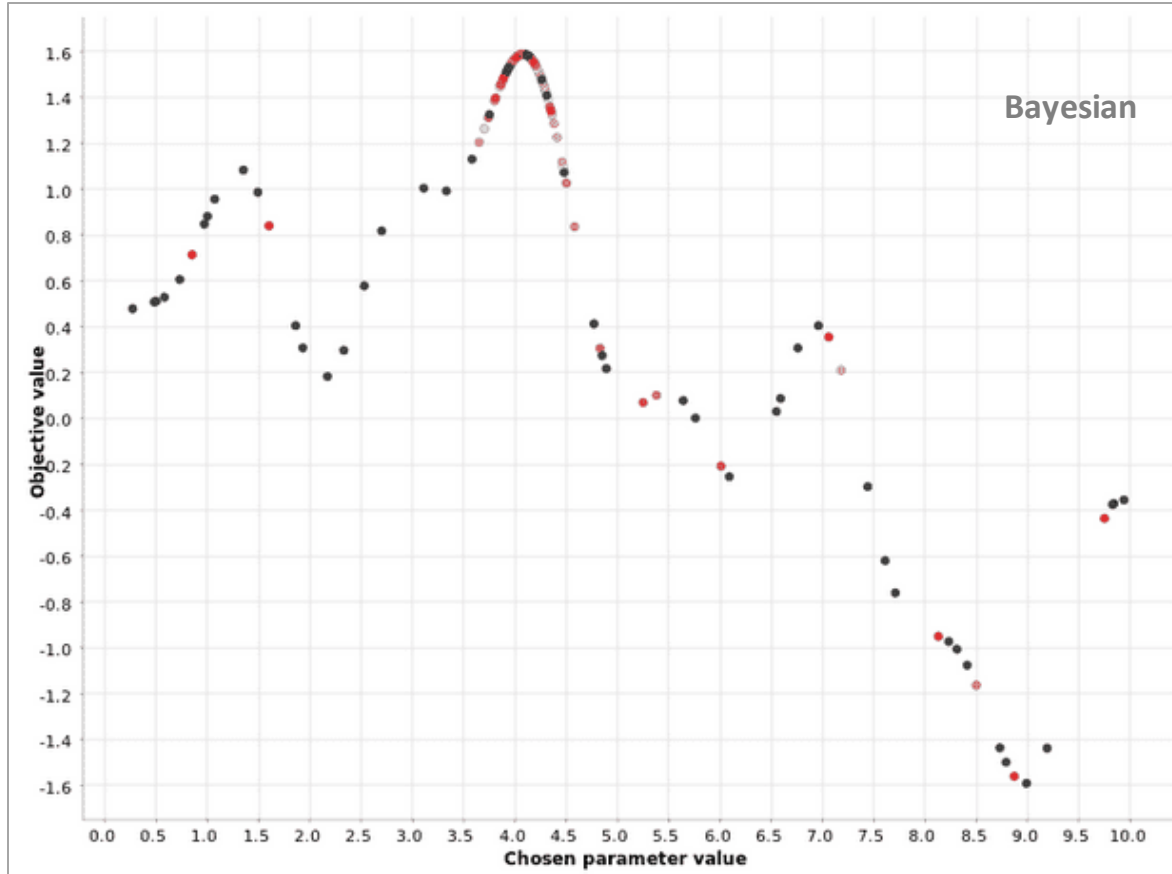
- If we do not know which value, we try N points at random
- At the end we find the optimum
- Faster, esp. on many parameters
- In figure:
 - range [0,10]
 - Start point 5.0
 - Whiter points generated earlier
 - Optimum at $p = 4.3$

Hill Climbing



- Greedy Strategy
- We try points in the neighborhood
- If no points in the neighborhood do better, we stop
- Faster but local optima
- In figure:
 - range [0,10]
 - Start point 8.5
 - Whiter points generated earlier
 - Converging to optimum at $p \sim 7.0$

Bayesian Optimization



- Phase 1 (warm up): Random points
- Phase 2: surrogate model from random points
 $P(\text{output} | \text{past points})$
- Optimize surrogate model
- In figure:
 - range [0,10]
 - Gray points in warm up phase
 - Whiter points generated earlier
 - Converging to optimum at $p \sim 4.0$

Validate Results: Errors

- Pure / Experimental / Intrinsic Error (Bayes Error)

The **pure error** or **experimental error** is inherent in the data and is due to noise, random variations, imprecise measurements, or the influence of hidden variables that cannot be observed.

- Sample Error

The **sample error** is caused by the fact that a finite sample, especially when its size is quite small, will seldom exactly reflect the true distribution of the probability distribution generating the data (example of throwing a dice).

Variance

- Model Error

A large error may be caused by a **lack of fit**. When the set of considered models is too simple for the structure inherent in the data, no model will yield a small error. Too complex models might run into overfitting.

Machine Learning Bias

- Algorithmic Error

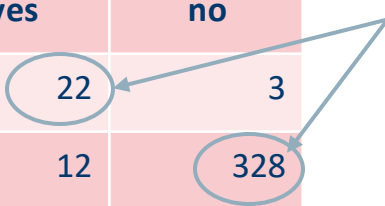
The **algorithmic error** is caused by the method that is used to fit the model. Usually this is ignored.

Confusion Matrix for Classification

- Confusion matrix is a matrix-like representation of the correctness of the classification model
- A binary classification problem: yes / no classes
- The matrix below summarizes the model predictions

True class	Predicted class	
	yes	no
yes	22	3
no	12	328

Correct predictions



Arbitrarily we take one class as the **POSITIVE** class and the remaining class as the **NEGATIVE** class, then:

TRUE POSITIVES (**TP**): True and predicted class is positive

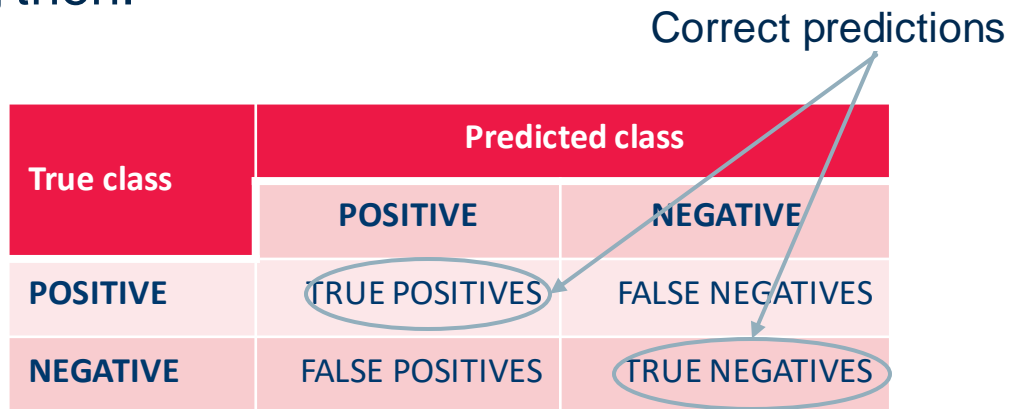
TRUE NEGATIVES (**TN**): True and predicted class is negative

FALSE NEGATIVES (**FN**): True class is positive and predicted negative

FALSE POSITIVES (**FP**): True class is negative and predicted positive

Correct predictions

True class	Predicted class	
	POSITIVE	NEGATIVE
POSITIVE	TRUE POSITIVES	FALSE NEGATIVES
NEGATIVE	FALSE POSITIVES	TRUE NEGATIVES



Most evaluation metrics are based on the confusion matrix

– Sensitivity vs. Specificity

$$\text{sensitivity} = \frac{TP}{TP + FN}$$

Ratio of samples
in the positive
class correctly
classified

$$\text{specificity} = \frac{TN}{TN + FP}$$

Ratio of samples
in the negative
class correctly
classified

– Precision vs. Recall

$$\text{recall} = \frac{TP}{TP + FN}$$

Same as
sensitivity

$$\text{precision} = \frac{TP}{TP + FP}$$

Ratio of samples
correctly classified
among those assigned
to the positive class

Both measures must be high!

- F-Measure

$$F = 2 \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}$$

- Overall Accuracy

$$\textit{accuracy} = \frac{TP + TN}{\underbrace{TP + TN + FP + FN}_{\text{All samples}}} = \frac{TP + TN}{n}$$

All correctly classified samples

- Cohen's Kappa

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

Cohen's Kappa

Overall
accuracy

$$\kappa = \frac{p_o - p_e}{1 - p_e}$$

Introduces the a
priori probabilities
of the classes

$$p_e = p_{pos} + p_{neg}$$

All samples in
the positive class

$$p_{pos} = \frac{TP + FN}{n} \cdot \frac{TP + FP}{n}$$

All samples assigned
to the positive class

All samples in the
negative class

$$p_{neg} = \frac{TN + FP}{n} \cdot \frac{TN + FN}{n}$$

All samples assigned
to the negative class

Perfectly correct classification $\kappa = 1$
Perfectly wrong classification $\kappa = 0$

Cohen's Kappa (κ) vs. Overall accuracy

	Positive	Negative
Positive	14	6
Negative	5	75

$$p_{e1} = \frac{19}{100} \times \frac{20}{100}$$

$$p_{e2} = \frac{81}{100} \times \frac{80}{100}$$

$$p_e = p_{e1} + p_{e2} = 0.686$$

$$p_0 = \frac{89}{100} = 0.89$$

$$\kappa = \frac{p_0 - p_e}{1 - p_e} = \frac{0.204}{0.314} \approx 0.65$$

$\kappa = 1$: perfect model performance
 $\kappa = 0$: the model performance is equal to a random classifier

	Positive	Negative
Positive	6	14
Negative	5	75

$$p_{e1} = \frac{11}{100} \times \frac{20}{100}$$

$$p_{e2} = \frac{89}{100} \times \frac{80}{100}$$

$$p_e = p_{e1} + p_{e2} = 0.734$$

$$p_0 = \frac{81}{100} = 0.81$$

$$\kappa = \frac{p_0 - p_e}{1 - p_e} = \frac{0.076}{0.266} = 0.29$$

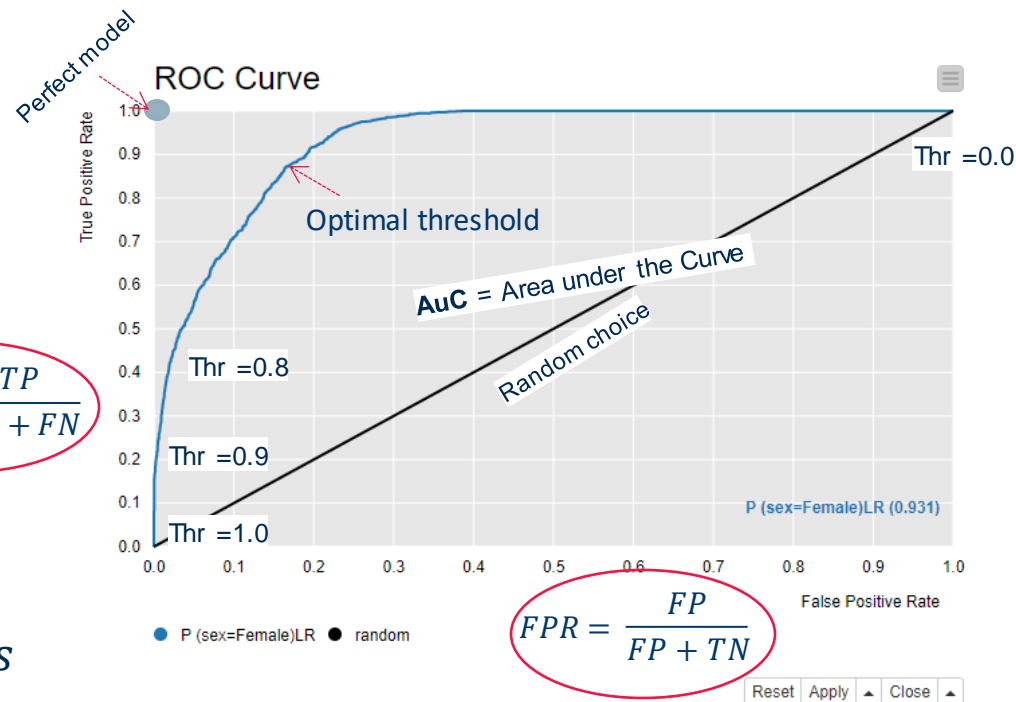
Classification model performance as reported by:

- False positive rate (FPR)
 - negative events **incorrectly** classified as positive
- True positive rate (TPR)
 - positive events correctly classified as positive

$$TPR = \frac{TP}{TP + FN}$$

$P(pos|x) > threshold \Rightarrow class\ pos$

True class	Predicted class	
	POSITIVE	NEGATIVE
POSITIVE	TRUE POSITIVES	FALSE NEGATIVES
NEGATIVE	FALSE POSITIVES	TRUE NEGATIVES



$$FPR = \frac{FP}{FP + TN}$$

Numeric Error Measures

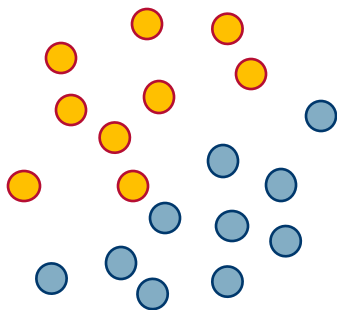
Error Metric	Formula	Notes
R-squared	$1 - \frac{\sum_{i=1}^n (y_i - f(x_i))^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$	Universal range: the closer to 1 the better
Mean absolute error (MAE)	$\frac{1}{n} \sum_{i=1}^n y_i - f(x_i) $	Equal weights to all distances Same unit as the target column
Mean squared error (MSE)	$\frac{1}{n} \sum_{i=1}^n (y_i - f(x_i))^2$	Common loss function
Root mean squared error (RMSE)	$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - f(x_i))^2}$	Weights big differences more Same unit as the target column
Mean signed difference	$\frac{1}{n} \sum_{i=1}^n (y_i - f(x_i))$	Only informative about the direction of the error
Mean absolute percentage error (MAPE)	$\frac{1}{n} \sum_{i=1}^n \frac{ y_i - f(x_i) }{ y_i }$	Requires non-zero target column values

Validate Results: Model Validation

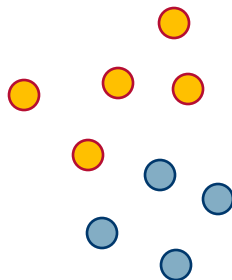
The most common principle to estimate a ***realistic performance*** of the model for unknown or future data is separating the data set for training and testing purposes:

- ***Training phase***: the algorithm trains a model using the data in the training set
- ***Testing phase***: a metric measures how well the model is performing on data in a new dataset (the test set)

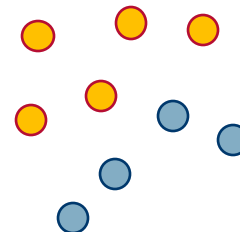
Training Set



Evaluation Set*



Test Set



* sometimes

By chance, we might be lucky and the test set contains easy examples leading to an overoptimistic evaluation of the model. Or we might be unlucky and the test set contains difficult examples and the performance of the model is underestimated.

We can get a more realistic estimation of the model quality, by repeating the test many times on different test sets.

- For k -fold cross-validation, the data set is partitioned into k subsets of approximately equal size
- The first of the k subsets is used as a test set, and the other $(k - 1)$ sets are used as training data for the model
- This procedure is repeated by using each of the other k subsets as test data and the remaining $(k - 1)$ subsets as training data
- Altogether, we obtain k estimates for the model error. The average of these values is taken as the estimate for the model error. The variance is taken as an estimate for the quality of the underlying data.
- Typically, $k = 10$ is chosen.

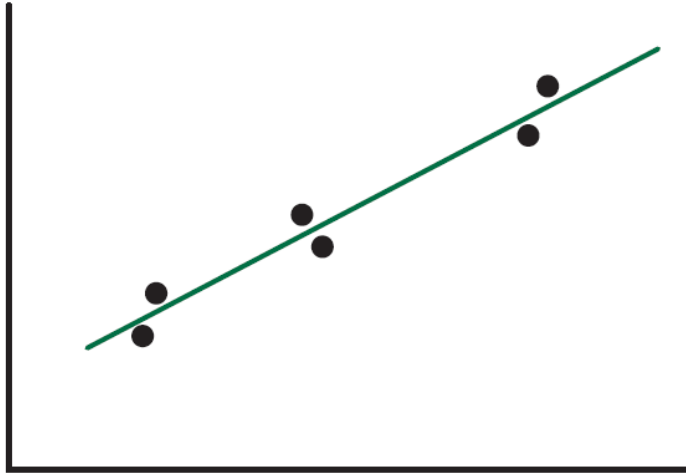
- Bootstrapping is a resampling technique from statistics aiming at estimating the variance of the model parameters.
- Like in cross-validation, the model is computed multiple (k) times on different data sets.
- k bootstrap samples, each of size n , are drawn randomly *with replacement* from the original data set with n records.
- The model is then fitted to each one of these bootstrap samples, so that we obtain k estimates for the model parameters.
- Based on these k estimates, the empirical standard deviation is computed for each parameter to provide information on how reliable the estimate of the parameter is.

- Sampling with **stratification** means that random assignments of the data to the test and training sets are carried out per class and not simply for the whole data set. This is to ensure that the relative class frequency in the original data set, the training, and the test set are the same.
- Sometimes we want to force a model to represent the classes in the datasets equally even though they are not equally represented. **Under-sampling** and **over-sampling** techniques can produce a new dataset with equally distributed classes, though with unrealistic a priori probabilities.
 - **Under-sampling** techniques randomly remove samples from the majority class to have as many samples as the minority class. Attention! It may discard potentially useful information!
 - **Over-sampling** supplements the datasets with multiple copies of samples from the minority class – with or without replacement - as to reach the same number as in the majority class. Instead of using just copies of existing samples, some oversampling techniques create synthetic samples from the same statistical distribution of the minority class. The most common technique is called **SMOTE** (Synthetic Minority Oversampling Technique).

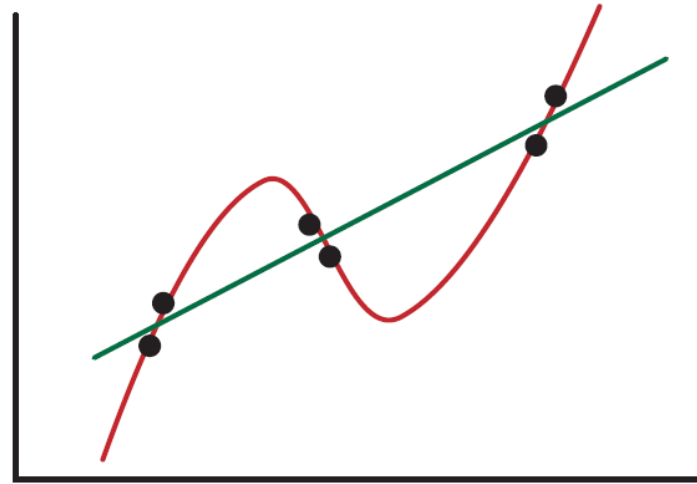
- Considering a data set as a collection of examples - describing the dependency between the predictor variables and the dependent variable - the model should “learn” this dependency from the data and **generalize** it in order to make correct predictions on new data.
- To achieve this, the model must be universal (flexible) enough to be able to learn the dependency.
- This does not mean that a more complex model with more parameters leads to better generalization than a simple one.
- Complex models can lead to **overfitting**.

Keep it simple!

Generalized



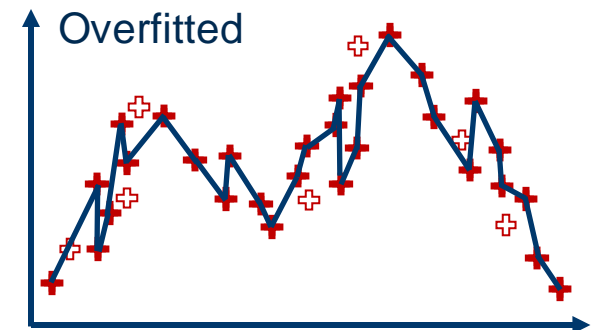
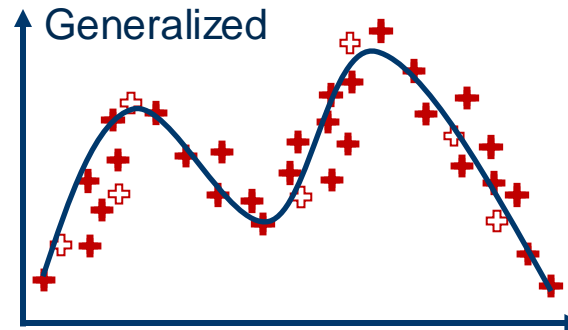
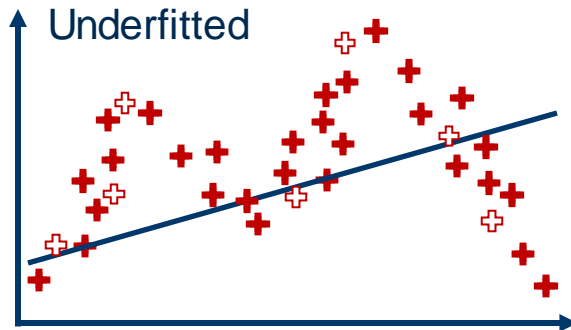
Overfitted



- The model must “learn” a description of the data, not of the details and noise inherent in the data.
- The prediction using a complex function can be worse than for a simpler model.

Overfitting vs Underfitting

Overfitting	Underfitting
<ul style="list-style-type: none">Model that fits the training data too well, including details and noiseNegative impact on the model's ability to generalize	<ul style="list-style-type: none">A model that can neither fit the training data nor generalize to new data



Validate Results: Model Complexity

- Based on the principle of **Occam's razor**, one should choose the simplest model that “explains” the data. If a linear function fits the data well enough, one should prefer the linear function and not a quadratic or cubic function.
- There is a need for a trade-off between model simplicity and model fit.
- **Regularization** is a general mathematical concept that introduces additional information in order to solve an otherwise ill-posed problem.
- Here a penalty term for more complex models is incorporated into the pure measure for model fit as a regularization term.
- By favouring simpler models, regularization also leads to avoiding overfitting.

- The **minimum description length principle (MDL)** attempts to join measures for model fit and complexity into one single measure.
- The basic idea behind MDL is to understand modelling as a technique for data compression.
- To recover the original data, the compressed data and decompression rule are needed. Therefore, the overall size of the compressed data file is the sum of the bits needed for the compressed data and the bits needed to encode the decompression rule.
- The model can be interpreted as a compression or decompression scheme. That is, to represent it we need the (binary) coding of the model and the compressed (binary) coding of the data (or of the errors).
- Models with smallest MDL offer the best compromise between model error and model size.

- Akaike's Information Criterion (AIC)

$$AIC = 2k - 2 \ln(L)$$

Where k = number of model parameters and L = value of the likelihood function.

- Bayesian Information Criterion (BIC)

$$BIC = k \ln(n) - 2 \ln(L)$$

Where n = number of samples in dataset

- Basic strategies for ML algorithms

- Closed form solutions
- Gradient descent
- Search strategies

- Error measures to validate models

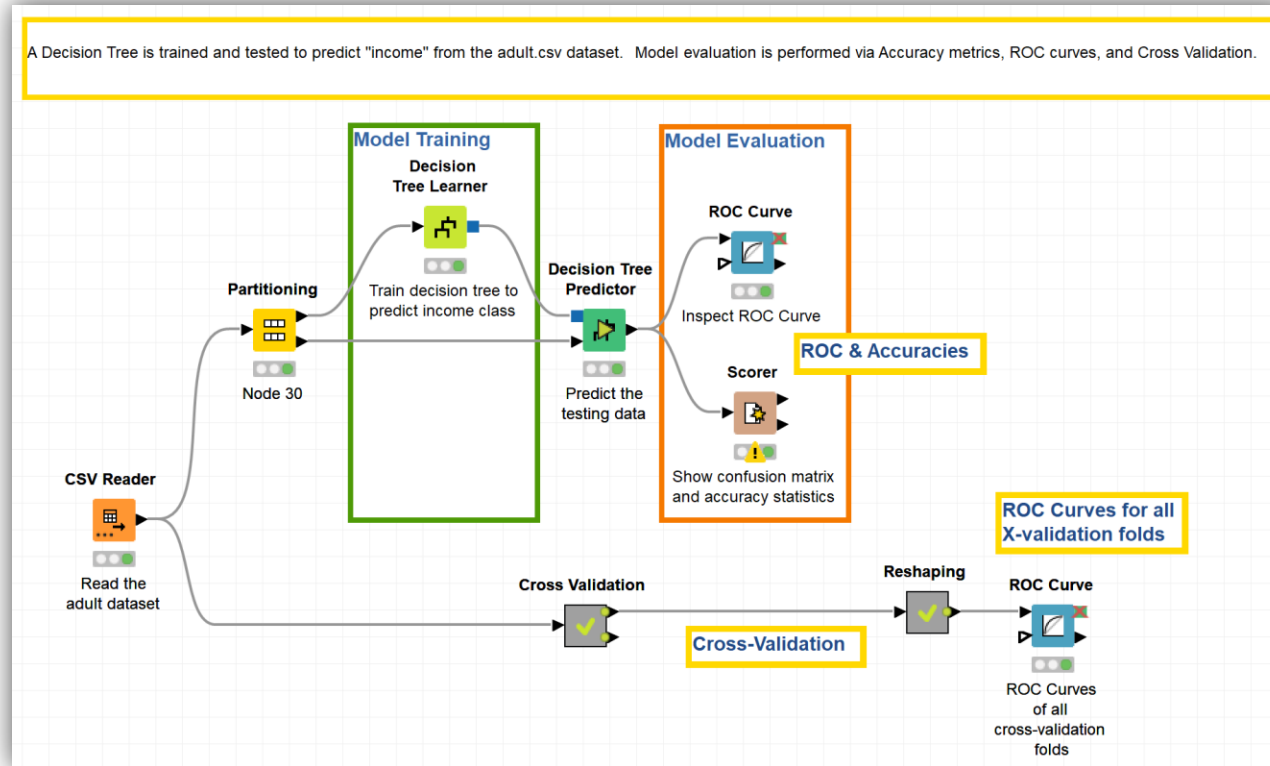
- Confusion matrix
- Class statistics measures
- Accuracy measures
- ROC curves
- Cohen's kappa
- Numeric error measures

- Model validation

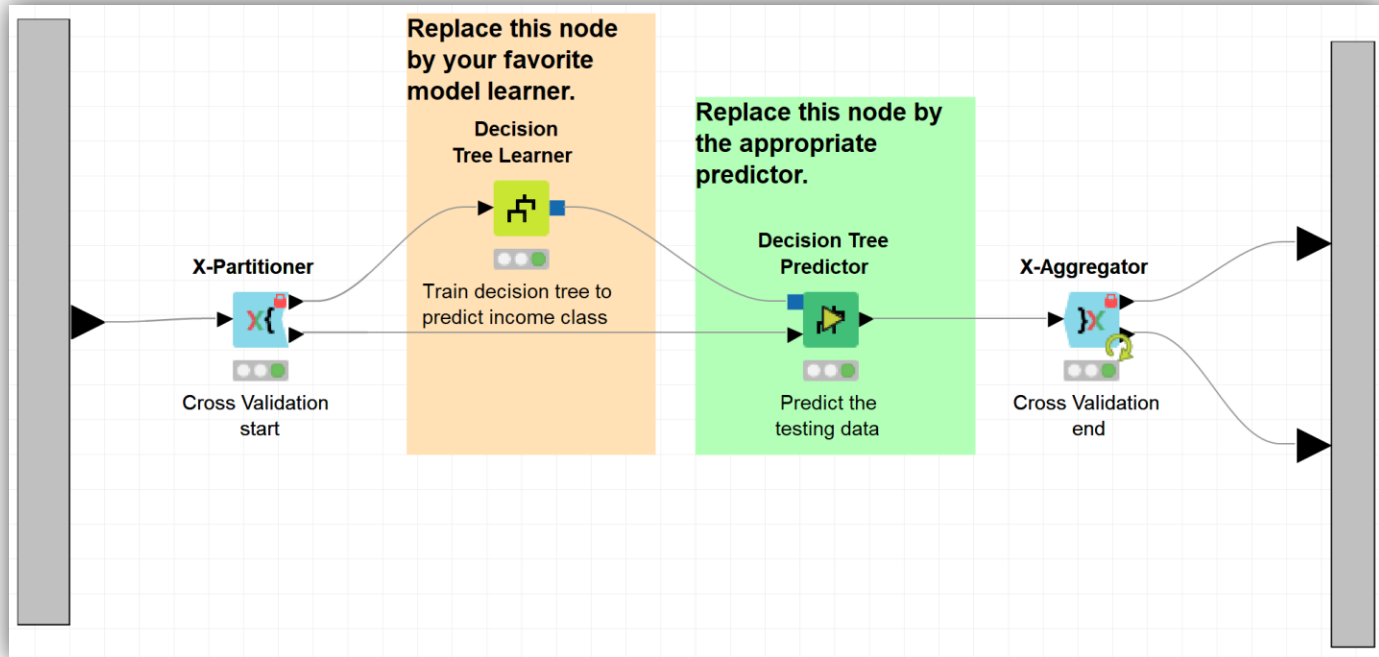
- Types of model errors
- Strategies for more reliable model evaluation
 - Cross-validation
 - Bootstrapping
 - Coping with unbalanced datasets
 - Measures for model complexity

Practical Examples with KNIME Analytics Platform

– Training and testing a classification model (decision tree)

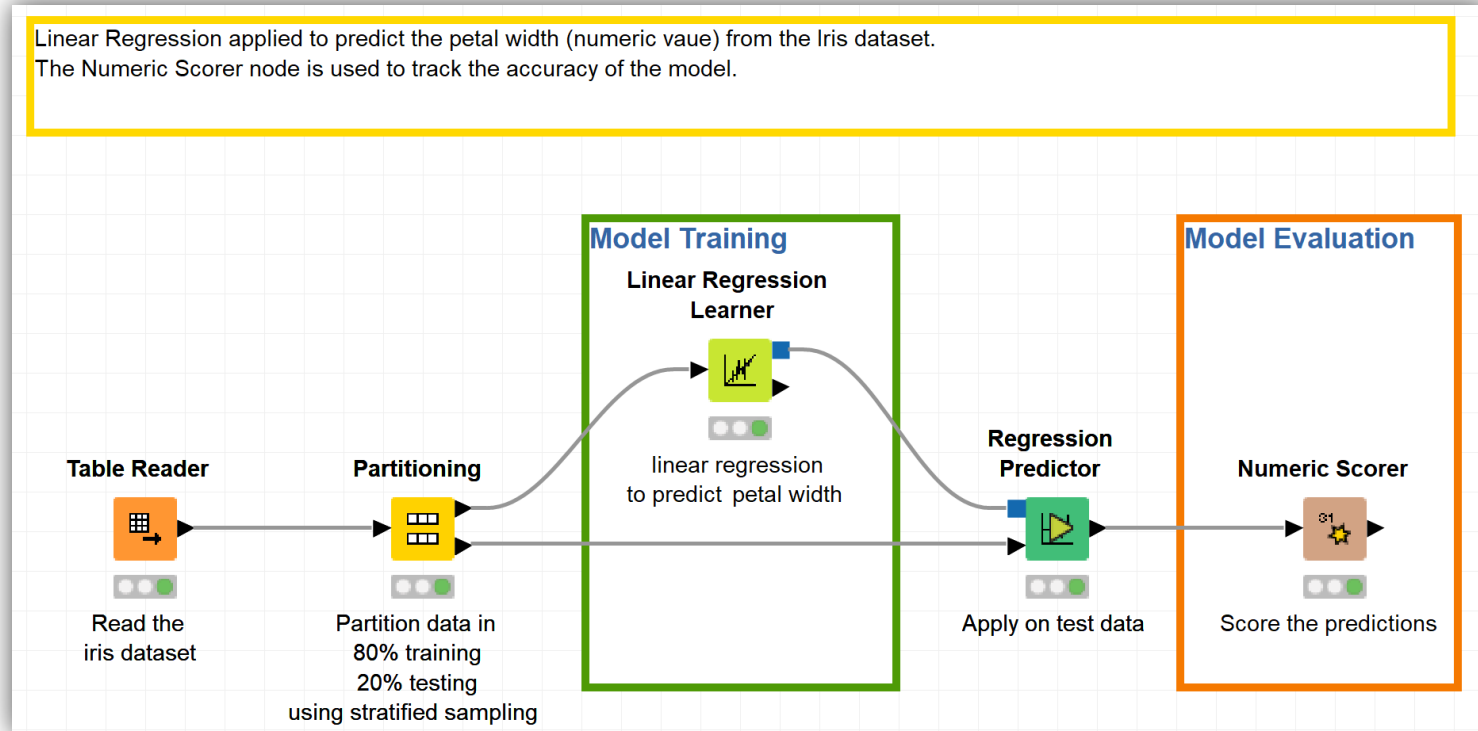


– Cross-validation metanode



– Numeric errors on a linear regression model

Linear Regression applied to predict the petal width (numeric value) from the Iris dataset.
The Numeric Scorer node is used to track the accuracy of the model.



Thank you